Binary Search

Searching an element in a Sorted Sequence

```
binary_search(A, target):
   lo = 1, hi = size(A)
   while lo <= hi:
      mid = lo + (hi-lo)/2
      if A[mid] == target:
         return mid
      else if A[mid] < target:
         lo = mid+1
      else:
         hi = mid-1
```

First Occurrence of an element

```
FirstOccur(A, target):
   lo = 1, hi = size(A), result=-1;
   while lo <= hi:
      mid = lo + (hi-lo)/2
      if A[mid] == target:
         result=mid; high=mid-1;
```

```
        else if A[mid] < target:
           lo = mid+1
        else:
           hi = mid-1
      return res;
```

Last Occcurrence of an element

```
LastOccur(A, target):
   lo = 1, hi = size(A), result=-1;
   while lo <= hi:
      mid = lo + (hi-lo)/2
      if A[mid] == target:
         result=mid; low=mid+1;
      else if A[mid] < target:
         lo = mid+1
      else:
         hi = mid-1
    return res;
```

Beyond Sorted arrays

Binary search obviously works on searching for elements in a sorted array. But if you think about the reason why it works is because the array itself is monotonic (either increasing or decreasing). So, if you are at a particular position, you can make a definite call whether the answer lies in the left part of the position or the right part of it. But Most importantly you need to know the range **[start,end]**

Similar thing can be done with monotonic functions (monotonically increasing or decreasing) as well.
Lets say we have f(x) which increases when x increases.

So, given a problem of finding x so that **f(x) = p**, I can do a binary search for x.
/*Take Example*/
Therefore

1. if **f(current_position) > p**, then I will search for values lower than current position.
2. if **f(current_position) < p**, then I will search for values higher than current position
3. if **f(current_position) = p**, then I have found my answer.

Optimisation Problems

Consider an optimisation problem where you need to minimise a quantity X satisfying some constraint Y such that:

->If X satisfies the constraint, anything more than X will satisfy the constraint too

->For a given value of X, you know how to check whether the constraint is satisfied

Painter's Partition Problem

You have to paint N boards of length {A0, A1, A2, A3 … AN-1}. There are K painters available and you are also given how much time a painter takes to paint 1 unit of board. You have to get this job done **as soon as** possible under the constraints that any painter will only paint contiguous sections of board.

/Take example/

Observations

->The lowest possible value for costmax must be the maximum element in A (name this as lo).

->The highest possible value for costmax must be the entire sum of A, (name this as hi).

->As costmax increases, x decreases. The opposite also holds true.

```
int painterNum(vector<int> &C, long long X){
    long long sum = 0;
    long long num = 1;
    for(auto c: C){
        if(sum + c > X){
            sum = c;num++;
        }else sum += c;
    }
    return num;
}
```

```
int paint(int A, int B, vector<int> &C) {

    int high=0,low=0;
    for(int x: C){low = max(x, low); high += x;}
    while(low < high){
        long long mid = low + (high-low)>>1;
        if(painterNum(C, mid) <= A)high = mid;
        else low = mid+1;
    }

    return low;
}
```

Allocate books

N number of books are given.

The ith book has Pi number of pages.

You have to allocate books to M number of students so that maximum number of pages alloted to a student is minimum. A book will be allocated to exactly one student. Each student has to be allocated at least one book. Allotment should be in contiguous order, for example: A student cannot be allocated book 1 and book 3, skipping book 2.

Return -1 if a valid assignment is not possible

```
int painterNum(vector<int> &C, long long X){
    long long sum = 0;
    long long num = 1;
    for(auto c: C){
        if(sum + c > X){
            sum = c;num++;
        }else sum += c;
    }
    return num;
}
int paint(int A, int B, vector<int> &C) {

    int high=0,low=0;
    if(B > A.size())return -1;
    for(int x: C){low = max(x, low); high += x;}
    while(low < high){
        long long mid = low + (high -low)>>1;
        if(painterNum(C, mid) <= A)high = mid;
        else low = mid+1;
    }

    return low;
}
```

Problem

Aggressive cows

http://www.spoj.com/problems/AGGRCOW/

Approach:

Define the following function:

Func(x) = 1 if it is possible to arrange the cows in stalls such that the distance between any two cows is at least x

Func(x) = 0 otherwise

The problem satisfies the monotonicity condition necessary for binary search.Why??

Check that if Func(x)=0, Func(y)=0 for all y>x and if Func(x)=1, Funx(y)=1 for all y < x

Step 2:

Find low and high values

Func(0)=1 trivially since the distance between any two cows is at least 0. Also, since we have at least two cows, the best we can do is push them towards the stalls at the end - so there is no way we can achieve better. Hence Func(maxbarnpos-minbarnpos+1)=0.

```
#include <bits/stdc++.h>
using namespace std;
int n,c;
int func(int num,int array[])
{
    int cows=1,pos=array[0];
    for (int i=1; i<n; i++)
    {
        if (array[i]-pos>=num)
        {
            cows++;
            if (cows==c)
                return 1;
            pos=array[i];
        }
    }
    return 0;
}
int bs(int array[])
{
    int ini=0,last=array[n-1],max=-1;
    while (last>ini)
    {
        int mid=(ini+last)/2;
        if (func(mid,array)==1)
        {
            if (mid>max)
                max=mid;
            ini=mid+1;
        }
        else
            last=mid;
    }
    return max;
```

```
}
int main()
{
    int t;
    scanf("%d",&t);
    while (t--)
    {
        scanf("%d %d",&n,&c);
        int array[n];
        for (int i=0; i<n; i++)
            scanf("%d",&array[i]);
        sort(array,array+n);
        int k=bs(array);
        printf("%dn",k);
    }
    return 0;
}
```