Друштво математичара Србије
Фондација Петља

Такмичења из програмирања за ученике основних школа

2019. година

Садржај

Глава 1

Сезона 2019/2020.

1.1 Први круг квалификација

Задатак: Књига

Поставка: Књига има N поглавља. Никола је првог дана прочитао A поглавља, а другог дана два поглавља више него првог. Написати програм који учитава целе позитивне бројеве N, A, и исписује колико поглавља је остало Николи да прочита.

Улаз: Са стандардног улаза се у првом реду уноси број N ($1 \leq N \leq 99$), а у другом реду број A ($1 \leq A \leq 99$). Улазни подаци су такви да преостали број поглавља никад није негативан.

Излаз: На стандардни излаз исписати један број, број поглавља која Никола још није прочитао.

Пример 1

Улаз	*Излаз*
15	9
2	

Пример 2

Улаз	*Излаз*
20	0
9	

Решење: Решење је сасвим једноставно. Ако је Никола првог дана прочитао A поглавља а другог дана $A+2$ поглавља, то значи да му остаје да прочита још $N-A-(A+2)$ поглавља.

```
#include <iostream>
using namespace std;

int main()
{
    int n, a;
    cin >> n >> a;
    cout << n - a - (a+2) << endl;
    return 0;
```

}

Задатак: Сијалица

Поставка: Стубови уличне расвете су нумерисани редом по улицама. У свакој улици има по N стубова, тако да су стубови у првој улици нумерисани $1, 2, \ldots N$, у другој су бројеви стубова $N + 1, N + 2, \ldots 2N$ итд. Мирко је добио задатак да у свакој другој улици замени сијалицу на сваком трећем стубу. Када је дошао до стуба са бројем A, Мирко се забројао. Написати програм који учитава целе позитивне бројеве N и A и одговара на питање да ли на том стубу треба заменити сијалицу.

Улаз: Са стандардног улаза се у првом реду уноси број N $(1 \leq N \leq 1000)$, а у другом реду број A $(1 \leq A \leq 1000)$.

Излаз: На стандардни излаз исписати само реч da ili ne.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
20	da	100	ne
75		300	

Решење: Када би се стубови и улице бројали од 0, тада би редни број улице могао да се добије као $A \operatorname{div} N$, а редни број стуба у улици као $A \bmod N$ (где су са div и $\bmod$ означени целобројни количник, тј. остатак при дељењу два цела броја). Међутим, пошто се у задатку броји од 1, учитану ознаку на стубу треба смањити за 1, а добијене резултате за редни број улице и стуба у улици повећати за 1. То значи да редни број улице треба рачунати као $(A - 1) \operatorname{div} N + 1$, а редни број стуба у улици као $(A - 1) \bmod N + 1$.

Након овог израчунавања потребно је још само проверити да ли је редни борј улице дељив са 2 и редни број стуба у улици дељив са 3.

```
#include <iostream>
using namespace std;

int main()
{
    int n, a;
    cin >> n >> a;
    int ulica = (a-1) / n + 1;
    int brUUlici = (a-1) % n + 1;
    if ((ulica % 2 == 0) and (brUUlici % 3 == 0))
        cout << "da" << endl;
    else
        cout << "ne" << endl;
    return 0;
}
```

Задатак: Запета

Поставка: Дат је низ ASCII карактера, међу којима се појављује тачно један знак "," (запета) и на самом крају тачно један знак "." (тачка). Сви остали знаци, ако их има, су слова енглеске абецеде, цифре, заграде и размаци и они могу да се понављају. Написати програм који учитава дати низ карактера, а на стандардни излаз исписује текст у коме су део до запете и део од запете разменили места, док тачка остаје на крају текста.

Улаз: На стандардном улазу се у једном реду налази низ од највише 100 ASCII карактера.

Излаз: На стандардни излаз исписати измењени низ карактера.

Пример 1

Улаз	*Излаз*
`Da si hteo, mogao si.`	`mogao si,Da si hteo.`

Пример 2

Улаз	*Излаз*
`,a b c.`	`a b c,.`

Пример 3

Улаз	*Излаз*
`x,.`	`,x.`

Решење: У овом задатку је најпогодније да се прочита цео ред улаза као један стринг (без обзира на то да ли стринг садржи размаке). Након учитавања можемо да издвојимо део A од почетка стринга до запете (не укључујући запету) а затим и део B од запете до тачке (не укључујући запету и тачку).

У језику C++ позицију карактера `c` унутар стринга `s` можемо одредити позивом `s.find(c)`, док део стринга `s` који почиње на позицији `p` и има `d` карактера можемо одредити позивом `s.substr(p, d)`.

После оваквог издвајања остаје још само да испишемо делове у траженом редоследу (део B, запета, део A и на крају тачка).

```
#include <iostream>
using namespace std;

int main()
{
    string s;
    getline(cin, s);
    int pozZapete = s.find(',');
    int pozTacke = s.find('.');
    string a = s.substr(0, pozZapete);
    string b = s.substr(pozZapete + 1, pozTacke - pozZapete - 1);
    cout << b << ”,” << a << ”.” << endl;
    return 0;
}
```

Задатак: Непливачи

Поставка: Деда Раде жели да упише својих четворо унука непливача у школу пливања. Инструктор му је рекао да је остао само један слободан термин, за који могу да се пријаве само деца виша од 110 сантиметара. Деда Раде не жели да раздваја унуке, па ће их уписати у школу пливања само ако сви испуњавају услов. Написати програм који одређује да ли деда Раде већ сада може да упише свих четворо унука у школу пливања.

Улаз: Са стандардног улаза се уносе четири цела позитивна броја не већа од 180, сваки у посебном реду – висине деда Радетових унука.

Излаз: На стандардни излаз исписати реч SVI ако је свако од четворо деце више од 110cm, а реч NIKO ако бар једно дете није више од 110 cm.

Пример 1

Улаз	*Излаз*
131	SVI
111	
128	
117	

Пример 2

Улаз	*Излаз*
131	NIKO
110	
128	
117	

Решење: Након учитавања потребно је само проверити да ли су сва 4 учитана броја већа од 110. Најједноставнији начин је употреба сложеног логичког услова.

```
#include <iostream>
using namespace std;

int main()
{
    int a, b, c, d;
    cin >> a >> b >> c >> d;
    if (a > 110 and b > 110 and c > 110 and d > 110)
        cout << "SVI" << endl;
    else
        cout << "NIKO" << endl;
    return 0;
}
```

Задатак: Сусрет

Поставка: Новак и Јагош су стари другари и они се често без договарања сачекују на свом омиљеном месту у неко уобичајено време, па ако први дочека другог онда проведу неко време дружећи се. Када Новак стигне први, он чека Јагоша 10 минута, а Јагош (ако он стигне први) чека Новака 15 минута.

Написати програм који за дата времена данашњег појављивања Новака и Јагоша одговара на питање да ли су се састали.

Улаз: Учитавају се четири цела броја, NS, NM, JS, JM редом, сваки у посебном реду стандардног улаза. Бројеви NS, NM представљају сат и минут Новаковог доласка, а JS, JM представљају сат и минут Јагошевог доласка. Подаци ће предста-

вљати исправно записана времена (то јест, важиће $0 \leq NS \leq 23$, $0 \leq NM \leq 59$, $0 \leq JS \leq 23$, $0 \leq JM \leq 59$).

Излаз: На стандардни излаз исписати само реч da ili ne.

Пример 1		**Пример 2**		**Пример 3**		**Пример 4**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
10	da	21	ne	0	da	0	ne
15		15		15		0	
10		21		0		23	
25		26		0		59	

Решење: Задатак се једноставније решава ако се после учитавања прво израчуна број минута од почетка дана до Николиног и Јагошевог доласка: $N = N_{sat} \cdot 60 + N_{min}$, и $J = J_{sat} \cdot 60 + J_{min}$.

Уїнежїена їранања

Решавање задатка сада можемо да довршимо тако што најпре проверимо ко је први стигао на место састанка. Ако је то Новак, онда проверавамо да је Јагош дошао највише 10 минута након Новака, а ако је Јагош први стигао, онда проверавамо да ли је Новак дошао највише 15 минута након Јагоша.

```
#include <iostream>
using namespace std;

int main()
{
  int nsat, nmin, jsat, jmin;
  cin >> nsat >> nmin >> jsat >> jmin;
  int n = nsat*60 + nmin;
  int j = jsat*60 + jmin;

  if (n<j) // ako je Novak dosao pre
    if (j <= n+10)
      cout << "da" << endl;
    else
      cout << "ne" << endl;
  else // ako je Jagos dosao pre (ili su dosli istovremeno)
    if (n <= j+15)
      cout << "da" << endl;
    else
      cout << "ne" << endl;
  return 0;
}
```

Сложени услов

Можемо и да формирамо сложени услов, који директно одговара на питање да ли је дошло до сусрета. Да би одговор био потврдан, треба да важи $N \leq J \leq N + 10$ или $J \leq N \leq j + 15$, у противном одговор је одречан.

```
#include <iostream>
using namespace std;

int main()
{
    int nsat, nmin, jsat, jmin;
    cin >> nsat >> nmin >> jsat >> jmin;
    int n = nsat*60 + nmin;
    int j = jsat*60 + jmin;

    if ((n <= j && j <= n+10) || (j <= n && n <= j+15))
      cout << "da" << endl;
    else
      cout << "ne" << endl;

    return 0;
}
```

Задатак: Јаја

Поставка: У сваку кутију за јаја може да стане K јаја. Када Јоца пакује јаја, он користи најмањи број кутија у које јаја могу да стану, али пошто је сујеверан, он никада не оставља у кутији (позитиван) број јаја дељив са 3.

Написати програм који одређује колико кутија треба Јоци да би спаковао J јаја.

Улаз: у првом реду стандардног улаза је број K, који је увек 10 или 15. У другом реду је цео број J, такав да је $0 \leq J \leq 100$.

Излаз: На стандардни излаз исписати један цео број, број кутија које ће Јоца употребити.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
10	2	15	1
18		10	

Решење: Прво што можемо да приметимо је да у кутије од 15 јаја можемо да ставимо највише 14 јаја. Према томе, оваквим кутијама можемо одмах након учитавања да смањимо величину за 1 и поједноставимо даље разматрање.

Погледајмо шта би се догодило када би се кутије пуниле редом до краја. Тада би број потребних кутија био $\lceil \frac{J}{K} \rceil$, где је J број јаја која треба спаковати, а K број јаја која стају у кутију.

Пошто након почетне исправке величина кутије K више није дељива са 3, то услов дељивости може да утиче само на последњу употребљену кутију. Зато још треба посебно испитати ситуацију када је број јаја у последњој употребљеној кутији дељив са 3. Таква сиутација може бити разрешена или узимањем нове кутије (само ако је неопходно), или пребацивањем једног или више јаја из претходно попуњених кутија у последњу. Додавање јаја у последњу кутију није могуће само у следећа два случаја:

(1) када нема ниједне претходно попуњене кутије из које бисмо "позајмили" јаја (то јест када је $J \leq K$).

(2) када у последњој кутији има места само за још једно јаје (то јест када је $J \bmod K = K - 1$). У овом случају додавање није могуће зато што би пребацивањем једног јајета из било које попуњене кутије у тој претходно попуњеној кутији остао број јаја дељив са 3.

У свим осталим случајевима се из претпоследње у последњу кутију могу пребацити бар једно или два јаја. На тај начин, број јаја у последњој кутији у сваком случају више неће бити дељив са 3, а од два могућа пребацивања (једно или два јаја) једно од тих пребацивања мора одговарати и претпоследњој кутији, тако да ни у њој број јаја не буде дељив са 3. Према томе, у свим случајевима осим два издвојена раније, број кутија се не мора повећавати.

```
#include <iostream>
using namespace std;

int main()
{
    int velKutije, brJaja;
    cin >> velKutije >> brJaja;
    if (velKutije % 3 == 0) velKutije--;

    int brKutija = (brJaja + velKutije - 1) / velKutije;
    int uPoslednjoj = brJaja % velKutije;

    // ako je u poslednjoj kutiji nezgodan broj
    if (uPoslednjoj > 0 && uPoslednjoj % 3 == 0)
      // ako nemamo odakle u nju da dodamo (poslednja kutija je jedina)
      // ili ako bi se dodavanjem pokvarila prethodna kutija
      if (brKutija == 1 || uPoslednjoj + 1 == velKutije)
        brKutija++;  // onda ce trebati kutija vise

    cout << brKutija << endl;
    return 0;
}
```

Задатак: Огрлица

Поставка: Сара чува перле у N кутијица нумерисаних бројевима од 1 до N. Она увек прави огрлицу тако што из сваке од N кутијица редом по бројевима узме по једну перлу и у истом редоследу наниже перле на конац.

Написати програм који одређује на колико начина Сара може да изабере перле за следећу огрлицу.

Улаз: у првом реду стандардног улаза је цео позитиван број N, број кутијица, не већи од 10. У сваком од следећих N редова је по један цео једноцифрен број, број перли у одговарајућој кутијици по реду.

