# Бинарна претрага и сортирање - задаци

## Употреба библиотечких функција у Ц++

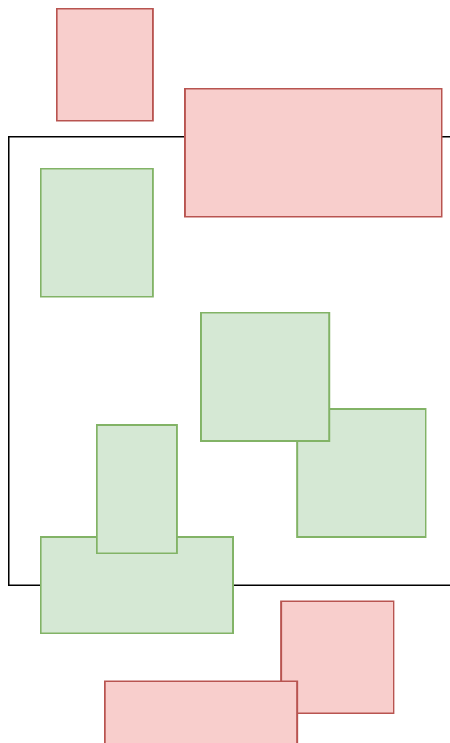
## (upper_bound, lower_bound, distance, partial_sum)

https://arena.petlja.org/sr-Latn-RS/competition/2021binarnapretragasortiranje

1.

Први већи и последњи мањи

Веб-страница садржи неколико објеката правоугаоног облика (слика, пасуса, табела и слично). За сваки објекат је познат положај његове горње и доње ивице у односу на врх странице. Приликом приказа странице и померања приказа (скроловања) многи објекти се не приказују. Систем ради тако што приказује све оне објекте чија се горња ивица налази унутар приказаног дела странице (како је приказано на слици). Напиши програм који одређује објекте чија се горња ивица види током приказа одређених делова странице.



Приказани и скривени објекти

Улаз

Са стандардног улаза учитава се број n (1≤n≤50000), а затим n линија које садрже парове природних бројева мањих од $10^6$ раздвојене са по једним размаком, а који представљају положај (удаљеност од врха странице) горње и доње ивице n објеката са странице. Објекти су поређани неопадајуће у односу на положај њихове горње ивице.

Након тога се уноси број m (1≤m≤50000), а затим m парова природних бројева раздвојених са по једним размаком (њих највише 50000) који представљају положај (удаљеност од врха странице) горње и доње границе видљивог дела странице.

Излаз

За сваки пар бројева који одређују видљиви део странице на стандардни излаз исписати по једну линију која садржи два цела броја раздвојена размаком. Први број представља позицију (индекс унутар низа, бројано од нуле) првог објекта чији је положај горње ивице строго већи од положаја горње границе видљивог дела странице, а други број представља позицију последњег објекта чији је положај горње ивице строго мањи од положаја доње границе видљивог дела странице. Ако се горње ивице свих објеката у низу налазе изнад горње границе видљивог дела, први број треба да буде једнак n. Ако се горње ивице свих објеката у низу налазе испод доње границе, други број треба да буде -1.

Пример

Улаз

```
11

3 5

4 7

5 7

8 13

8 16

9 11

11 17

11 12

11 20

13 20

14 18

6

5 12

6 13

8 14

0 20

20 25
```

```
1 2
```

Излаз

```
3 8

3 8

5 9

0 10

11 10

0 -1
```

Објашњење

У првом упиту видљив део странице је од 5 до 12, први видљив објекат је (8,13) на позицији 3 у низу (код објекта (5,7) горња ивица није строго већа од врха странице 5, као што се тражи), а последњи видљив је (11,20) на позицији 8 у низу. Слично важи и за наредна три упита.
У упиту (20,25), сви врхови почињу изнад висине 20, па први објекат који почиње изнад границе 20 не постоји (зато се исписује вредност 11), док је објекат (14, 18), на позицији 10 последњи који почиње испод границе 20.
У упиту (1,2) сви врхови почињу испод висине 2, па је објекат (3, 5) на позицији 0 први који почиње испод висине 1, док не постоји ни један објекат који почиње изнад висине 2 (па ни последњи такав) и као други број исписује се -1.

