# Binary Search Problems

Q: <https://codeforces.com/contest/1486/problem/C1>

There is an array *a* of *n* **different** numbers. In one query you can ask the position of the second maximum element in a subsegment *a*[*l*..*r*]. Find the position of the maximum element in the array in no more than **40** queries.

A subsegment *a*[*l*..*r*] is all the elements *al*,*al*+1,...,*ar*. After asking this subsegment you will be given the position of the second maximum from this subsegment **in the whole** array.

a= [5,1,4,2,3]

l=1,r=5 -> 3

l=3,r=5 -> 5

**In interactive problems:**

**Judge** **You**

(Knows the array) (can ask queries and get answer)

int x; cout<<”? “<<l<<” “<<r<<endl;

cin>>x;

a= [5,1,4,2,3]

int n;

cin>>n;

int l=1,r=5;

cout<<”? “<<l<<” “<<r<<endl;

int x;

cin>>x; // x=3

cout<<”? “<<4<<” “<<5<<endl;

cin>>x; // x=4

int ans;

cout<<”! “<<ans<<endl;

1……….n

(1,n) -> z

int mid = (n+1)/2;

(1,mid) and (mid+1,n)

a1,a2,a3,a4 -> (a1,a2) (a3,a4) -> =z

(a2,a4) (a1,a3) ->!=z

(a2,a3) (a1,a4) -> !=z

a2 is in first half(1...mid)

(1,mid) -> x

log(n) -> (15-17)\*2 <=40

Sol:

| map<pair<int,int>,int> m; int query(int l,int r) {  if(m.find(mp(l,r))!=m.end()){  return m[mp(l,r)];  }  cout<<"? "<<l<<" "<<r<<endl;  int z;  cin>>z;  m[mp(l,r)]=z;  return z; } void solve() {  int n;  cin>>n; //5,1,4,2,3  int l=1,r=n; //l=1,r=3  while(r-l>1){  int mid = (l+r)/2;  int z = query(l,r);  if(z<=mid){ //a2 is present in the first half  int x = query(l,mid);  if(x==z){ //a1 and a2 present in l...mid -> go left  r=mid;  }else{ // a1 and a2 present in different halfs....go right  l=mid;  }  }else{ //a2 is present in 2nd half(mid+1...r)  int x = query(mid,r);  if(x==z){ // a1 and a2 present in same half...mid to r  l=mid;  }else{ // a1 and a2 present in diff half...go l...mid  r=mid;  }  }  }  int z = query(l,r);  if(z==l){  cout<<"! "<<r<<endl;  }else{  cout<<"! "<<l<<endl;  } } |
| --- |

Q: <https://atcoder.jp/contests/abc174/tasks/abc174_e>

| #include<bits/stdc++.h> using namespace std;  int32\_t main(){    int n,k;  scanf("%d%d",&n,&k);  vector<int> a(n);  for(int i=0;i<n;i++) cin>>a[i];  int low=1,high=1e9;  // search space-> length of max stick;  int ans=-1;  while(low<=high){  int mid=low+(high-low)/2;  int req\_number\_of\_cuts=0;  for(int i=0;i<n;i++){  if(a[i]%mid!=0) req\_number\_of\_cuts+=(a[i]/mid);  else req\_number\_of\_cuts+=((a[i]/mid)-1);  }  if(req\_number\_of\_cuts<=k){  ans=mid;  high=mid-1;  }  else low=mid+1;  }  cout<<ans<<endl; } |
| --- |
|  |

**Some useful built-in functions for Binary Search in C++:**

1. If you are given a sorted vector (a[i] <= a[i+1]) and a number x, and you need to find the index of the first element >=x, this is called **lower\_bound of x**.

In C++, there is a builtin function to find this in O(log N) using binary search.

Eg:

| int ind = lower\_bound(a.begin(), a.end(), x) - a.begin(); |
| --- |

2. If you are given a sorted vector (a[i] <= a[i+1]) and a number x, and you need to find the index of the first element >x, this is called **upper\_bound of x**.