Излаз: На стандардни излаз исписати један цео број, број начина на које Сара може да направи следећу огрлицу.

Пример 1

Улаз	*Излаз*
5 3 3 2 5 1	90

Пример 2

Улаз	*Излаз*
3 7 0 2	0

Решење: Пошто се перле могу комбиновати свака са сваком, број начина да се наниже огрлица је једнак производу броја перли из појединих кутија. Дакле, потребно је само помножити дате бројеве перли и исписати производ тих бројева.

```
#include <iostream>
using namespace std;

int main()
{
    int n, p;
    cin >> n;
    p = 1;
    for (int i = 0; i < n; i++)
    {
        cin >> a;
        p *= a;
    }
    cout << p << endl;
    return 0;
}
```

Задатак: Секција

Поставка: У школи је велико интересовање за програмерску секцију, на којој ће се правити рачунарске игре. Наставник је поставио услов да на секцију могу да иду само они ученици који имају 5 из информатике и бар 4 из математике.

Написати програм који одређује колико ученика може да се пријави за секцију.

Улаз: у првом реду број N, број ученика заинтересованих за секцију, природан број не већи од 200. У сваком од следећих N редова низ оцена једног од заинтересованих ученика из свих 11 предмета редом, без размака. Оцена из математике је шеста, а из информатике девета у низу од 11 цифара.

Излаз: На стандардни излаз исписати један цео број, број ученика који испуњавају услов.

Пример 1

Улаз	*Излаз*
4	2
55555355455	
55555455455	
55555455555	
22222522522	

Пример 2

Улаз	*Излаз*
2	0
55555555455	
55555355555	

Решење: Најједноставније је да учитавамо сваки ред као стринг од 11 карактера (цифара). Потребно је да пребројимо оне стрингове код којих је шеста цифра слева (бројећи од 1) већа од 3, а девета слева једнака 5.

Након пребројавања исписујемо вредност бројача.

```
#include <iostream>
using namespace std;

int main()
{
    int n, br;
    cin >> n;
    string s;
    br = 0;
    for (int i = 0; i < n; i++)
    {
        cin >> s;
        if (s[5] - '0' > 3 && s[8] - '0' == 5)
            br++;
    }
    cout << br << endl;
    return 0;
}
```

Задатак: Статистике33

Поставка: Чувени кошаркаш Дабло Трипловић има толико добре статистике, да се већ помало сумња да су његови успеси добро пребројани.

Написати програм који одређује колико пута је Трипловић постигао такозвани трипл–дабл.

Улаз: На стандардноим улазу се у првом реду налази цео број N, $(1 \leq N \leq 100)$, број утакмица које је Трипловић одиграо у сезони. У сваком од следећих N редова по 3 цела броја раздвојена по једним размаком: број поена, скокова и асистенција редом (ови бројеви нису већи од 1000).

Излаз: На стандардни излаз исписати један број, број утакмица на којима је Трипловић постигао трипл–дабл.

Напомена

За потребе овог задатка трипл–дабл дефинишемо као најмање двоцифрен учинак (10 или више) у све три категорије које се прате. Другим речима: сва три броја већа од 9 у једном реду улаза.

Пример

Улаз	*Излаз*
3 20 9 7 127 12 11 11 14 12	2

Решење: Потребно је пребројати редове улаза у којима су сва три броја већа од 9. Увешћемо бројач који пре петље постављамо на 0, а затим у петљи учитавамо по три броја, прверавамо да ли су сва три већа од 9, и ако јесу - увећавамо бројач.

По завршетку петље исписујемо вредност бројача.

```
#include <iostream>
using namespace std;

int main()
{
    int n, brTriplDabl = 0, a, b, c;
    cin >> n;
    for (int utak = 0; utak < n; utak++)
    {
        cin >> a >> b >> c;
        if (a > 9 && b > 9 && c > 9)
            brTriplDabl ++;
    }
    cout << brTriplDabl << endl;
    return 0;
}
```

Задатак: Статистике35

Поставка: Чувени кошаркаш Дабло Трипловић има толико добре статистике, да се већ помало сумња да су његови успеси добро пребројани.

Написати програм који одређује колико пута је Трипловић постигао такозвани трипл–дабл.

Улаз: Са стандардног улаза се у првом реду уноси цео број N $(1 \leq N \leq 100)$, број утакмица које је Трипловић одиграо у сезони. У сваком од следећих N редова је по 5 целих бројева раздвојених по једним размаком: број поена, скокова, асистенција, блокада и украдених лопти редом (ови бројеви нису већи од 1000).

Излаз: На стандардни излаз исписати један цео број, број утакмица на којима је Трипловић постигао трипл–дабл.

Напомена

За потребе овог задатка трипл–дабл дефинишемо као најмање двоцифрен учинак (10 или више) у бар три од пет категорија које се прате. Другим речима: бар три броја већа од 9 у једном реду улаза.

Пример

Улаз	*Излаз*
3	2
20 9 7 2 1	
127 12 11 3 0	
0 10 11 14 12	

Решење: У овом задатку треба пребројати редове улаза у којима су бар три од пет бројева већи од 9. Да бисмо установили да ли неки ред улаза треба бројати, треба у сваком реду пребројати елементе који су већи од 9. То значи да ћемо у овом задатку применити технику пребројавања "на два нивоа": глобално бројимо редове који испуњавају услов, а у сваком реду бројимо елементе веће од 9.

Формираћемо двоструку петљу: спољна петља ће ићи по утакмицама (тј. редовима улаза), а унутрашња по категоријама које се прате на свакој утакмици. Бројач трипл-даблова иницијализујемо пре спољне петље, а бројач двоцифрених учинака пре унутрашње.

```
#include <iostream>
using namespace std;

int main()
{
    int n, brTriplDabl = 0;
    int a[5];
    cin >> n;
    for (int utak = 0; utak < n; utak++)
    {
        cin >> a[0] >> a[1] >> a[2] >> a[3] >> a[4];
        int brDvoc = 0;
        for (int kateg = 0; kateg < n; kateg++)
            if (a[kateg] > 0)
                brDvoc++;

        if (brDvoc >= 3)
            brTriplDabl++;
    }
    cout << brTriplDabl << endl;
    return 0;
}
```

Додатна решења овог задатка су доступна на страни 24.

Задатак: Близанци

Поставка: Марија и Петар су близанци и желимо да свакоме од њих двоје купимо по једно одело као поклон за рођендан, али тако да се цене та два поклона што мање разликују (при томе није битно чији поклон ће бити скупљи).

Написати програм који учитава цене свих женских одела и свих мушких одела, а одређује и исписује најмању разлику између цена женског и мушког одела.

Улаз: Опис улаза: са стандардног улаза се учитава:

- у првом реду број мушких одела m ($1 \leq m \leq 50000$),
- у другом реду m целих бројева (цели бројеви између 1 и $2 \cdot 10^9$ раздвојени по једним размаком) - цене мушких одела
- у трећем реду број женских одела z ($1 \leq z \leq 50000$)
- и у четвртом реду z целих бројева (цели бројеви између 1 и $2 \cdot 10^9$ раздвојени по једним размаком) - цене женских одела.

Излаз: На стандардни излаз исписати најмању вредност разлике цена мушког и женског одела.

Пример

Улаз	*Излаз*
5 4680 2120 7940 11530 17820 4 850 13420 5770 6300	1090

Објашњење

Најмања разлика се постиже када се купе одела чије су цене 4680 и 5770 динара.

Решење:

Анализа

Наивно решење

Један могући приступ је да одредимо разлику (прецизније, апсолутну вредност разлике) у цени између сваког мушког и сваког женског одела, па од тих разлика нађемо најмању. Овакво решење ће дати тачан резултат у мањим примерима, али на већим примерима се неће извршити на време.

```
#include <iostream>
#include <vector>
#include <cmath>
#include <limits>
using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  int n1; cin >> n1;
```

```
  vector<int> a1(n1);
  for (int i = 0; i < n1; i++)
    cin >> a1[i];
  int n2; cin >> n2;
  vector<int> a2(n2);
  for (int i = 0; i < n2; i++)
    cin >> a2[i];

  int minRazlika = numeric_limits<int>::max();
  for (int i1 = 0; i1 < n1; i1++)
    for (int i2 = 0; i2 < n2; i2++)
      minRazlika = min(minRazlika, abs(a1[i1] - a2[i2]));

  cout << minRazlika << endl;

  return 0;
}
```

Упоредни пролаз кроз уређене низове

Ефикаснији приступ је да се низови цена најпре сортирају, а да се затим истовремено пролази кроз оба низа, рачунајући разлику текућих елемената и напредујући у оном низу у којем је цена тренутно мања. Успут се, наравно, по потреби ажурира најмања забележена разлика. Када се стигне до краја било којег низа, поступак је завршен и најмања забележена разлика је тада и укупно најмања.

Заиста, пошто су низови сортирани, када се упореде почетни елементи из оба низа, онај који је мањи од њих нема потребе упоређивати са осталим елементима низа коме он не припада, јер ће разлика моћи бити само већа (јер је тај низ сортиран). Тај елемент онда можемо избацити из даљег разматрања тако што ћемо у низу у ком се он налази прећи на следећи елемент. У специјалном случају када су почетни елементи оба низа једнаки, разлика је једнака нули, што је најмања могућа разлика, па нема потребе вршити даљу анализу.

На пример, нека су након сортирања вредности једнаке следећим.

```
1 14 28 33 45
8 21 22 41 56 68
```

- Прво поредимо елементе 1 и 8. Разлика је 7. Разлика између броја 1 и свих даљих бројева у доњем низу је већа од 7, па број 1 не морамо више анализирати.
- Након тога поредимо бројеве 14 и 8 и добијамо разлику 6. Разлика између броја 8 и свих бројева иза 14 је већа, па сада ни 8 не морамо више анализирати.
- Поредимо сада бројеве 14 и 21, разлика је 7, а 14 не морамо више да анализирамо.
- И разлика између 28 и 21 је 7, а број 21 не морамо више да анализирамо.
- Разлика између 28 и 22 је 6, а 22 не морамо да анализирамо даље.
- Разлика између 28 и 41 је 13, а 28 не морамо да анализирамо даље.

- Разлика између 33 и 41 је 8, а 33 не морамо да анализирамо даље.
- Разлика између 45 и 41 је 4, а 41 не морамо да анализирамо даље.
- Разлика између 45 и 56 је 11, а 45 не морамо да анализирамо даље. Пошто нема више елемената у горњем низу, поступак се завршава.

Можемо закључити да је најмања могућа разлика једнака 4 (за бројеве 41 и 45).

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <limits>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  int n1; cin >> n1;
  vector<int> a1(n1);
  for (int i = 0; i < n1; i++)
    cin >> a1[i];
  int n2; cin >> n2;
  vector<int> a2(n2);
  for (int i = 0; i < n2; i++)
    cin >> a2[i];

  sort(begin(a1), end(a1));
  sort(begin(a2), end(a2));
  int i1 = 0, i2 = 0;

  int minRazlika = numeric_limits<int>::max();
  while (i1 < n1 && i2 < n2)
    if (a1[i1] <= a2[i2]) {
      minRazlika = min(minRazlika, a2[i2] - a1[i1]);
      i1++;
    } else {
      minRazlika = min(minRazlika, a1[i1] - a2[i2]);
      i2++;
    }

  cout << minRazlika << endl;

  return 0;
}
```

Задатак: Поклони за родитеље

Поставка: Ненад жели да са путовања донесе поклоне својим родитељима. Пошто путује нискотарифном авио-компанијом и не жели да плаћа додатне трошкове, ограничена је маса коју може да понесе у свом ранцу. Оцу ће купити поклон у једној, а мајци у другој продавници. Написати програм који на основу познатих маса и цена свих поклона у обе продавнице помаже Ненаду да сваком од родитеља купи по један поклон, тако да поклони збирно буду што вреднији и да заједно не прелазе дато ограничење масе.

Улаз: са стандардног улаза се уносе следећи подаци: У првом реду број N_1 ($1 \leq N_1 \leq 10^5$), број производа у првој продавници. У сваком од наредних N_1 редова по два цела броја, сваки између 1 и 10^9, који представљају масу и затим цену једног производа из прве продавнице. У следећем реду број N_2 ($1 \leq N_2 \leq 10^5$), број производа у другој продавници. У сваком од наредних N_2 редова по два цела броја, сваки између 1 и 10^9, који представљају масу и затим цену једног производа из друге продавнице.

У следећем и последњем реду цео број између 1 и $2 \cdot 10^9$, највећа дозвољена маса за пар поклона заједно.

Излаз: На стандардни излаз исписати највећи могући збир цена првог и другог поклона које је могуће понети.

Пример

Улаз	*Излаз*
3 2 4 3 6 4 5 3 2 3 3 2 4 5 6	9

Решење:

Наивно решење

Можемо да проверимо сваки пар предмета (у коме је један предмет из једне, а други из друге продавнице). Ово решење је квадратне сложености, па може да донесе поене само за мале примере.