РЕШЕЊЕ:

1. Оптмизација учитавања: Пошто се у овом задатку преплићу фаза учитавања и исписа података, потребно је оптимизовати улаз и излаз и прилагодити га аутоматском тестирању (помоћу `cin.tie(0)` и коришћења `\n` уместо `endl`).

```
int main() {
  ios_base::sync_with_stdio(false); cin.tie(0);
```

2. **Математички модел**

По формулацији задатка видимо да су битне само висине горњих ивица објеката, тако да ћемо само њих учитати у низ, док ћемо доње ивице игнорисати.

Потребно је имплементирати функције које проналазе позицију првог елемента у низу чија је вредност строго већа од датог броја у упиту задатка и позицију последњег елемента у низу чија је вредност строго мања од датог броја у упиту задатка.

5. Мање ефикасно решење – линеарна претрага

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

using namespace std;
```

```
int prviVeci(const vector<int>& a, int x) {
 for (int i = 0; i < a.size(); i++)
 if (a[i] > x)
 return i;
 return a.size();
}

int poslednjiManji(const vector<int>& a, int x) {
 for (int i = 0; i < a.size(); i++)
 if (a[i] >= x)
 return i-1;
 return a.size() - 1;
}

int main() {
  ios_base::sync_with_stdio(false); cin.tie(0);
  // ucitavamo niz gornjih ivica objekata
  int n;
  cin >> n;
  vector<int> gornjeIvice(n);
  for (int i = 0; i < n; i++) {
    cin >> gornjeIvice[i];
    // donje ivice su irelevantne, pa ih ne pamtimo
    int _;
    cin >> _;
  }

  int m;
  cin >> m;
  for (int i = 0; i < m; i++) {
    // ucitavamo gornju i donju granicu
    int gornjaGranica, donjaGranica;
    cin >> gornjaGranica >> donjaGranica;
    // pozicija prvog objekta ciji je vrh strogo veci od pocetka skrola
    int pv = prviVeci(gornjeIvice, gornjaGranica);
    int pm = poslednjiManji(gornjeIvice, donjaGranica);
    cout << pv << " " << pm << '\n';
  }

  return 0;
}
```

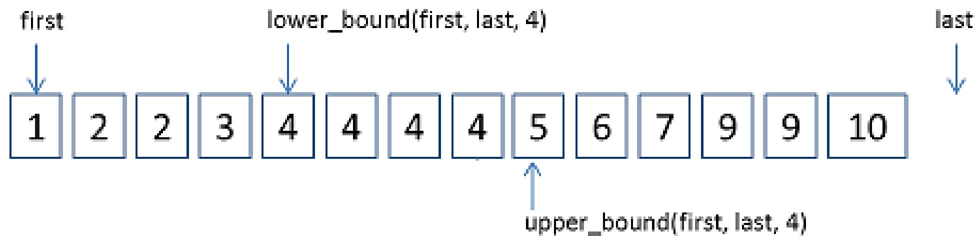
Сложеност овог приступа је линеарна (и износи O(n)), јер вршимо линеарну претрагу првог већег и последњег мањег елеемнта по читавом низу горњих граница.
Дакле, за m различитих упита (позиција скрола) имамо O(mn) операција .

Приметимо да другу претрагу можемо започети од позиције која је пронађена у првој, јер је висина (удаљеност од почетка странице) доњег дела скрола већа од висине његовог горњег дела, међутим, ово неће побољшати асимптотску сложеност.

Али, бинарна претрага ће побољшати асимптотску сложеност.

3. **Библиотечке функције** `lower_bound, upper_bound`

У језику C++ на располагању нам је библиотечка функција `upper_bound` која враћа итератор који указује на први елемент низа који је строго већи од траженог (ако су сви елементи мањи или једнаки од траженог, функција враћа итератор који указује непосредно иза краја низа).




*Syntax 1:*
*ForwardIterator upper_bound (ForwardIterator first, ForwardIterator last, const T& val);*
*Syntax 2:*
*ForwardIterator upper_bound (ForwardIterator first, ForwardIterator last, const T& val, Compare comp);*

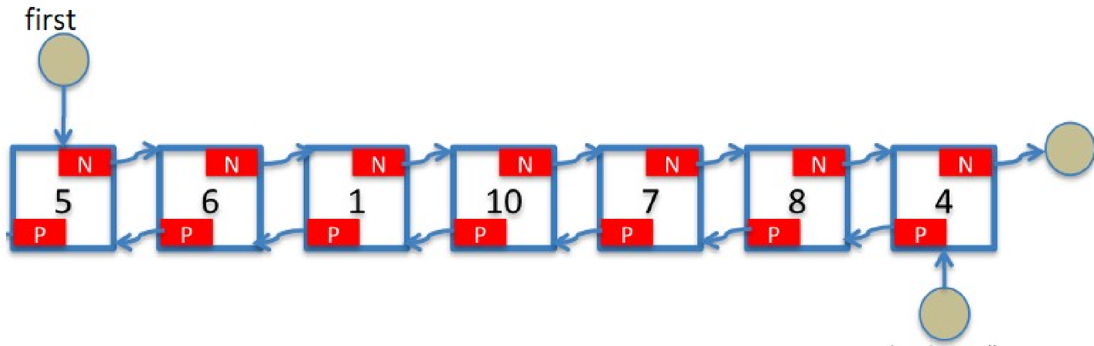
Функцијом `lower_bound` која је веома слична функцији `upper_bound` добијамо итератор који указује на први елемент који је већи или једнак од траженог (ако су сви елементи мањи од траженог, функција враћа итератор који указује непосредно иза краја низа).

Обе функције примају итератор на почетак низа који се претражује, итератор који указује непосредно иза краја низа и елемент који се тражи.
Сложеност обе функције је логаритамска у односу на број елемената у распону итератора који се претражује.

