Бинарный поиск

binary_search

Алгоритм `binary_search(A.begin(), A.end(),val)` осуществляет двоичный поиск значения `val`. Контейнер должен быть упорядочен. Возвращается значение типа `bool`, то есть `true` или `false` в зависимости от того, есть ли такой элемент в контейнере.

lower_bound

Алгоритм `lower_bound(A.begin(), A.end(), val)` осуществляет двоичный поиск значения `val` и возвращает итератор `res` на первый элемент, который не меньше, чем `val`, то есть `*res>=val`, `*(res-1)<val`.

Если все элементы контейнера (начиная с `A.begin()`) будут не меньше, чем `val`, то будет возвращено значение `A.begin()`. Если в контейнере все элементы меньше `val`, то возвращается значение `A.end()`.

upper_bound

Алгоритм `upper_bound(A.begin(), A.end(), val)` осуществляет двоичный поиск значения `val` и возвращает итератор `res` на первый элемент, который строго больше, чем `val`, то есть `*res>val`, `*(res-1)<=val`. Если же все элементы контейнера (начиная с `A.begin(),`) будут больше, чем `val`, то будет возвращено значение `A.begin(),`. Если в контейнере все элементы меньше или равны `val`, то возвращается значение `A.end()`.

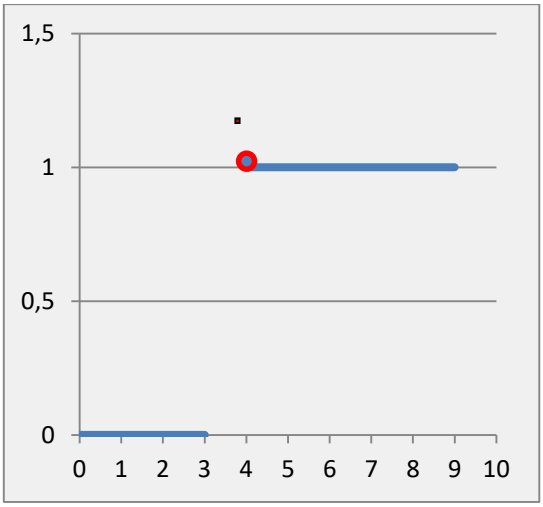
Левое (x <=A[i]) и правое (x < A[i]) вхождение x (ищем i)

B = [11, 23, 34, 45, 68, 73, 87]
0 1 2 3 4 5 6
X = 68
4 5
X = 100
7 7
X = 46
4 4

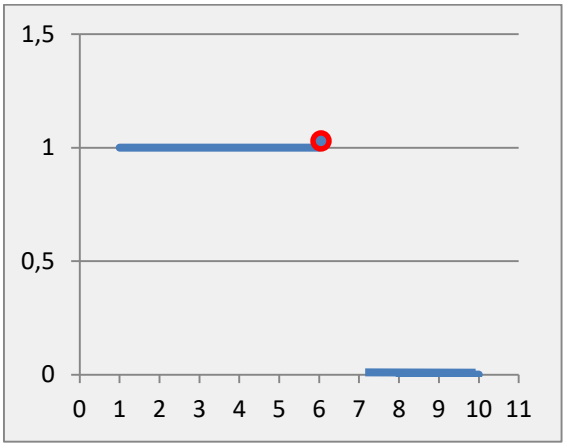
upper_bound == lower_bound
нет эл-та

upper_bound - lower_bound
количество эл-тов

Двоичный поиск по ответу (логическая функция)




```
(l;r]
while(r - l >1){
        m =(l + r)/2;
        if(f(m)){
            r = m;
        }
        else{
            l = m;
        }
}
cout << r;
```




```
[l;r)
while(r - l >1){
        m =(l + r)/2;
        if(f(m)){
           l= m;
        }
        else{
           r= m;
        }
}
cout <<l;
```