```
#include <iostream>
#include <vector>
#include <utility>
using namespace std;

typedef pair<int, int> Proizvod;
vector<Proizvod> ucitajProizvode() {
  int n;
  cin >> n;
```

```
  vector<pair<int, int>> proizvodi(n);
  for (int i = 0; i < n; i++) {
    int cena, tezina;
    cin >> cena >> tezina;
    proizvodi[i] = make_pair(cena, tezina);
  }
  return proizvodi;
}

int main() {
  vector<Proizvod> proizvodi1 = ucitajProizvode();
  vector<Proizvod> proizvodi2 = ucitajProizvode();
  int maksTezina;
  cin >> maksTezina;

  int maksCena = 0;
  for (const auto& p1 : proizvodi1) {
    if (p1.first >= maksTezina) continue;
    for (const auto& p2 : proizvodi2)
      if (p1.first + p2.first <= maksTezina &&
          p1.second + p2.second > maksCena)
        maksCena = p1.second + p2.second;
  }
  cout << maksCena << endl;
}
```

Нагађање

Такмичар који не уме да реши задатак, може да покуша да "упеца" неки број поена тако што напише једноставнији програм, који у суштини нагађа резултат. Нагађање може бити мање или више успешно.

Један могући покушај је да се нађе највећа цена из сваке продавнице и да се испише збир тих највећих цена.

Овај покушај нагађања се може комбиновати са наивним (неефикасним) решењем, тако да за довољно мале дужине низова задатак решавамо егзактно квадратним алгоритмом, а за велике примере покушамо да погодимо решење (што обично не даје добре резултате).

```
#include <iostream>

using namespace std;

int main() {
  int n1;
  cin >> n1;
  int maks1 = 0;
  for (int i = 0; i < n1; i++) {
    int tezina, cena;
```

```
    cin >> tezina >> cena;
    if (cena > maks1)
      maks1 = cena;
  }
  int n2;
  cin >> n2;
  int maks2 = 0;
  for (int i = 0; i < n2; i++) {
    int tezina, cena;
    cin >> tezina >> cena;
    if (cena > maks2)
      maks2 = cena;
  }

  cout << maks1 + maks2 << endl;

  return 0;
}
```

Сортирање и максимуми префикса

Решење које претендује на максималан број поена треба да буде ефикасније од квадратног. У ту сврху увек је корисно да подаци буду на неки начин сортирани. Питање је: како сортирати?

Ако оба низа уредимо тако да масе расту, можемо у једном низу да кренемо од почетка (од најлакшег предмета) а у другом од краја. На тај начин брзо долазимо до парова предмета који су у збиру испод лимита масе, а при томе најближи том лимиту. Ипак, ово није довољно да се задатак реши ефикасно, јер масе и цене не расту заједно, па предмет мање масе може бити вреднији. То значи да при прегледању свих парова који су кандидати за најбољи пар не бисмо избегли квадратно време.

Уређивање по цени није боље, јер бисмо тада за фиксиран предмет из једне продавнице морали међу онима из друге (који у збиру са првим) поправљу укупну вредност, да тражимо оне који се уклапају по маси и опет спадамо на квадратно време.

Оно што би нам овде помогло (након сортирања оба низа предмета по маси) је да можемо за сваки предмет P из једног низа да брзо одговоримо колико вреди највреднији предмет тог низа, чија маса није већа од масе предмета P. Такве одговоре можемо да припремимо након сортирања а пре испитивања парова, тако што фромирамо још један низ у коме ће се ти одговори наћи. Конкретније, можемо у једном пролазу кроз низ сортиран по маси да за сваки префикс тог низа израчунамо цену највреднијег предмета у префиксу. Након тога првобитна идеја о истовременом проласку кроз један низ од почетка а кроз други од краја доводи до ефикасног решења.

Прикажимо рад овог алгоритма на једном примеру. Нека су цене и тежине мушких поклона дате у левој, а женских у десној колони, при чему смо поклоне сортирали по тежинама и нека је капацитет ранца једнак 20.

```
4   3              5   6          6
7   9              9   4          6
```

```
12  4                10  12        12
16  11               14  9         12
19  2                18  7         12
```

Покушавамо да за први мушки поклон (то је (4, 3)) пронађемо скуп поклона који се могу са њим упарити. Пошто је други низ сортиран, то ће бити неки поклони који се налазе на почетку тог низа. Крећемо од краја, елиминишемо последњи женски поклон јер је збир 4 + 18 већи од 20 и проналазимо да је последњи женски поклон који се може упарити са првим мушким онај који има масу 14. Потребно је пронаћи најскупљи женски поклон који има масу мању или једнаку 14. У томе нам помаже последња колона у којој су записани максимуми префикса. За поклон чија је маса 14 можемо очитати вредност 12, што значи да постоји неки женски поклон чија је маса мања или једнака 14 који има вредност 12 (то је поклон (10, 12)). Дакле, ако купимо први мушки поклон, тада је највећа цена коју можемо постићи једнака 3 + 12 = 15.

Прелазимо на други мушки поклон (то је (7, 9)) и одређујемо женске поклоне са којима се он може комбиновати. Веома важна напомена је то да се поклони који се нису могли укомбиновати са претходним мушким поклонима, не могу укомбиновати ни са текућим (јер је он још тежи од претходних). Дакле, довољно је само међу оним поклонима који су се могли укомбиновати са претходним поклоном наћи оне који се могу укомбиновати са текућим. Пошто је низ женских поклона сортиран по тежини, анализирамо и евентуално елиминишемо елементе са његовог краја (тј. краја оног дела низа где смо се претходно зауставили). Поклон масе 14 се не може комбиновати са поклоном масе 7 (јер им је укупна маса 21 већа од носивости ранца 20). Најтежи женски поклон који се може упарити са оним мушким масе 7 је онај чија је маса 10. Поново на основу низа максимума префикса очитавамо да је највреднији женски поклон чија маса не прелази 10 онај чија је вредност 12 (то је опет (10, 12)), па се његовим комбиновањем са мушким поклоном (7, 9) добија вредност 12 + 9 = 21, што је боље од претходне.

Прелазимо на следећи мушки поклон (то је (12, 4)), елиминишемо женске поклоне (10, 12) и (9, 4) јер се они не могу комбиновати са мушким поклоном масе 12 и закључујемо да је једини женски поклон који се може комбиновати са (12, 4) поклон (5, 6). Тиме добијамо вредност 4+6 = 10, што је лошије од претходне.

Преласком на следећи мушки поклон (то је (16, 11)), елиминишемо и женски поклон (5, 6) и закључујемо да се ни један мушки поклон који је тежак 16 или више не може укомбиновати ни са једним женским поклоном.

Дакле, оптимално упаривање је упаривање поклона (7, 9) и (10, 12).

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;


typedef pair<int, int> Proizvod;
```

```
vector<Proizvod> ucitajProizvode() {
  int n;
  cin >> n;
  vector<pair<int, int>> proizvodi(n);
  for (int i = 0; i < n; i++) {
    int cena, tezina;
    cin >> cena >> tezina;
    proizvodi[i] = make_pair(cena, tezina);
  }
  return proizvodi;
}

int main() {
  vector<Proizvod> proizvodi1 = ucitajProizvode();
  vector<Proizvod> proizvodi2 = ucitajProizvode();
  int maksTezina;
  cin >> maksTezina;

  sort(begin(proizvodi1), end(proizvodi1));
  sort(begin(proizvodi2), end(proizvodi2));
  int n1 = proizvodi1.size(), n2 = proizvodi2.size();

  vector<int> maksCenaDo2(n2 + 1);
  maksCenaDo2[0] = 0;
  for (int i = 1; i <= n2; i++)
    maksCenaDo2[i] = max(maksCenaDo2[i-1], proizvodi2[i-1].second);

  int maksCena = 0;
  int i = 0, j = n2-1;
  while (i < n1 && proizvodi1[i].first < maksTezina) {
    while (j >= 0 && proizvodi1[i].first + proizvodi2[j].first > maksTezina)
      j--;
    if (j >= 0)
      maksCena = max(maksCena, proizvodi1[i].second + maksCenaDo2[j+1]);
    i++;
  }
  cout << maksCena << endl;

  return 0;
}
```

Сортирање, максимуми префикса и бинарна претрага

Претходна идеја се може мало и изменити. Довољно је да сортирамо само други низ и израчунамо максимуме његових префикса, као у претходном решењу. Након тога можемо за сваки предмет неуређеног првог низа да бинарном претрагом нађемо предмет највеће масе из другог низа, који се може понети заједно са првим, а онда (користећи максимуме префикса другог низа) да нађемо и вредност највреднијег предмета у

другом низу, који се може понети заједно са текућим предметом из првог низа.

```
#include <iostream>
#include <utility>
#include <algorithm>
using namespace std;

typedef pair<int, int> Proizvod;

vector<Proizvod> ucitajProizvode() {
  int n;
  cin >> n;
  vector<pair<int, int>> proizvodi(n);
  for (int i = 0; i < n; i++) {
    int cena, tezina;
    cin >> cena >> tezina;
    proizvodi[i] = make_pair(cena, tezina);
  }
  return proizvodi;
}

int main() {
  // ucitavamo podatke o proizvodima i maksimalnu tezinu koja moze da
  // stane u ranac
  vector<Proizvod> proizvodi1 = ucitajProizvode();
  vector<Proizvod> proizvodi2 = ucitajProizvode();
  int maksTezina;
  cin >> maksTezina;

  // sortiramo proizvode iz druge prodavnice po tezini
  sort(begin(proizvodi2), end(proizvodi2));

  // za svaku poziciju u sortiranom nizu proizvoda iz druge prodavnice
  // odredjujemo maksimalnu cenu proizvoda striktno pre te pozicije
  vector<int> maksCenaDo(proizvodi2.size() + 1);
  maksCenaDo[0] = 0;
  for (size_t i = 1; i <= proizvodi2.size(); i++)
    maksCenaDo[i] = max(maksCenaDo[i-1], proizvodi2[i-1].second);

  // maksimalni zbir cena dva proizvoda
  int maksCena = 0;
  // analiziramo jedan po jedan proizvod iz prve prodavnice
  for (const auto& p1 : proizvodi1) {
    // odredjujemo najvecu poziciju pre koje se svi proizvodi iz druge
    // pozicije mogu staviti u ranac sa tekucim proizvodom iz prve
    // prodavnice
    auto p2Granica = upper_bound(begin(proizvodi2), end(proizvodi2),
                                 maksTezina - p1.first,
```

```
                                    [](int tezina, const Proizvod& pj) {
                                      return tezina < pj.first;
                                    });
    int pj = distance(begin(proizvodi2), p2Granica);
    // ako ima bar neki takav proizvod
    if (pj > 0)
      // kombinujemo tekuci proizvod iz prve prodavnice sa najskupljim
      // proizvodom iz druge pre te pozicije (on se sigurno moze
      // iskombinovati sa tekucim proizvodom iz prve prodavnice)
      maksCena = max(maksCena, p1.second + maksCenaDo[pj]);
  }

  // ispisujemo pronadjeni zbir
  cout << maksCena << endl;

  return 0;
}
```

1.2 Други круг квалификација

Задатак: Кусур

Поставка: Андрија и Бојан су у кафићу попили по један (исти) сок. Андрија је дао a, а Бојан b динара, а добили су кусур од k динара. Како треба да поделе кусур да би једнако платили?

Улаз: Са стандардног улаза се учитавају три цела позитивна броја, a, b и k редом, сваки у посебном реду. Сва три броја су мања или једнака 1000 и таква да постоји целобројно решење.

Излаз: На стандардни излаз исписати два броја, сваки у посебном реду. Први број је део кусура који треба да добије Андрија, а други број је Бојанов део кусура.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
80	0	500	390
100	20	150	40
20		430	

Решење: Два сока су коштала $a + b - k$, а један сок $s = \frac{a+b-k}{2}$ динара. Према томе, Андрија треба да добије $a - s$ а Бојан $b - s$ динара.

```
#include <iostream>

using namespace std;

int main() {
  int a, b, k;
  cin >> a >> b >> k;
```

```
  int sok = (a + b - k) / 2;
  cout << a - sok << endl;
  cout << b - sok << endl;
}
```

Задатак: Деца

Поставка: Сања има тачно онолико година колико и њено троје деце заједно. Дате су године Сање и њено двоје деце у произвољном редоследу. Одредити године трећег Сањиног детета.

Улаз: Са стандардног улаза се учитавају три цела позитивна броја који нису већи од 100, сваки у посебном реду. Један од бројева представља Сањине године, а остала два године двоје од њене деце.