In C++, there is a builtin function to find this in O(log N) using binary search.

| int ind = upper\_bound(a.begin(), a.end(), x) - a.begin(); |
| --- |

Suppose, there is no such element, then ind = size of the array.

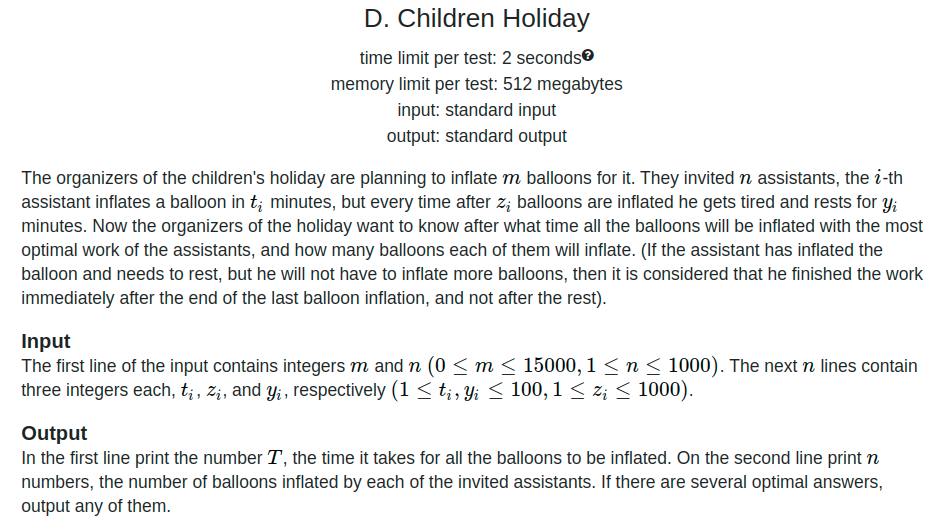
Q. <https://codeforces.com/edu/course/2/lesson/6/1/practice/contest/283911/problem/D>

| #include <bits/stdc++.h>  using namespace std;  int32\_t main() {  int n;  cin >> n;   vector<int> vec(n);   for (int i = 0; i < n; i++) {  cin >> vec[i];  }   sort(vec.begin(), vec.end());   int k;  cin >> k;   int l, r;  while (k--) {  cin >> l >> r;  int y = upper\_bound(vec.begin(), vec.end(), r) - vec.begin();  int x = lower\_bound(vec.begin(), vec.end(), l) - vec.begin();   cout << y - x << ' ';  }   return 0; } |
| --- |

Q. <https://codeforces.com/edu/course/2/lesson/6/2/practice/contest/283932/problem/E>

| #include <bits/stdc++.h>  using namespace std;  typedef long double ld;  const ld eps = 1e-6; // 10^(-6)  int32\_t main() {  // fastio;  ld c;  cin >> c;   long double lo = 1, hi = 1e10, mid;   while (hi - lo >= eps) {  mid = lo + (hi - lo) / 2;   ld val = mid \* mid + sqrt(mid);   if (val <= c) {  lo = mid;  } else {  hi = mid;  }  }   cout << fixed << setprecision(7) << mid << endl;   return 0; } |
| --- |