```
distance (InputIterator first, InputIterator last);
```

Return distance between iterators
Calculates the number of elements between *first* and *last*.

If *it* is a *random-access iterator*, the function uses `operator-` to calculate this. Otherwise, the function uses the increase operator (`operator++`) repeatedly.
Complexity
Constant for *random-access iterators*.
Otherwise, linear in *n*.




Функцијом `distance` можемо израчунати растојање између добијеног итератора и почетка низа и тако добити позицију у низу првог елемента који је строго већи тј. већи или једнак од траженог.

4. У нашем задатку, позиција последњег елемента који је строго мањи од траженог је једна позиција испред првог елемента који је већи или једнак од траженог (а који можемо одредити функцијом `lower_bound`). Позиција последњег елемента који је мањи или једнак од траженог је једна позиција испред првог елемента који је већи или једнак од траженог (а који можемо одредити функцијом `upper_bound`).

Сложеност обе библиотечке функције је O(log n). Пошто се претрага понавља m пута, укупна сложеност је O(m log n).

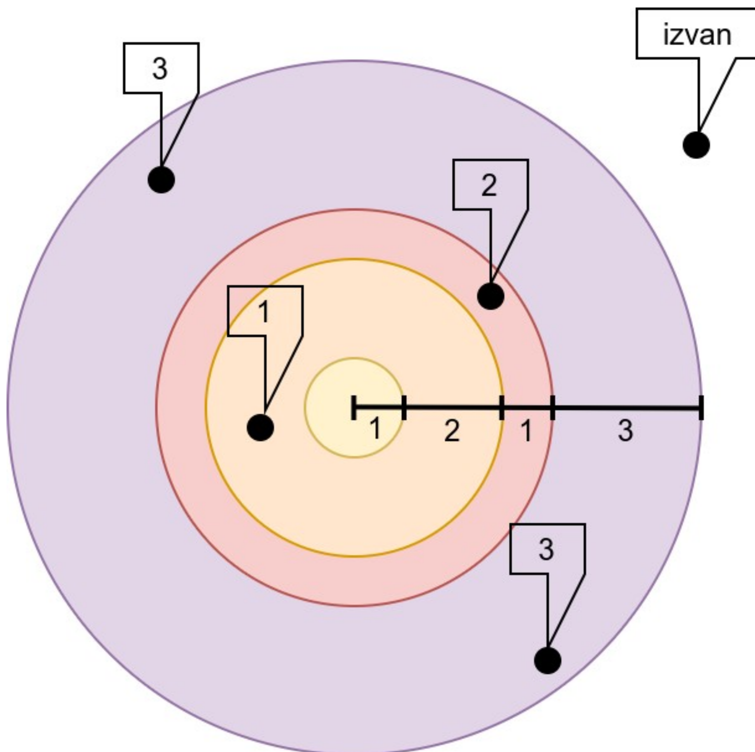
```
#include <iostream>
#include <algorithm>
#include <vector>
#include <iterator>