Излаз: На стандардни излаз исписати један број, број година трећег Сањиног детета.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
17	14	15	15
43		20	
12		50	

Решење: Од три учитана броја прво треба одредити највећи - то је број година маме Сање. Збир година два детета можемо да одредимо тако што од збира сва три броја одузмемо мамин број година. Коначно, године трећег детета добијамо када од мамиког броја година одузмемо збир година два детета.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int a, b, c;
  cin >> a >> b >> c;
  int mama = max({a, b, c});
  int prva_dva = a + b + c - mama;
  int trece = mama - prva_dva;
  cout << trece << endl;
}
```

Задатак: Кутије

Поставка: Дате су две кутије за ципеле облика квадра. Свака кутија је дата својом дужином, ширином и висином. Испитати да ли се нека од ових кутија може убацити у другу. Једна кутија се може убацити у другу ако се може окренути тако да јој одговарајуће димензије буду строго мање од одговарајућих димензија друге кутије.

Улаз: У првом реду стандардног улаза налазе се три цела броја d_1, s_1 и v_1 раздвојени по једним размаком, дужина, ширина и висина прве кутије. У другом реду стандард-

ног улаза налазе се три цела броја d_2, s_2 и v_2 такође раздвојени по једним размаком, дужина, ширина и висина друге кутије. Сви бројеви су позитивни и мањи од 1000.

Излаз: На стандардни излаз исписати `DA` ако се било која од кутија може убацити у другу, а `NE` иначе.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`5 9 2`	`DA`	`3 4 5`	`NE`	`3 4 5`	`DA`
`3 6 10`		`3 4 5`		`2 3 4`	

Решење: Нека су $a_1 \leq b_1 \leq c_1$ димензије прве, а $a_2 \leq b_2 \leq c_2$ димензије друге кутије. Да не бисмо проверавали све могуће оријентације кутија, доказаћемо да прва кутија може да стане у другу ако и само ако је $a_1 \leq a_2, b_1 \leq b_2, c_1 \leq c_2$. Смер $\Leftarrow$ овог тврђења је очигледан, па је довољно доказати смер $\Rightarrow$. Претпоставимо да нису испуњена сва три услова $a_1 \leq a_2, b_1 \leq b_2, c_1 \leq c_2$.

Ако није $a_1 \leq a_2$, онда $c_1 \geq b_1 \geq a_1 \geq a_2$, па ни једна димензија прве кутије није мања од a_2 и прва кутија не може да стане у другу.

Ако није $b_1 \leq b_2$, онда $c_1 \geq b_1 \geq b_2 \geq a_2$, па пошто не могу обе ивице c_1 и b_1 да се поставе паралелно са c_2, то прва кутија ни у овом случају не може да стане у другу.

Ако није $c_1 \leq c_2$, онда $c_1 \geq c_2 \geq b_2 \geq a_2$, па c_1 није мање ни од једне стране друге кутије и поново прва кутија не може да стане у другу. Овим је доказ завршен.

Према томе, да бисмо проверили да ли прва кутија може да стане у другу, довољно је да након сортирања ивица сваке кутије проверимо да важе сва три услова $a_1 \leq a_2, b_1 \leq b_2, c_1 \leq c_2$ (ако прва кутија не може да стане у другу када су овако окренуте, онда не може да стане ни на који начин).

Наравно, из истог разлога важи да друга кутија може да стане у прву само ако је $a_2 \leq a_1, b_2 \leq b_1, c_2 \leq c_1$.

На питање из задатка дајемо потврдан одговор било да је $a_1 \leq a_2, b_1 \leq b_2, c_1 \leq c_2$ или $a_2 \leq a_1, b_2 \leq b_1, c_2 \leq c_1$, а ако не једна од од ових тројки услова није испуњена, одговор ће бити одречан.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int a[3];
  int b[3];
  cin >> a[0] >> a[1] >> a[2];
  cin >> b[0] >> b[1] >> b[2];
  sort(a, a+3);
  sort(b, b+3);
  if ((a[0] < b[0] && a[1] < b[1] && a[2] < b[2]) ||
      (b[0] < a[0] && b[1] < a[1] && b[2] < a[2]))
```

```
    cout << "DA" << endl;
  else
    cout << "NE" << endl;
  return 0;
}
```

Задатак: Статистике35

Поставка: *Овај задатак је поновљен.* *Види текст задатка на страни 10.*

Решење: У овом задатку треба пребројати редове улаза у којима су бар три од пет бројева већи од 9. Да бисмо установили да ли неки ред улаза треба бројати, треба у сваком реду пребројати елементе који су већи од 9. То значи да ћемо у овом задатку применити технику пребројавања "на два нивоа": глобално бројимо редове који испуњавају услов, а у сваком реду бројимо елементе веће од 9.

Формираћемо двоструку петљу: спољна петља ће ићи по утакмицама (тј. редовима улаза), а унутрашња по категоријама које се прате на свакој утакмици. Бројач трипл-даблова иницијализујемо пре спољне петље, а бројач двоцифрених учинака пре унутрашње.

```
#include <iostream>
using namespace std;

int main()
{
    int n, brTriplDabl = 0;
    int a[5];
    cin >> n;
    for (int utak = 0; utak < n; utak++)
    {
        cin >> a[0] >> a[1] >> a[2] >> a[3] >> a[4];
        int brDvoc = 0;
        for (int kateg = 0; kateg < n; kateg++)
            if (a[kateg] > 0)
                brDvoc++;

        if (brDvoc >= 3)
            brTriplDabl++;
    }
    cout << brTriplDabl << endl;
    return 0;
}
```

Задатак: Трансформације

Поставка: Трансформацијом броја x зовемо замену броја x бројем $x \cdot x + 1$. Одредити троцифрени завршетак броја који се добија после n трансформација броја a.

Улаз: Са стандардног улаза се у првом реду уноси број a $(2 \leq a \leq 9)$, а у другом реду број n $(10 \leq n \leq 1000)$.

Излаз: На стандардни излаз исписати три цифре без размака, цифре којима се завршава број a^n.

Пример 1

Улаз	*Излаз*
5	802
11	

Пример 2

Улаз	*Излаз*
2	026
403	

Решење:

Анализа

Број a је трансформисањем веома брзо увећава, тако да већ после неколико трансформација даље рачунање или није тачно (због прекорачења) или је врло споро (због величине броја).

Пошто су нам потребне само последње три цифре трансформисаног броја, користићемо модуларну аритметику. Уместо са $a \cdot a + 1$, замењиваћемо a са $(a \cdot a + 1) mod 1000$. Нови резултат ће бити конгруентан старом по модулу 1000, па се зато при овој измени последње три цифре резулатата неће променити.

Након израчунавања трансформисаног броја по модулу 1000 треба још повести рачуна да се испишу и водеће нуле ако их има.

```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    int a, n;
    cin >> a >> n;
    for (int i = 0; i < n; i++)
        a = (a * a + 1) % 1000;
    cout << setfill('0') << setw(3) << a << endl;
    return 0;
}
```

Задатак: Рутер

Поставка: Дуж једне улице су равномерно распоређене зграде (растојање између сваке две суседне је једнако). За сваку зграду је познат број корисника које нови добављач интернета треба да повеже. Одредити у коју од зграда треба поставити рутер тако да би укупна дужина оптичких каблова којим се сваки од корисника повезује са рутером била минимална (рачунати само дужину каблова од зграде до зграде и занемарити дужине унутар зграда).

Улаз: У првом реду стандардног улаза налази се број n $(0 \leq n \leq 50000)$, а у наредном n природних бројева раздвојених размацима који представљају број корисника у свакој од n зграда.

Излаз: На стандардни излаз исписати минималну дужину каблова.

Пример

Улаз	*Излаз*
`6` `3 5 1 6 2 4`	`30`

Решење: Наивно решење би подразумевало да се израчуна дужина каблова за сваку могућу позицију рутера и да се одабере најмањи. Да бисмо израчунали дужину каблова, ако је рутер у згради на позицији k, рачунамо заправо збир

$$\sum_{i=0}^{n-1} |k-i| \cdot a_i,$$

где је a_i број корисника у згради i. Тај тежински збир можемо израчунати у времену $O(n)$, па пошто се испитује n позиција, алгоритам би био сложености $O(n^2)$.

Много боље решење и линеарни алгоритам можемо добити ако применимо принцип инкременталности и избегнемо рачунање у сваком кораку из почетка. Размотримо како се дужина каблова мења када се рутер помера са зграде k на зграду $k+1$.

Дужину каблова за рутер у згради $k+1$ добијамо од дужине каблова за рутер у згради k тако што ту дужину увећамо за укупан број станара закључно са зградом k и умањимо је за укупан број станара почевши од зграде $k+1$. То је заправо интуитивно прилично јасно и без компликованог математичког извођења. Померањем рутера за дужину једне зграде надесно, сваком станару који живи закључно до зграде k дужина кабла се повећала за једно растојање између зграда, а свим станарима од зграде $k+1$ надесно се та дужина смањује за једно растојање између зграда.

Укупне бројеве станара пре и после дате зграде можемо такође рачунати инкрементално (при преласку на наредну зграду, први број се увећава, а други умањује за број станара текуће зграде).

Дакле, у програму можемо да памтимо три ствари: дужину каблова d_k ако је рутер на позицији k, укупан број станара pre_k пре зграде k и укупан број станара $posle_k$ од зграде k до краја. На почетку, када је $k=0$, први број d_0 морамо експлицитно израчунати као $\sum_{i=1}^{n-1} i \cdot a_i$ (за то нам је потребно време $O(n)$), други број треба иницијализовати на нулу $pre_0 = 0$, а трећи на укупан број свих станара $posle_k = \sum_{i=0}^{n-1} a_i$ (и за то нам је потребно време $O(n)$). Затим за свако k од 1 до $n-1$ рачунамо $pre_k = pre_{k-1} + a_{k-1}$, $posle_k = posle_{k-1} + a_{k-1}$ и $d_k = d_{k-1} + pre_k + posle_k$.

Интересантно, укупну почетну дужину каблова можемо израчунати у линеарној сложености и без множења, тако што на збир додамо број станара у последњој згради, затим број станара у последње две зграде, затим број станара у последње три зграде и тако даље, све док не додамо број станара у свим зградама осим прве.

Пошто је и за једну и за другу фазу потребно време $O(n)$, то је уједно сложеност овог алгоритма.

```
#include <iostream>
#include <vector>
```

```cpp
using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> stanara(n);
  for (int i = 0; i < n; i++)
    cin >> stanara[i];

  // krećemo od zgrade 0
  // ukupna dužina kablova ako je ruter u tekućoj zgradi
  long long duzina_kablova = 0;
  for (int i = 0; i < n; i++)
    duzina_kablova += stanara[i] * i;
  // broj stanara pre tekuće zgrade
  long long stanara_pre = 0;
  // broj stanara od tekuće zgrade do kraja
  long long stanara_posle = 0;
  for (int i = 0; i < n; i++)
    stanara_posle += stanara[i];

  // minimalna duzina kablova
  long long min_duzina_kablova = duzina_kablova;

  // obrađujemo sve zgrade od 1 do n-1
  for (int k = 1; k < n; k++) {
    // ažuriramo brojeve stanara
    stanara_pre += stanara[k-1];
    stanara_posle -= stanara[k-1];
    // ažuriramo duz
    duzina_kablova += stanara_pre - stanara_posle;
    if (duzina_kablova < min_duzina_kablova)
      min_duzina_kablova = duzina_kablova;
  }

  cout << min_duzina_kablova << endl;

  return 0;
}
```

Задатак: Одбојка

Поставка: Данас се игра важан квалификациони меч између две локалне одбојкашке екипе, A и B. Бранко, који је велики љубитељ одбојке, због обавеза није могао да прати меч, па је видео само како се играчи поздрављају по завршетку утакмице. Тада се на екрану појавио последњи од оних досадних статистичких података који каже да је екипа A освојила укупно a поена, а екипа B укупно b поена током целе утакмице.

На основу ове информације Бранко сада покушава да закључи која екипа је победила. Наравно, Бранко зна да се меч играо на три добијена сета, а СВАКИ сет на 25 освојених поена (ни пети сет није скраћен), с тим да сет не може да се заврши са $25 : 24$. У случају резултата $24 : 24$ сет се наставља док једна од екипа не стекне предност од 2 поена.

Напишите програм који Бранку даје одговор на питање које га мучи.

Улаз: Са стандардног улаза се у првом реду уноси цео број a $(0 \leq a \leq 100)$, а у другом реду цео број b $(0 \leq b \leq 100)$, бројеви поена које су освојиле екипе током целе утакмице.

Излаз: На стандардни излаз исписати само једно слово.

- Ако је на основу укупног броја освојених поена извесно да је меч добила екипа A, исписати слово A.
- Ако је на основу укупног броја освојених поена извесно да је меч добила екипа B, исписати слово B.
- Ако није извесно која екипа је добила меч (то јест, ако је меч могла добити било која екипа), исписати слово N.

Пример 1

Улаз	*Излаз*
72 90	B

Пример 2

Улаз	*Излаз*
89 82	N

Решење: Размотримо најпре колико најмање поена треба да освоји екипа да би добила меч. Да би екипа добила меч, треба да добије три сета, а у сваком добијеном сету треба да освоји бар 25 поена. То значи да екипа која на крају меча има мање од 75 поена никако није могла да добије меч. Са друге стране, могуће је да екипа која освоји 75 поена добије меч, али то зависи и од броја поена противничке екипе.