Q. <https://codeforces.com/edu/course/2/lesson/6/2/practice/contest/283932/problem/D>

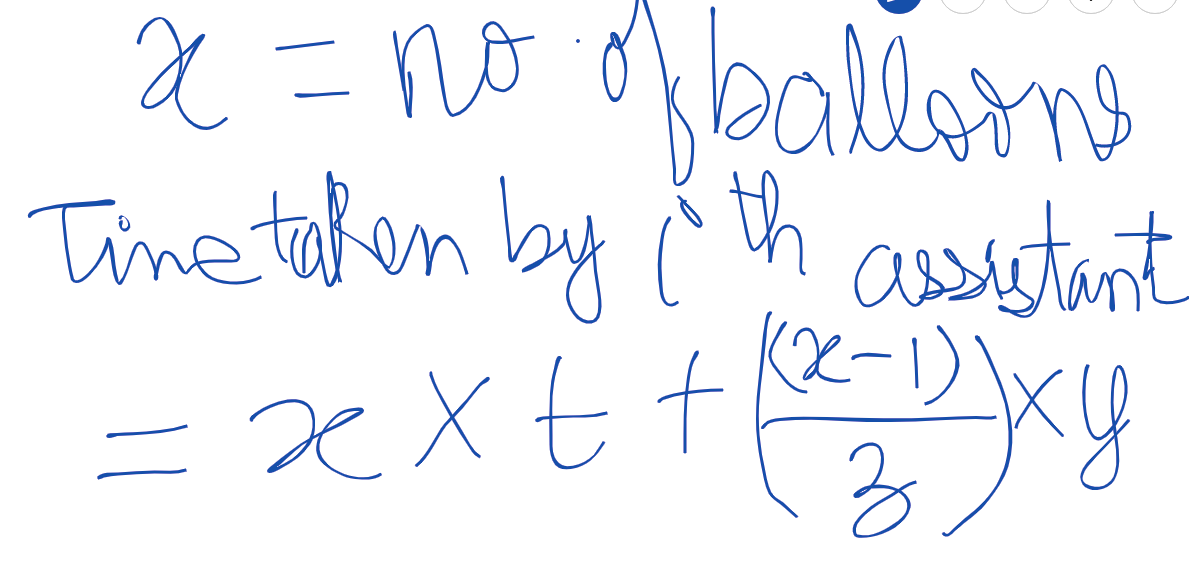


Let f(x) = No. of balloons that can be inflated in x minutes

It is a monotonic non-decreasing function.

Let g(x) = Time taken by one person to inflate x balloons.

It is a monotonic non-decreasing function.



<https://codeforces.com/edu/course/2/lesson/6/2/practice/contest/283932/problem/D>

| #include <bits/stdc++.h> #define int long long  using namespace std;  int get\_balloons(int p, int ti, int zi, int yi) {  // p: time provided  // return the maximum number of balloons that  // this person can fill in t=x minutes   int lo = 0;  int hi = 1e8;  int mid;  int ans = 0;  while (lo <= hi) {  mid = lo + (hi - lo) / 2;  int gx = mid \* ti + ((mid - 1) / zi) \* yi;  // time taken to fill x=mid balloons  if (gx <= p) {  ans = mid;  lo = mid + 1;  } else {  hi = mid - 1;  }  }  return ans; }  int32\_t main() {  // fastio;   int m, n;  cin >> m >> n;   vector<int> t(n), z(n), y(n);   for (int i = 0; i < n; i++) {  cin >> t[i] >> z[i] >> y[i];  }   int lo = 0;  int hi = 1e8;  int mid;   int ans;   vector<int> balloons;   while (lo <= hi) {  mid = lo + (hi - lo) / 2;   // Find the number of balloons that can be  // inflated in x=mid minutes   int num = 0; // no. of balloons filled  vector<int> temp;  for (int i = 0; i < n; i++) {  if (num == m) {  temp.push\_back(0);  continue;  }  int cnt = get\_balloons(mid, t[i], z[i], y[i]);  num += cnt;  if (num >= m) {  int diff = num - m;  cnt -= diff;  num = m;  }  temp.push\_back(cnt);  }   if (num >= m) {  balloons = temp;  ans = mid;  hi = mid - 1;  } else {  lo = mid + 1;  }  }   cout << ans << '\n';   for (int i = 0; i < balloons.size(); i++) {  cout << balloons[i] << ' ';  }  return 0; } |
| --- |

**Time complexity:** O ( log (108) \* n \* log (108) )