using namespace std;

int prviVeciIliJednak(const vector<int>& a, int x) {
  return distance(a.begin(), upper_bound(a.begin(), a.end(), x));
}

int poslednjiManji(const vector<int>& a, int x) {
  return distance(a.begin(), lower_bound(a.begin(), a.end(), x)) - 1;
}


int main() {
  ios_base::sync_with_stdio(false); cin.tie(0);
  // ucitavamo niz gornjih ivica objekata
  int n;
  cin >> n;
  vector<int> gornjeIvice(n);
  for (int i = 0; i < n; i++) {
    cin >> gornjeIvice[i];
    // donje ivice su irelevantne, pa ih preskacemo
```

```
    int _;
    cin >> _;
  }

  int m;
  cin >> m;
  for (int i = 0; i < m; i++) {
    // ucitavamo gornju i donju granicu
    int gornjaGranica, donjaGranica;
    cin >> gornjaGranica >> donjaGranica;
    // pozicija prvog objekta ciji je vrh strogo veci od pocetka skrola
    int pv = prviVeciIliJednak(gornjeIvice, gornjaGranica);
    int pm = poslednjiManji(gornjeIvice, donjaGranica);
    cout << pv << " " << pm << "\n";
  }

  return 0;
}
```

2.

Кружне зоне

Квалитет сигнала зависи од удаљености тачке од предајника. Простор је подељен у зоне облика кружних прстенова, при чему ширине прстенова могу бити међусобно различите (како је приказано на слици). Напиши програм који за дату тачку одређује зону којој припада.




Улаз

Са стандардног улаза уноси се број n (1≤n≤50000), а затим и n реалних бројева заокружених на две децимале, сваки у посебном реду, који представљају ширине свих кружних прстенова (за почетни прстен, тај број представља полупречник). Након тога се уноси број m (1≤m≤50000) и затим m парова координата тачака (у сваком реду се налазе два реалана броја заокружена на две децимале, раздвојена са по једним размаком).

Излаз

На стандардни излаз исписати m линија. У свакој линији исписати или индекс зоне (броје се од нуле) којој тачка припада или текст `izvan` ако је тачка изван последње зоне. Ако је тачка на граници две зоне, сматрати да припада унутрашњој.

Пример

Улаз

```
3
2.0

3.0

7.0

5

1.0 1.0

2.0 3.0

8.0 7.0

13.2 14.5

0.0 12.0
```

Излаз

```
0

1

2

izvan

2
```

РЕШЕЊЕ

Математички модел:

Тачка припада некој зони ако и само ако је њено Еуклидско растојање од координатног почетка (а њега можемо израчунати коришћењем Питагорине теореме) строго веће од унутрашњег, а мање или једнако спољашњем полупречнику те зоне.
Унутрашњи полупречник неке зоне једнак је збиру ширина свих зона пре ње, не укључујући њену ширину, а спољашњи полупречник једнак је збиру ширина свих зона пре ње, укључујући и њену ширину.

На пример, ако су ширине задате низом 1, 2, 1, 3 (као на слици), тада зоне редом одговарају следећим интервалима удаљености тачке од координатног почетка, а који су одређени унутрашњим и спољашњим полупречницима: [0, 1], (1, 3], (3, 4] и (4, 7].

Интервали су затворени здесна, јер тачка на граници по поставци задатка припада унутрашњој зони. Приметимо да је унутрашњи полупречник неке зоне једнак спољашњем полупречник претходне зоне (осим код зоне 0, која има унурашњи пречник нула). Зато је довољно да израчунамо спољашње полупречнике свих зона.

БИБЛИОТЕЧКА ФУНКЦИЈА  partial_sum

<numeric> partial_sum (first, last, result, binary_operator)

If *x* represents an element in `[first, last)` and *y* represents an element in *result*, the *y*s can be calculated as:

$y_0 = x_0$
$y_1 = x_0 + x_1$
$y_2 = x_0 + x_1 + x_2$
$y_3 = x_0 + x_1 + x_2 + x_3$
$y_4 = x_0 + x_1 + x_2 + x_3 + x_4$
... ... ...