Размотримо зато и колико екипа која изгуби три сета (а тиме и меч) може да надмаши победничку екипу у простом збиру поена. Екипа која је изгубила може да добије два сета са највише по 25 поена предности (резултатом 25:0), а да изгуби три сета са по најмање два поена заостатка. То значи да поражена екипа може да сакупи чак $2 \cdot 25 - 3 \cdot 2 = 50 - 6 = 44$ поена више од противника а да ипак изгуби меч. Екипа која има 45 или више поена предности у збиру, није могла да изгуби меч.

Из ове анализе произилази следећи закључак:

- ако је $a \geq 75$ и $b - a \leq 44$, екипа A је могла да добије меч.
- ако је $b \geq 75$ и $a - b \leq 44$, екипа B је могла да добије меч.

Пошто је меч завршен победом једне екипе, бар један од ових услова ће бити испуњен. Јасно, ако је испуњен само један од ових услова онда знамо победника, а ако су испуњена оба услова, онда је победник могла бити било која екипа.

```
#include <iostream>

using namespace std;

int main() {
  int a, b;
```

```
  cin >> a >> b;
  bool mogla_a = (a > 74) && (a + 44 >= b);
  bool mogla_b = (b > 74) && (b + 44 >= a);
  if (mogla_a && mogla_b)
    cout << 'N' << endl;
  else if (mogla_a)
    cout << 'A' << endl;
  else // if (mogla_b)
    cout << 'B' << endl;
}
```

Задатак: Хороскоп

Поставка: Јединствени матични број грађана (скраћено ЈМБГ) је низ од 13 цифара, који у нашој земљи свако добија још као сасвим мали. Прве две цифре овог низа представљају дан, а следеће две месец рођења. На пример, ако нечији ЈМБГ почиње са 0904, он је рођен деветог априла, а ако почиње са 3112 он је рођен тридесет првог децембра.

Перица воли да се забавља читајући разне хороскопе, а у једном тренутку се запитао колико има људи рођених у сваком од хороскопских знакова, за које би требало да важи оно што пише у хороскопу. Перица је некако успео да дође до повеће листе матичних бројева, а зна и почетне и завршне датуме за сваки знак:

- ОВАН – од 21. марта до 20. априла
- БИК – од 21. априла до 21. маја
- БЛИЗАНЦИ – од 22. маја до 21. јуна
- РАК – од 22. јуна до 22. јула
- ЛАВ – од 23. јула до 22. августа
- ДЕВИЦА – од 23. августа до 22. септембра
- ВАГА – од 23. септембра до 23.октобра
- ШКОРПИЈА – од 24. октобра до 22. новембра
- СТРЕЛАЦ – од 23. новембра до 21. децембра
- ЈАРАЦ – од 22. децембра до 20. јануара
- ВОДОЛИЈА – од 21. јануара до 19. фебруара
- РИБЕ – од 20. фебруара до 20. марта

Перица сада жели да преброји рођења у сваком од хороскопских знакова. Напишите програм који помаже Перици у бројању.

Улаз: На стандардном улазу се у првом реду налази број N $(1 \leq N \leq 200)$, број матичних бројева које је Перица набавио. У сваком од N наредних редова је по један матични број (13 цифара без размака).

Излаз: На стандардни излаз исписати 12 целих бројева, сваки у посебном реду. Први број је број особа рођени у знаку овна, други - у знаку бика, итд. Последњи, дванаести број је број особа рођених у знаку рибе.

Пример 1

Улаз	*Излаз*
5	2
1504279718463	0
0412622712918	0
1903326718649	0
0710262713307	0
0104646719372	0
	1
	0
	1
	0
	0
	1

Решење: Потребно је за сваки прочитани матични број одредити датум рођења особе, а затим за тај датум рођења одредити одговарајући хороскопски знак. Ово можемо да урадимо на различите начине. Један начин је да прво направимо списак уређених тројки, које се састоје од индекса хороскопског знака, и дана и месеца када се тај знак завршава. Тада је довољно да у петљи за текући ЈМБГ издвојимо месец и дан рођења особе и поредимо га редом са завршним датумима хороскопских знакова, док датум рођења не буде мањи од краја неког знака (пошто је број знакова веома мали, можемо слободно употребљавати линеарну претрагу). Када се то догоди, знамо којем знаку припада текући датум рођења. Датуме можемо представити било уређеним паровима (месец, дан), било их кодирати целим бројевима по принципу 100 пута месец + дан.

Да бисмо довршили задатак, можемо да уведемо листу од 12 бројача, тако да сви почињу од 0, а када за текућу особу установимо хороскопски знак, увећавамо за 1 бројач који одговара том знаку.

```
#include <iostream>
#include <string>
#include <vector>
#include <tuple>
using namespace std;

int main() {
  enum Znak {OVAN = 0, BIK, BLIZANCI, RAK, LAV, DEVICA,
             VAGA, SKORPIJA, STRELAC, JARAC, VODOLIJA, RIBE};

  // datume kodiramo kao 100 * mesec + dan
  vector<tuple<Znak, int>> poslednjiDanZnaka {
    make_tuple(JARAC, 120), // JARAC do 20. januara
    make_tuple(VODOLIJA, 219), // VODOLIJA do 21. februara
    make_tuple(RIBE, 320), // RIBE do 20 marta
    make_tuple(OVAN, 420), // OVAN do 20. aprila
    make_tuple(BIK, 521), // BIK do 21. maja
    make_tuple(BLIZANCI, 621), // BLIZANCI do 21. juna
    make_tuple(RAK, 722), // RAK do 22. jula
```

```
    make_tuple(LAV, 822), // LAV do 22. avgusta
    make_tuple(DEVICA, 922), // DEVICA do 22. septembra
    make_tuple(VAGA, 1023), // VAGA do 23. oktobra
    make_tuple(SKORPIJA, 1122), // SKORPIJA do 22. novembra
    make_tuple(STRELAC, 1221),  // STRELAC do 21. decembra
    make_tuple(JARAC, 1231)  // JARAC do 31. decembra
  };
  vector<int> broj(12, 0);

  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    // ucitavamo jmbg i izdvajamo datum rodjenja osobe
    string jmbg;
    cin >> jmbg;
    int dan = stoi(jmbg.substr(0, 2));
    int mesec = stoi(jmbg.substr(2, 2));
    int datum = 100 * mesec + dan;
    // linearnom pretragom pronalazimo znak osobe
    for (int znak = 0; znak <= 12; znak++)
      if (datum <= get<1>(poslednjiDanZnaka[znak])) {
        broj[get<0>(poslednjiDanZnaka[znak])]++;
        break;
      }
  }

  // ispisujemo broj osoba svakog znaka
  for (int i = 0; i < 12; i++)
    cout << broj[i] << endl;

  return 0;
}
```

Задатак: Аритмогриф

Поставка: При решавању једне познате врсте аритметичких ребуса потребно је иста слова заменити истим цифрама, а различита слова различитим цифрама, тако да се добије тачна једнакост. На пример, за ребус $SNEG + KRUG = SPORT$ једно од решења је $1937 + 8627 == 10564$. Можемо се уверити да је наведена једнакост заиста тачна, али то није тема овог задатка. Овде нас интересује други услов, а то је да ли је замена слова цифрама правилно изведена. На пример, понуђено "решење" $4832 + 5962 == 10794$ није исправно (иако је, узгред речено, једнакост тачна) зато што је слово S на једном месту замењено цифтом 4, а на другом цифром 1. Такође, цифра 4 одговара двама различитим словима (S и T).

Написати програм који проверава да ли је у овом типу ребуса замена слова цифрама правилно изведена.

Улаз: Са стандардног улаза се у првом реду уноси стринг, који представља описани аритметички ребус. Стринг може да садржи велика слова енглеске абецеде, размаке, знаке математичких операција и и знак једнакости. Укупна дужина стринга није већа од 100 а број различитих слова која се појављују у стрингу није већи од 10.

У другом реду се уноси стринг који представља понуђено решење ребуса из првог реда. Други стринг се од првог разликује искључиво по томе што је свако слово замењено неком цифром.

Излаз: На стандардни излаз исписати "da" ако је замена правилна (без обзира на то да ли је једнакост тачна), а "ne" ако замена није правилна.

Пример 1

Улаз	*Излаз*
`SNEG + KRUG = SPORT`	`da`
`2341 + 8751 = 20679`	

Пример 2

Улаз	*Излаз*
`SNEG + KRUG = SPORT`	`ne`
`4832 + 5962 = 10794`	

Решење: Задатак ћемо свакако решавати пролазећи истовремено кроз оба дата стринга. Пошто пролазећи кроз дате стрингове словима ребуса придружујемо цифре понуђеног решења, природно је да формирамо речник у коме памтимо та придруживања.

Када у петљи дохватимо текуће слово и текућу цифру, интересује нас да ли је ово прво појављивање тог слова.

Ако текуће слово није у речнику, ово је његово прво појављивање. Сада је важно да ли се и текућа цифра први пут појављује. Да бисмо то установили, користићемо скуп искоришћених цифара. Ако цифра није у скупу, то је њено прво појављивање и можемо у речнику да текућем слову придружимо текућу цифру, а у скуп искоришћених цифара да додамо текућу цифру. Ако је цифра већ била у скупу, то значи да понуђено решење не ваља.

Ако је текуће слово већ у речнику, онда само проверавамо да је њему придружена вредност из речника једнака текућој цифри. Ако није једнака, онда то поново значи да понуђено решење не ваља.

На крају, ако нисмо констатовали да понуђено решење не ваља, онда је замена слова цифрама исправна.

```
#include <iostream>
#include <map>
#include <set>

using namespace std;

int main() {
  string rebus, resenje;
  getline(cin, rebus);
  getline(cin, resenje);

  map<char, int>  zamene;
  set<char> iskoriscene_cifre;
  bool dobraZamena = true;
```

```
  for (size_t i = 0; i < rebus.size(); i++) {
    auto cifra_it = zamene.find(rebus[i]);
    if (cifra_it == zamene.end()) {
      if (iskoriscene_cifre.find(resenje[i]) != iskoriscene_cifre.end()) {
        dobraZamena = false;
        break;
      }
      zamene[rebus[i]] = resenje[i];
      iskoriscene_cifre.insert(resenje[i]);
    } else if (cifra_it->second != resenje[i]) {
      dobraZamena = false;
      break;
    }
  }
  if (dobraZamena)
    cout << "da" << endl;
  else
    cout << "ne" << endl;
}
```

Друга могућност је да у речнику за свако слово памтимо све цифре које су том слову придружене. Дакле кључеви тог речника су слова, а вредности су скупови цифара. Након формирања речника проверавамо да су сви скупови цифара једночлани (ако нису, понуђено решење не ваља). Овом провером смо потврдили да је сваком слову придружена тачно једна цифра, али не снамо да ли су различитим словима придружене различите цифре. Да бисмо то проверили, можемо да претходну проверу ставимо у функцију, а затим да функцију позовемо још једном, али са датим стринговима (ребус и понуђено решење) у замењеним улогама. Тако за сваку цифру добијамо скуп слова којима је та цифра придружена. Ако су и сви ови скупови једночлани, тиме потврђујемо и да је свака цифра придружена тачно једном слову, што значи да је замена слова цифрама исправна.

Задатак: Две полице

Поставка: У продавници се продају две врсте полица чије су цене и дужине познате. Желимо да поставимо полице дуж једног зида тако да попунимо што већи део дужине зида. При том, ако се иста дужина зида може попунити на више начина, бирамо јефтинију варијанту. Напиши програм који одређује највећу могућу дужину дела зида која се може попунити полицама и најмању цену по којој се то може урадити.

Улаз: У првом реду стандардног улаза се налази број z ($1 \leq z \leq 10^6$) који представља дужину зида. У два наредна реда се налазе описи полица: дужина (природан број између 1 и 10^6) и цена (природан број између 1 и 1000).

Излаз: На стандардни излаз исписати највећу могућу дужину дела зида који се може попунити полицама и најмању могућу цену, раздвојене једним размаком.

Пример

Улаз	*Излаз*
24 3 10 5 8	24 54

Објашњење

Цео зид се може попунити са 3 полице дужине 3 и 3 полице дужине 5 или са 8 полица дужине 3. Цена у првом случају је 54, а у другом случају је 80.

Решење: Задатак решавамо грубом силом тј. исцрпном претрагом свих могућности. Крећемо од варијанте у којој немамо полица типа 1 и затим додајемо једну по једну полицу типа 1, све док је то могуће (док је дужина дела зида попуњеног полицама типа 1 мања или једнака од укупне дужине зида). За сваки број полица типа 1 који тренутно разматрамо, израчунавамо највећи могући број полица типа 2 који се може поставити (њега одређујемо целобројним дељењем дужине дела зида иза постављених првих полица дужином друге полице). Након тога израчунавамо преосталу дужину зида иза свих полица и цену и ако је потребно ажурирамо минимум.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int zid;
  cin >> zid;
  int polica1, cena1;
  cin >> polica1 >> cena1;
  int polica2, cena2;
  cin >> polica2 >> cena2;

  if (polica1 < polica2) {
     swap(polica1, polica2);
     swap(cena1, cena2);
  }

  int min_ostalo = zid;
  int min_cena = 0;

  // isprobavamo sve mogucnosti
  // dodajemo jednu po jednu policu tipa 1 dok god je to moguce
  for (int broj1 = 0; broj1*polica1 <= zid; broj1++) {
    // racunamo najveci moguci broj preostalih polica tipa 2
    int broj2 = (zid - broj1*polica1) / polica2;
    int ostalo = zid - broj1*polica1 - broj2*polica2;
    // cena svih polica
    int cena = broj1*cena1 + broj2*cena2;
    // azuriramo minimum ako je potrebno
```

```
    if (ostalo < min_ostalo ||
       (ostalo == min_ostalo && cena < min_cena)) {
      min_ostalo = ostalo;
      min_cena = cena;
    }
  }

  cout << zid - min_ostalo << " " << min_cena << endl;
  return 0;
}
```

Задатак: Пуно фигурица

Поставка: Друштвену игру игра k играча и сваки играч игру почиње са по k фигура, при чему све фигуре једног играча морају бити исте јачине, док су јачине фигура различитих играча различите. Фигуре су доступне у неограниченим количинама, при чему је познат низ јачина доступних фигура. Напиши програм који одређује највећи број играча k који могу играти игру тако да разлика између укупне јачине свих фигура било која два играча не пређе задату границу.

Улаз: Са стандардног улаза се учитава број n ($1 \leq n \leq 10^5$), а затим у наредном реду n различитих расположивих јачина фигура (природни бројеви између 1 и 10^5). Наредни ред садржи границу (природни број између 1 и 10^{12}).

Излаз: На стандардни излаз исписати два броја раздвојена размаком – број k и најмању разлику укупних јачина фигура када игра k играча.

Пример

Улаз	*Излаз*
5 5 4 2 7 3 15	4 12

Објашњење

Ако играчи узму по четири фигуре јачине 5, 4, 2 и 3 највећа разлика јачина фигура играча биће $4 \cdot 5 - 4 \cdot 2 = 12$. Ако би играло 5 играча, морали би да узму и фигуре јачине 7 и највећа разлика би била $5 \cdot 7 - 5 \cdot 2 = 25$, што је веће од дозвољене границе.

Решење:

Наивно решење

Задатак се може решити тако што се за свако k између 2 и n провери да ли је могуће игру одиграти са k играча.

Централно питање је како за дато k изабрати k играча, тако да разлика између укупне јачине фигура најјачег и најслабијег међу њима буде што мања. Ефикасан алгоритам се може направити ако се низ јачина фигура претходно сортира. Играчи тада треба да узимају фигуре чије су јачине узастопних k елемената низа (ако би неки играч заменио фигуре, добила би се већа разлика између најјачег и најслабијег играча). Зато је

довољно проверити све узастопне k-точлане поднизове низа (којих има $n-k$), одузети последњи од првог члана и помножити резултат са k (јер сваки играч узима по k фигура).

Пошто се са порастом броја играча разлика може само повећати, претрагу можемо прекинути када први пут нађемо вредност k такву да игру не могу играти k играча. Сложеност таквог алгоритма, суштински заснованог на линеарног претрази је $O(n^2)$.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <limits>

using namespace std;

long long najmanja_razlika_k(const vector<int>& a, int k) {
  int min = numeric_limits<int>::max();
  for (size_t i = 0; i + k - 1 < a.size(); i++)
    if (a[i+k-1] - a[i] < min)
      min = a[i+k-1] - a[i];
  return min;
}

int main() {
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
  long long granica;
  cin >> granica;
  sort(begin(a), end(a));
  int k = 2;
  while (k <= n && k * najmanja_razlika_k(a, k) <= granica)
    k++;
  cout << k-1 << " " << (k-1) * najmanja_razlika_k(a, (k-1)) << endl;
  return 0;
}
```

Бинарна претрага

Брже решење можемо добити ако оптималну вредност k тражимо бинарном претрагом (то је могуће, јер вредности разлика расту са порастом k). Сложеност таквог алгоритма је $O(n \log n)$.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <limits>
#include <cassert>
```

```
using namespace std;

long long najmanja_razlika_k(const vector<int>& a, int k) {
  int min = numeric_limits<int>::max();
  for (size_t i = 0; i + k - 1 < a.size(); i++)
    if (a[i+k-1] - a[i] < min)
      min = a[i+k-1] - a[i];
  return min;
}

int main() {
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
  long long granica;
  cin >> granica;
  sort(begin(a), end(a));
  int lK = 2, dK = n;
  while (lK <= dK) {
    int k = lK + (dK - lK) / 2;
    if (k * najmanja_razlika_k(a, k) <= granica)
      lK = k+1;
    else
      dK = k-1;
  }
  cout << dK << " " << dK * najmanja_razlika_k(a, dK) << endl;
  return 0;
}
```

Два показивача

Још једно брзо решење можемо да добијемо техником два показивача. За сваки леви крај сегмента одређујемо највећу могућу вредност десног краја која не прелази границу. Након сортирања користимо сегмент $[poc, kraj]$ који представља изабране фигуре. Овај сегмент је на почетку $[0, 0]$. Ако је разлика између укупне јачине најјачих и најслабијих фигура из сегмента довољно мала, продужавамо сегмент надесно (повећавамо $kraj$), а ако је већа од дозвољене, онда скраћујемо сегмент слева (повећавамо poc). Сигурни смо да након повећања левог краја, десни крај не морамо смањивати (размисли зашто). Успут памтимо највећу дужину сегмента и оптималну разлику при тој дужини сегмента. Када прођемо цео низ јачина фигура, исписујемо коначне вредности највеће дужине сегмента и оптималне разлике. Сложеност оваквог решења је $O(n \log n)$ у почетној фази сортирања, а након тога, у другој фази је линеарна $O(n)$.

```
#include <iostream>
#include <vector>
#include <algorithm>
```

```cpp
#include <limits>
#include <cassert>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
  long long granica;
  cin >> granica;

  sort(begin(a), end(a));
  int poc = 0, kraj = 0;
  int opt_k = 1;
  long long opt_razlika = 0;
  while (kraj < n) {
    long long k = kraj - poc + 1;
    long long razlika = k * (a[kraj] - a[poc]);
    if (razlika > granica)
      poc += 1;
    else {
      if (k > opt_k) {
        opt_k = k;
        opt_razlika = razlika;
      } else if (k == opt_k)
        opt_razlika = min(opt_razlika, razlika);
      kraj += 1;
    }
  }
  cout << opt_k << " " << opt_razlika << endl;
  return 0;
}
```

1.3 Трећи круг квалификација

Задатак: Клацкалица

Поставка: Ђаци прваци су после школе свратили у парк да се клацкају. Током игре покушавали су и да претегну једно друго. Испоставило се да је Милица са школском торбом тешка отприлике као Радоје без торбе, а Радоје са Милицином торбом као Бојан без торбе. Можете ли да одредите приближно Милицину тежину ако су познате тежине Радоја и Бојана?

Улаз: Са стандардног улаза се у првом реду уноси цео број R $(25 \leq R \leq 50)$, а у другом реду цео број B $(25 \leq B \leq 50)$, Радојева и Бојанова тежина редом.

Излаз: На стандардни излаз исписати један цео број, Милицину приближну тежину.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
35 42	28	39 41	37

Решење: Тежину торбе можемо да одредимо као $T = B - R$. Након тога лако добијамо Милицину тежину као $B - T$.

```
#include <iostream>
using namespace std;

int main() {
  int radoje, bojan;
  cin >> radoje >> bojan;
  int torba = bojan - radoje;
  int milica = radoje - torba;
  cout << milica << endl;
  return 0;
}
```

Задатак: Приземље

Поставка: У једној згради лифт се веома ретко користи. Кад год уђете у приземље те зграде, лифт је слободан и стоји на неком спрату, а често управо у приземљу.

Пензионер Божур се из хобија бави електроником, па га је интересовало да ли би се исплатило да се лифт подеси тако да чим је слободан, да се врати у приземље. Зато је током прошлог месеца аутоматски прикупио податке, а сада хоће да упосли унука Уроша, програмера, да одреди колико се у приземљу просечно чека на лифт. Урошу би помоћ добро дошла.

Улаз: Са стандардног улаза се у првом реду уноси цео број N $(1 \leq N \leq 1000)$, број позива лифта из приземља. У наредних N редова је по један цео број S $(0 \leq S \leq 15)$, број спратова које је празан лифт прешао по позиву из приземља (да би стигао у приземље).

Излаз: На стандардни излаз исписати један цео број, просечан број спратова које лифт пређе да би празан дошао у приземље на позив, заокружен навише.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
5 4 0 0 2 0	2	2 0 0	0

Решење: Да се добије средња вредност, потребно је сабрати свих N вредности S, добијени збир поделити са N. На крају треба још заокружити добијену вредност навише, што се може постићи стандардном функцијом из библиотеке или коришћењем везе да је $\left\lceil \frac{a}{b} \right\rceil = \frac{a+b-1}{b}$.

```
#include <iostream>
using namespace std;

int main() {
  int n;
  cin >> n;
  int zbir = 0;
  for (int i = 0; i < n; i++) {
    int s; cin >> s;
    zbir += s;
  }
  cout << (zbir + n - 1) / n << endl;
  return 0;
}
```

Задатак: Брана

Поставка: Даброви граде брану да би подигли ниво воде изнад улаза у њихова склоништа. Дабра Животу боли зуб, па он данас уместо да ради, процењује колико још има посла. Живота је прво измерио висину сваког улаза у склониште, а затим за неколико околних стабала оценио колико би она подигла ниво воде ако би била уграђена у брану.

Живота жели да за сваки могући ниво воде одреди колико би улаза у склоништа остало на сувом (те улазе би требало затрпати). Улаз је на сувом ако је његова висина строго већа од нивоа воде.

Улаз: Са стандардног улаза учитава се број улаза n ($0 \leq n \leq 50000$), а затим и висине улаза (цели бројеви), задати у сортираном редоследу од највишег до најнижег. Након тога се учитава број m ($1 \leq m \leq 50000$) који представља број могућих нивоа воде, а затим и m бројева који представљају нивое за које треба одговорити колико би улаза у склоништа остало на сувом, када би вода достигла тај ниво. Сваки број се налази у посебном реду.

Излаз: На стандардни излаз за сваки ниво воде исписати тражени број непотопљених улаза, сваки у посебном реду.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
5	0	4	3
79	4	30	0
63	3	29	1
63	5	29	0
46		28	4
13		5	
4		28	
85		30	
40		29	
60		31	
0		27	

Решење: У задатку је потребно ефикасно одредити број елемената сортираног низа који су већи од датог броја. Ако нађемо позицију првог елемента који је већи од датог броја, тада број таквих елемената можемо одредити тако што израчунамо разлику између укупног броја чланова низа и те позиције.

Најједноставнији начин да нађемо позицију првог елемента који је већи од датог броја је да применимо линеарну претрагу и да редом проверавамо један по један елемент све док не дођемо или до краја низа или до тражене позиције. Сложеност овакве претраге је $O(n)$, па би укупна сложеност решења била $O(m \cdot n)$, што је превише имајући у виду ограничења дата у задатку.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> visine_ulaza(n);
  for (int i = 0; i < n; i++)
    cin >> visine_ulaza[i];

  int m;
  cin >> m;
  for (int i = 0; i < m; i++) {
    int nivo_vode;
    cin >> nivo_vode;
    int broj = 0;
    for (int visina : visine_ulaza)
      if (visina > nivo_vode)
    broj++;
    cout << broj << endl;
  }
```

```
  return 0;
}
```

Позиција се ефикасно може пронаћи применом алгоритма бинарне претраге. Најједноставније је употребити библиотечку имплементацију. У језику C++ можемо употребити функцију `upper_bound` која враћа итератор који указује на тражену позицију или на крај низа (позицију непосредно иза последњег елемента) ако тражени елемент не постоји). Број елемената можемо одредити израчунавајући растојање од те позиције до краја низа (најбоље функцијом `distance`). Сложеност једне бинарне претраге је $O(\log n)$, па је укупна сложеност алгоритма $O(m \log n)$.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iterator>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false); cin.tie(0);
  int n;
  cin >> n;
  vector<int> visine_ulaza(n);
  for (int i = n-1; i >= 0; i--)
    cin >> visine_ulaza[i];

  int m;
  cin >> m;
  for (int i = 0; i < m; i++) {
    int nivo;
    cin >> nivo;
    cout << distance(upper_bound(begin(visine_ulaza), end(visine_ulaza), nivo),
                     end(visine_ulaza)) << '\n';
  }

  return 0;
}
```

Наравно, позицију првог елемента који је већи од датог можемо пронаћи и ручно имплементираном бинарном претрагом.