```
#include <iostream>     // std::cout
#include <functional>   // std::multiplies
#include <numeric>      // std::partial_sum

int myop (int x, int y) {return x+y+1;}

int main () {
  int val[] = {1,2,3,4,5};
  int result[5];

  std::partial_sum (val, val+5, result);
  std::cout << "Podrazumevani partial_sum: ";
  for (int i=0; i<5; i++) std::cout << result[i] << ' ';
  std::cout << '\n';
   //Испис: 1, 1+2, 1+2+3, 1+2+3+4, 1+2+3+4+5

  std::partial_sum (val, val+5, result, std::multiplies<int>());
  std::cout << "Upotrebom funkcionala multiplies: ";
  for (int i=0; i<5; i++) std::cout << result[i] << ' ';
  std::cout << '\n';
   //Испис: 1, 1*2, 1*2*3, 1*2*3*4, 1*2*3*4*5

  std::partial_sum (val, val+5, result, myop);
  std::cout << "Upotrebom korisnicke funkcije function: ";
  for (int i=0; i<5; i++) std::cout << result[i] << ' ';
  //Испис: 1, 1+2+1, 4+3+1, 8+4+1, 13+5+1

  std::cout << '\n';
  return 0;
}
```

**Примена библиотечке функције partial_sum у задатку**

Полупречници зона се могу израчунати као парцијални збирови ширина зона. Њих можемо рачунати инкрементално, додајући сваку нову учитану ширину на текући збир.

У језику C++ збирове префикса можемо израчунати и библиотечком функцијом `partial_sum`.

```
  // izracunavamo poluprecnike zona (kao zbir sirina svih zona zakljucno sa tom)
  partial_sum(zone.begin(), zone.end(), zone.begin());
```

Овим је проблем сведен на одређивање ком од неколико надовезаних интервала реалне праве дата тачка припада.

Када је познат низ спољашњих полупречника зона (у нашем примеру то је низ 1, 3, 4, 7), зона у којој се тачка налази се може наћи тако што се нађе први спољашњи полупречник зоне који је већи или једнак растојању те тачке од координатног почетка -- то можемо урадити на сличан начин као у задатку Први већи и последњи мањи.
Његова позиција у низу је тражени број зоне, а ако такав спољашњи пречник не постоји, онда је тачка изван свих зона.

Мње ефикасно решење - линеарна претрага

Сложеност једне линеарне претраге је O(n), где је n број зона, те се зоне за m тачака одређују у времену O(mn).

Пошто су учитавање података и испис резултата испреплетани, потребно
је подесити улаз и излаз за аутоматско тестирање (коришћењем `cin.tie(0)`
и `\n` уместо `endl`).

Бинарна претрага

Ефикасан начин је да се примени бинарна претрага.

Библиотечке функције

Бинарна претрага се најлакше реализује ако се употреби њена
библиотечка имплементација.
У језику C++ можемо употребити библиотечку функцију `lower_bound`.

Сложеност једне бинарне претраге је O(log n), те се одређивање зона за $m$ тачака врши у сложености O(m log n).

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

using namespace std;

int main() {
  ios_base::sync_with_stdio(false); cin.tie(0);

  // ucitavamo niz sirina zona
```

```
  int n;
  cin >> n;
  vector<double> zone(n);
  for (int i = 0; i < n; i++)
    cin >> zone[i];

  // izracunavamo poluprecnike zona (kao zbir sirina svih zona zakljucno sa tom)
  partial_sum(zone.begin(), zone.end(), zone.begin());

  // ucitavamo m tacaka
  int m;
  cin >> m;
  for (int i = 0; i < m; i++) {
    // ucitavamo koordinate tacke (x, y)
    double x, y;
    cin >> x >> y;
    // rastojanje tacke (x, y) od koordinatnog pocetka
    double r = sqrt(x*x + y*y);

    // pronalazimo prvu zonu takvu da joj je poluprecnik veci ili jednak r
    auto it = lower_bound(zone.begin(), zone.end(), r);

    // ako takva ne postoji, tacka je izvan svih zona
    if (it == zone.end())
      cout << "izvan" << endl;
    else
      cout << distance(zone.begin(), it) << '\n';
  }
  return 0;
}
```