```
#include <iostream>
#include <vector>

using namespace std;

int prvi_veci(const vector<int>& a, int x) {
  int l = 0, d = a.size()-1;
  while (l <= d) {
```

```
    int s = l + (d - l) / 2;
    if (a[s] <= x)
      l = s + 1;
    else
      d = s - 1;
  }
  return d + 1;
}

int main() {
  ios_base::sync_with_stdio(false); cin.tie(0);
  int n;
  cin >> n;
  vector<int> visine_ulaza(n);
  for (int i = n-1; i >= 0; i--)
    cin >> visine_ulaza[i];

  int m;
  cin >> m;
  for (int i = 0; i < m; i++) {
    int nivo_vode;
    cin >> nivo_vode;
    cout << n - prvi_veci(visine_ulaza, nivo_vode) << '\n';
  }

  return 0;
}
```

Задатак: Највећи број од датих цифара

Поставка: За дати број исписати највећи број који се пише истим цифрама.

Улаз: Са стандардног улаза се у првом реду уноси број N ($1 \leq N \leq 10^{30}$).

Излаз: На стандардни излаз исписати тражени број.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
90123	93210	777	777

Решење: Задатак једноставно решавамо тако што сортирамо цифре опадајуће. С обзиром на велики број цифара, најбоље је резултат представити ниском карактера.

```
#include <iostream>
#include <string>
#include <algorithm>
#include <functional>

using namespace std;
```

```
int main() {
  string cifre;
  cin >> cifre;
  sort(begin(cifre), end(cifre), greater<char>());
  cout << cifre << endl;

  return 0;
}
```

Резултат је могуће представити и низом цифара.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <functional>

using namespace std;

int main() {
  vector<int> cifre;
  char c;
  while (cin >> c && c != '\n')
    cifre.push_back(c - '0');
  sort(begin(cifre), end(cifre), greater<int>());
  for (size_t i = 0; i < cifre.size(); i++)
    cout << cifre[i];
  cout << endl;

  return 0;
}
```

Задатак: Ризико

Поставка: Током игре ризико нападачки и одбрамени играч улазе у битке бацајући коцкице. Сваки од њих има 3 коцкице а на основу тренутног стања игре одређује се колико од њих сме да баци. Када се баце коцкице, упоређују се најјача коцкица нападача (она са највећим бројем) са најачом коцкицом одбрамбеног играча, затим друга по јачини са другом по јачини и најмања са најмањом. Ако је неки играч бацио мање коцкица, тада се најслабије коцкице оног другог играча занемарују (онај ко је бацао више коцкица је у предности). Када се две коцкице упоређују, ако нападач има строго јачу коцкицу (коцикцу на којој је број строго већи) одбрана губи један поен, док у супротном напад губи један поен (тиме је одбрамбени играч у благој предности). Напиши програм који на основу резултата бацања коцкица одређује број поена које губе један и други играч.

Улаз: Први ред стандардног улаза садржи број бачених коцкица нападача (1, 2 или 3), а наредни ред садржи бројеве (од 1 до 6) на коцкицама које је нападач бацио. Наредни ред стандардног улаза садржи број бачених коцкица одбрамбеног играча (1, 2 или 3), а

наредни ред садржи бројеве (од 1 до 6) на коцкицама које је одбрамбени играч бацио.

Излаз: На стандардни излаз исписати број поена које губи нападач и број поена које губи одбрамбени играч, раздвојене једним размаком.

Пример 1

Улаз *Излаз*

```
3
5 6 3
3
5 3 4
```

```
1 2
```

Објашњење

Нападач 6 се пореди са одбраном 5, нападач 5 са одбраном 4 и нападач 3 са одбраном 3. У прве две борбе побеђује нападач, па одбрана губи два поена, док у трећој побеђује одбрана (јер нападач није добио строго већи број), па нападач губи 1 поен.

Пример 2

Улаз

```
3
1 4 6
2
5 5
```

Излаз

```
1 1
```

Објашњење

Нападач 6 се пореди са одбраном 5, нападач 4 са одбраном 5, док се нападач 1 занемарује. У првој борби побеђује нападач, па одбрана губи поен, док у другој побеђује одбрана, па нападач губи поен.

Решење: Задатак решавамо тако што сортирамо низ коцкица нападача и низ коцкица одбрамбеног играча опадајуће и онда редом поредимо једну по једну коцкицу од почетка низа све док се неки од низова не исцрпи и на одговарајући начин ажурирамо бројаче изгубљених фигура за играча који напада и играча који се брани.

```
#include <iostream>
#include <algorithm>
#include <functional>

using namespace std;

int main() {
  int nn;
  cin >> nn;
  int napad[3];
  for (int i = 0; i < nn; i++)
```

```
    cin >> napad[i];
  int no;
  cin >> no;
  int odbrana[3];
  for (int i = 0; i < no; i++)
    cin >> odbrana[i];

  sort(napad, napad + nn, greater<int>());
  sort(odbrana, odbrana + no, greater<int>());

  int gubi_napad = 0, gubi_odbrana = 0;
  for (int i = 0; i < min(nn, no); i++)
    if (napad[i] > odbrana[i])
      gubi_odbrana += 1;
    else
      gubi_napad += 1;

  cout << gubi_napad << " " << gubi_odbrana << endl;

  return 0;
}
```

Задатак: Атп победник

Поставка: Током године играју се многи тениски турнири на којима тенисери освајају поене. Поени се сабирају и на крају године објављује се завршна листа на којој су тенисери рангирани на основу укупног броја поена током те године. Напиши програм који на основу резултата свих турнира одређује најбољег тенисера и његове поене (претпоставити да ће победник имати строго већи број поена од свих осталих).

Улаз: Са стандардног улаза се уноси број n, а затим у наредних n редова подаци о освојеним поенима тенисера: презиме тенисера и затим број освојених поена.

Излаз: На стандардни излаз исписати презиме и укупан број поена победника.

Пример 1

Улаз	*Излаз*
9 Djokovic 1000 Nadal 800 Federer 600 Nadal 1000 Federer 800 Djokovic 600 Djokovic 1000 Cicipas 800 Nadal 600	Djokovic 2600

Пример 2

Улаз	*Излаз*
6 Zverev 1000 Cicipas 800 Medvedev 700 Thiem 1000 Cicipas 800 Medvedev 600	Cicipas 1600

Решење: У првој фази програма је потребно да израчунамо укупан број поена за сваког играча. Податке можемо чувати у мапи која пресликава име играча у његов број

поена. Након тога одређујемо највећу вредност у мапи (што можемо урадити било применом библиотечке функције, било проласком кроз све елементе мапе и ажурирањем максимума).

```
#include <iostream>
#include <algorithm>
#include <map>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  int n;
  cin >> n;
  map<string, int> bodovi;
  for (int i = 0; i < n; i++) {
    string teniser; int bodovi_na_turniru;
    cin >> teniser >> bodovi_na_turniru >> ws;
    bodovi[teniser] += bodovi_na_turniru;
  }

  auto it = max_element(bodovi.begin(), bodovi.end(),
                        [](const pair<string, int>& p1,
                           const pair<string, int>& p2) {
                          return p1.second < p2.second;
                        });
  cout << it->first << " " << it->second << endl;
  return 0;
}
```

Задатак: Атп листа

Поставка: Током године играју се многи тениски турнири на којима тенисери освајају поене. Поени се сабирају и на крају године објављује се завршна листа на којој су тенисери рангирани на основу укупног броја поена током те године. Напиши програм који на основу резултата свих турнира одређује k најбољих тенисера и њихове поене (претпоставити да ће сви тенисери имати различит број поена).

Улаз: Са стандардног улаза се учитава број турнира n, а затим подаци о освојеним поенима на тим турнирима. За сваки турнир се учитава број m који представља број тенисера који су освајали поене, и након тога m линија које садрже податке о освојеним поенима тенисера на том турниру (презиме тенисера које има највише 50 карактера и број поена између 10 и 2000). На крају се уноси број k који одређује колико најбољих тенисера треба одредити.

Излаз: На стандардни излаз исписати презимена и укупан освојени број поена за k најбољих тенисера у опадајућем редоследу укупног освојеног броја поена.

Пример

Улаз	*Излаз*
3 3 Djokovic 1000 Nadal 800 Federer 600 3 Nadal 1000 Federer 800 Djokovic 600 3 Djokovic 1000 Cicipas 800 Nadal 600 3	Djokovic 2600 Nadal 2400 Federer 1400

Решење: У првој фази програма је потребно да израчунамо укупан број поена за сваког играча. Податке можемо чувати у мапи која пресликава име играча у његов број поена. Након тога треба одредити k највећих вредности у мапи. Један начин је да се сви елементи из мапе прекопирају у низ, да се низ сортира опадајуће и да се прочита првих k елемената тако сортираног низа.

```
#include <iostream>
#include <algorithm>
#include <map>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  map<string, int> bodovi;

  int broj_turnira;
  cin >> broj_turnira;
  for (int i = 0; i < broj_turnira; i++) {
    int broj_tenisera;
    cin >> broj_tenisera;
    for (int j = 0; j < broj_tenisera; j++) {
      string teniser; int bodovi_na_turniru;
      cin >> teniser >> bodovi_na_turniru >> ws;
      bodovi[teniser] += bodovi_na_turniru;
    }
  }

  int k;
  cin >> k;
  vector<pair<string, int>> teniseri(bodovi.size());
```

```
  int i = 0;
  for (auto it : bodovi)
    teniseri[i++] = it;
  sort(begin(teniseri), end(teniseri),
       [](const pair<string, int>& p1, const pair<string, int>& p2) {
         return p1.second > p2.second;
       });
  for (int i = 0; i < k; i++)
    cout << teniseri[i].first << " " << teniseri[i].second << endl;

  return 0;
}
```

У језику C++ могуће је одредити k највећих елемената без сортирања целе колекције (што је донекле ефикасније). За колекције у којима је могуће произвољно мењати редослед елемената (као што је низ или вектор) можемо употребити функцију `partial_sort`. Пошто је редослед елемената у мапи фиксиран (мапа је уређена на основу кључева), за одређивање највећих елемената мапе можемо употребити библиотечку функцију `partial_sort_copy` која ће тих k највећих елемената прекопирати у нови низ (или вектор).

```
#include <iostream>
#include <algorithm>
#include <map>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  map<string, int> bodovi;

  int broj_turnira;
  cin >> broj_turnira;
  for (int i = 0; i < broj_turnira; i++) {
    int broj_tenisera;
    cin >> broj_tenisera;
    for (int j = 0; j < broj_tenisera; j++) {
      string teniser; int bodovi_na_turniru;
      cin >> teniser >> bodovi_na_turniru >> ws;
      bodovi[teniser] += bodovi_na_turniru;
    }
  }

  int k;
  cin >> k;
  vector<pair<string, int>> najbolji(k);
  partial_sort_copy(bodovi.begin(), bodovi.end(),
                    najbolji.begin(), najbolji.end(),
```

```
                    [](const pair<string, int>& p1,
                       const pair<string, int>& p2) {
                      return p1.second > p2.second;
                    });
  for (auto p : najbolji)
    cout << p.first << " " << p.second << endl;
  return 0;
}
```

Задатак: Слаткиши за сав новац

Поставка: Дуж једне улице деца продају слаткише. Јована има G динара и жели да их потроши тако што ће кренути да се шета дуж улице и од сваког детета, редом, ће купити тачно један слаткиш (ни једно дете неће прескочити). Ако су познате цене свих слаткиша и ако је позната позиција са које Јована креће у куповину, одредити број слаткиша које ће Јована на тај начин купити.

Улаз: Са стандардног улаза се уноси број деце који продају слаткише n ($1 \leq n \leq 5 \cdot 10^4$), а затим и низ који садржи цене слаткиша (позитивни природни бројеви мањи од 100). Након тога се уноси број упита k ($1 \leq k \leq 5 \cdot 10^4$), а затим у k наредних редова упити који садрже позицију првог детета које ће Јована обићи (позиције се броје од нуле), а затим број динара које Јована има.

Излаз: На стандардни излаз за сваки упит у посебном реду исписати број слаткиша које ће Јована купити.

Пример

Улаз	*Излаз*
7	0
3 5 1 2 3 1 4	2
4	5
3 1	3
2 5	
2 13	
0 10	

Објашњење

У првом упиту Јована има један динар, а креће од детета које продаје слаткиш за 2 динара, па не може да купи ни један слаткиш.

У другом упиту Јована има пет динара и купује слаткише који коштају 1 и 2 динара.

У трећем упиту Јована има 13 динара и купује слаткише који коштају 1, 2, 3, 1 и 4 динара.

У четвртом упиту Јована има 10 динара и купује слаткие који коштају 3, 5 и 1 динар.

Решење: Решење грубом силом подразумева да се за сваку почетну позицију са које Јована креће редом сабирају цене колача, све док је збир мањи од новца које Јована има. Сложеност таквог решења је $O(n \cdot q)$, што је с обзиром на ограничења недопустиво споро.

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> cena(n);
  for (int i = 0; i < n; i++)
    cin >> cena[i];
  int Q;
  cin >> Q;
  for (int q = 0; q < Q; q++) {
    int p, novac;
    cin >> p >> novac;
    int kupljeno = 0;
    while (p < n && novac >= cena[p]) {
      novac -= cena[p];
      kupljeno++;
      p++;
    }
    cout << kupljeno << endl;
  }
}
```

Ефикасније решење можемо постићи ако уместо низа цена колача одржавамо низ збирова префикса низа цена (чувамо низ z такав да је $z_k = 0$ и $z_k = \sum_{i=0}^{k-1} c_k$). Пошто су цене позитивне, низ збирова префикса је сортиран растуће и може се претраживати бинарном претрагом. Ако Јована креће са позиције p а последњи колач купује на позицији p', тада се цена коју ће платити може израчунати као $z_{p'+1} - zp$. Дакле, бинарном претрагом ћемо наћи највећу вредност p' такву да је $z_{p'+1} - z_p \leq N$ (где је N износ новца који Јована има).

Низ префиксних збирова можемо израчунати у сложености $O(n)$ приликом учитавања низа цена, након чега вршимо q бинарних претрага, свака је сложености $O(\log n)$. Дакле, укупна сложеност је $O(n + q \log n)$.

Нагласимо да је ово један од задатака у ком се одговара на упите тј. у ком се наизменично читају подаци са стандардног улаза и исписују на стандардни излаз. Ако се не обрати посебна пажња, пуно времена може отићи на синхронизацију између улаза и излаза.

У језику C++ прво што треба урадити да би се убрзао улаз и излаз је искључивање сихнронизације са C-овском библиотеком за улаз и излаз (то се постиже наредбом `ios_base::sync_with_stdio(false)`). Када се то уради веома важно је да се нигде у програму не позивају функције из библиотеке `<cstdio>`. Након тога, важно је да спречимо пражњење излазног бафера при сваком испису. Прво је да искључимо аутоматско пражњење излазног бафера пре учитавања података са улаза, што радимо наредбом `cin.tie(0)`. Након тога програм неће лепо радити у интерактивном режиму.

На крају, потребно је да осигурамо да не користимо `endl`, који по својој дефиницији врши пражњење излазног бафера и да га заменимо исписом карактера `\n`. И ова измена доводи до тога да се у интерактивном тестирању програм не понаша исправно, али то нам не смета, јер је циљ да максимално убрзамо програм за аутоматско тестирање.

Друго решење (које додуше троши више меморије) је да резултате све сместимо у низ и да на крају програма испишемо тај низ.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false); cin.tie(0);
  int n;
  cin >> n;
  vector<int> zbir_prefiksa_cena(n+1);
  zbir_prefiksa_cena[0] = 0;
  for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    zbir_prefiksa_cena[i+1] = zbir_prefiksa_cena[i] + x;
  }
  int q;
  cin >> q;
  for (int i = 0; i < q; i++) {
    int p, novac;
    cin >> p >> novac;
    auto poc = next(begin(zbir_prefiksa_cena), p);
    auto kraj = end(zbir_prefiksa_cena);
    auto it = upper_bound(poc, kraj, zbir_prefiksa_cena[p] + novac);
    cout << distance(poc, it) - 1 << '\n';
  }
  return 0;
}
```

Задатак: Мали поштар

Поставка: Јовица зарађује џепарац тако што доноси пакете својим комшијама. Поделу креће од своје куће и потребно је да пакете разнесе у друге куће распоређене дуж улице и да се врати назад у своју кућу. За сваку кућу познато је растојање од почетка улице. Најкраћи пут би прешао ако би пакете сложио тако да их редом дели комшијама дуж улице. Пошто Јовица жели да буде у доброј физичкој форми, он током поделе пакета трчи и жели да пакете уреди тако да пређе што већи пут. Напиши програм који одређује највећи пут који може да пређе.

Улаз: Са стандардног улаза се уноси број кућа у које треба донети пакете (међу њима

се налази и Јовицина кућа), а затим и растојања тих кућа од почетка улице.

Излаз: На стандардни излаз исписати највеће растојање које Јовица може прећи током поделе пакета.

Пример 1

Улаз	*Излаз*
5 7 3 6 10 2	24

Објашњење

Постоји више начина да Јовица претрчи 24 дужне јединице. На пример, ако је његова кућа на позицији 3, он може да редом обилази куће 3, 7, 2, 10, 6, 3.

Пример 2

Улаз

```
7
3 5 11 4 2 17 9
```

Излаз

```
56
```

Решење: Решење грубом силом подразумева да се провере сви могући редоследи обиласка n кућа, тј. свих $n!$ пермутација датих бројева и да се утврди која од њих даје највећу могућу вредност пређеног пута. Овај алгоритам је изразито неефикасан и може се применити само на веома, веома мале улазе (практично, само за $n \leq 10$ програм може да реши задатак у датом временском ограничењу).

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int zbirApsRazlika(const vector<int>& a) {
  int zbir = 0;
  for (size_t k = 1; k < a.size(); k++)
    zbir += abs(a[k] - a[k-1]);
  zbir += abs(a[a.size()-1] - a[0]);
  return zbir;
}

int main() {
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
```

```
  int maks = 0;
  do {
    maks = max(zbirApsRazlika(a), maks);
  } while(next_permutation(begin(a), end(a)));

  cout << maks << endl;
  return 0;
}
```

Интуитивно нам је јасно да ће се пуно претрчати ако се стално трчи са једног на други крај улице. Један грамзиви приступ је да се прво обиђе крајња лева кућа, па затим крајња десна, па друга слева, па друга здесна и тако "цик-цак". Докажимо да је ово грамзиво решење исправно.

За дати распоред $x_0, \dots x_{n-1}$ одређујемо суму апсолутних вредности разлика елемената тј. покушавамо да максимизујемо суму

$$|x_0 - x_1| + |x_1 - x_2| + ... + |x_{n-2} - x_{n-1}| + |x_{n-1} - x_0|.$$

Сваки елемент се у суми јавља тачно два пута. У зависности од међусобног односа бројева неки елементи ће бити узети са знаком $+$, а неки са знаком $-$, и то тако да се тачно елемената узима са знаком $+$ и тачно елемената узима са знаком $-$. Циљ нам је да елементи који се узимају са знаком $+$ буду што већи, а да ови са знаком $-$ буду што мањи. Распоред који иде цик-цак постиже да се за знаком $+$ узме $\frac{n}{2}$ већих елемената низа, а са знаком $-$ узме $\frac{n}{2}$ мањих елемената низа (ако их је непаран број, тада се средњи узима једном са знаком $-$, а једном са знаком $+$). Заиста, ако имамо 6 елемената $a_0 \leq a_1 \leq a_2 \leq a_3 \leq a_4 \leq a_5$, цик-цак распоред даје вредност

$$|a_0 - a_5| + |a_5 - a_1| + |a_1 - a_4| + |a_4 - a_2| + |a_2 - a_3| + |a_3 - a_0|,$$

што је једнако

$$(a_5 - a_0) + (a_5 - a_1) + (a_4 - a_1) + (a_4 - a_2) + (a_3 - a_2) + (a_3 - a_0),$$

тј.

$$2 \cdot (a_5 + a_4 + a_3) - 2 \cdot (a_2 + a_1 + a_0)$$

Јасно је да се не може добити више од овога, а ово се, видели смо, увек може експлицитно достићи баш цик-цак распоредом.

Елементе низа можемо експлицитно сортирати елементе низа, па затим распоредити елементе у нови низ по цик-цак редоследу у нови низ и за тај нови низ израчунати збир апсолутних вредности разлика суседних елемената.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;
```

```
int main() {
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
  sort(begin(a), end(a));
  vector<int> b(n);
  int i = 0, j = n-1;
  for (int k = 0; k < n; k++)
    if (k % 2 != 0)
      b[k] = a[i++];
    else
      b[k] = a[j--];

  int zbir = 0;
  for (int k = 1; k < n; k++)
    zbir += abs(b[k] - b[k-1]);
  zbir += abs(b[n-1] - b[0]);

  cout << zbir << endl;
  return 0;
}
```

Помоћни низ се може веома једноставно избећи у имплементацији.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
  sort(begin(a), end(a));
  int i = 0, j = n-1;
  int zbir = 0;
  bool paran = true;
  while (i < j) {
    zbir += abs(a[j]-a[i]);
    if (paran)
      i++;
    else
```

```
      j--;
    paran = !paran;
  }
  zbir += abs(a[0] - a[i]);
  cout << zbir << endl;
  return 0;
}
```

Ако пажљиво размотримо доказ коректности, примећујемо да распоред у коме идемо од прве до последње, па до друге, затим претпоследње куће итд., није једини који даје максимални пређени пут. Довољно је само да наизменично узимамо елементе из прве и друге половине низа (у било ком редоследу). Ово инспирише још једноставнији алгоритам за израчунавање траженог максимума (саберемо $\frac{n}{2}$ бројева са почетка, одузмемо $\frac{n}{2}$ бројева са краја низа и помножимо са 2).

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
  sort(begin(a), end(a));

  int zbir = 0;
  for (int i = 0; i < n/2; i++) {
    zbir += a[n-1-i];
    zbir -= a[i];
  }

  cout << zbir * 2 << endl;
}
```

Задатак: Инвестициони фонд

Поставка: Инвестициони фонд који се бави продајом криптовалута је објавио очекивани профит за сваки могући уложени износ у биткоинима. Никола је студент-програмер који је био веома вредан током прошле године, зарадио је N биткоина и жели да распореди улоге тако да добије што је већи могући профит. Пошто тренутно мора да спрема испите и не може да програмира, ти му помози тако што ћеш написати програм који одређује максимални профит.

Улаз: Са стандардног улаза се уноси број биткоина N ($1 \leq N \leq 5000$), које Никола има на располагању а затим N позитивних природних бројева раздвојених са по јед-

ним размаком који представљају очекивану добит за сваки уложени износ од 1 до N у биткоинима.

Излаз: На стандардни излаз исписати највећи могући износ биткоина који Никола може добити након улагања.

Пример 1

Улаз
```
6
1 2 3 4 5 6
```
Излаз
```
6
```

Пример 2

Улаз
```
6
10 21 32 41 51 61
```
Излаз
```
64
```

Објашњење

Ако Јовица два пута уложи по 3 биткоина имаће профит 32+32 = 64.

Пример 3

Улаз
```
10
1 5 8 9 10 17 17 20 24 26
```

Излаз
```
27
```

Објашњење

Ако Јовица уложи 6 биткоина и два пута по 2 биткоина, имаће профит 17 + 5 + 5 = 27.

Решење: Задатак решавамо тако што за сваки могући новчани улог n између 1 и N израчунавамо зараду која се може добити ако се уложи баш тај улог. Добит од улога n је дата на улазу, и након тога на преостаје $N - n$ новца. Тиме смо за сваки улаз n полазни проблем свели на потпроблем истог облика, али мање димензије, који се може решити рекурзијом. Ако обележимо са $f(n)$ највећи профит који се може добити улагањем n динара, важе следеће рекурентне релације:

$$\begin{aligned} f(0) &= 0 \\ f(N) &= \max_{1 \leq n \leq N} (d(n) + f(N-n)) \end{aligned}$$

На основу овога је веома једноставно дефинисати рекурзивну функцију.

```
#include <iostream>
#include <vector>

using namespace std;

int zarada(int N, const vector<int>& dobit) {
  if (N == 0)
```

```
    return 0;
  int maks = 0;
  for (int n = 1; n <= N; n++)
    maks = max(maks, dobit[n] + zarada(N-n, dobit));
  return maks;
}

int main() {
  int N;
  cin >> N;
  vector<int> dobit(N+1);
  dobit[0] = 0;
  for (int n = 1; n <= N; n++)
    cin >> dobit[n];
  cout << zarada(N, dobit) << endl;
  return 0;
}
```

Пошто у претходном решењу долази до понављања рекурзивних позива, потребно је применити динамичко програмирање. Један начин је да се изврши мемоизација.

```
#include <iostream>
#include <vector>

using namespace std;

int zarada(int N, const vector<int>& dobit, vector<int>& memo) {
  if (memo[N] != -1)
    return memo[N];

  if (N == 0)
    return 0;
  int maks = 0;
  for (int n = 1; n <= N; n++)
    maks = max(maks, dobit[n] + zarada(N-n, dobit, memo));
  return memo[N] = maks;
}

int zarada(int N, const vector<int>& dobit) {
  vector<int> memo(N+1, -1);
  return zarada(N, dobit, memo);
}

int main() {
  int N;
  cin >> N;
  vector<int> dobit(N+1);
  dobit[0] = 0;
```

```
  for (int n = 1; n <= N; n++)
    cin >> dobit[n];
  cout << zarada(N, dobit) << endl;
  return 0;
}
```

Рекурзије се можемо ослободити и задатак можемо решити динамичким програмирањем навише.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int N;
  cin >> N;
  vector<int> dobit(N+1);
  dobit[0] = 0;
  for (int n = 1; n <= N; n++)
    cin >> dobit[n];
  vector<int> dp_zarada(N+1);
  dp_zarada[0] = 0;
  for (int n = 1; n <= N; n++) {
    dp_zarada[n] = 0;
    for (int ulog = 1; ulog <= n; ulog++) {
      int c = dobit[ulog] + dp_zarada[n-ulog];
      if (c >= dp_zarada[n])
        dp_zarada[n] = c;
    }
  }
  cout << dp_zarada[N] << endl;
  return 0;
}
```