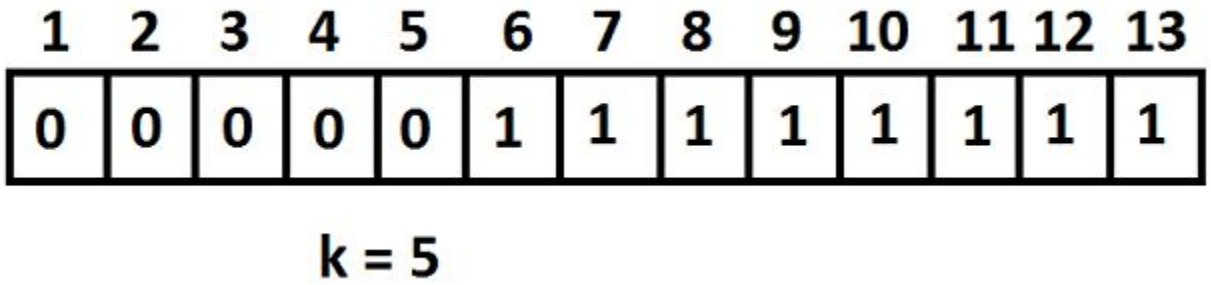
Advanced Binary Search

So, you have heard of how binary search is used to find the position of an element in a sorted array. That's not the end of the picture. Binary search is a lot more powerful than that. In this class, we will learn some non trivial applications of binary search.

Problem Statement: You are given an array a[1 ... N]. There is some k for which, a[i] = 0 for all i in the range 1 <= i <= k, and a[i] = 1 for all i in the range k + 1 <= i <= N. We want to find this position k which splits the groups of 0s and 1s

Consider this example first:



Naive approach: Iterate from right to left and return the value of k at which 0 is seen for the first time. This approach has a time-complexity of O(N) but can we do better?
Yes, we can.

Better approach:

1. set lo = 1 and hi = N (where N is the size of the array)
2. set mid = (hi + lo + 1) / 2
3. if a[mid] is 1, then our k lies in the part a[lo .. mid - 1]
4. if a[mid] is 0, then our k lies in the part a[mid .. hi]
5. solve recursively until lo and hi converge

Time Complexity: Since the search space is always reduced to half the time complexity of the solution is O(logN).

Code:

```
void findk(int a[],int n) {
    int lo = 0, hi = n - 1 ,ans=-1;
    while(lo <= hi) {
        int mid = (lo + hi) / 2 ;
        if(a[mid] == 0)
        {
            ans = mid;
            lo = mid + 1;
        }
        else
        {
            hi = mid - 1 ;
        }
    }
    if(ans!=-1)
        printf("%d\n", ans) ;
    else // this means ans==-1
        printf("Array is all 1s\n") ;
}
```

Question - Aggressive Cow

- Farmer John has built a new long barn, with N (2 <= N <= 100,000) stalls. The stalls are located along a straight line at positions x1,...,xN (0 <= xi <= 1,000,000,000).
- His C (2 <= C <= N) cows don't like this barn layout and become aggressive towards each other once put into a stall. To prevent the cows from hurting each other, FJ wants to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible. What is the largest minimum distance?

N: 6

C: 4

$\geq x$

f(3) $\Rightarrow$ True

f(4) $\Rightarrow$ False

```cpp
// check if a distance of x is possible b/w each cow
bool chk(int x)
{
  // greedy approach, put each cow in the first place you can
  int cows_placed = 1, last_pos = A[0];
  for (int i = 1; i < N; i++)
  {
    if ((A[i] - last_pos) >= x)
    {
      if (++cows_placed == C)
        return true;
      last_pos = A[i];
    }
  }
  return false;
}
```

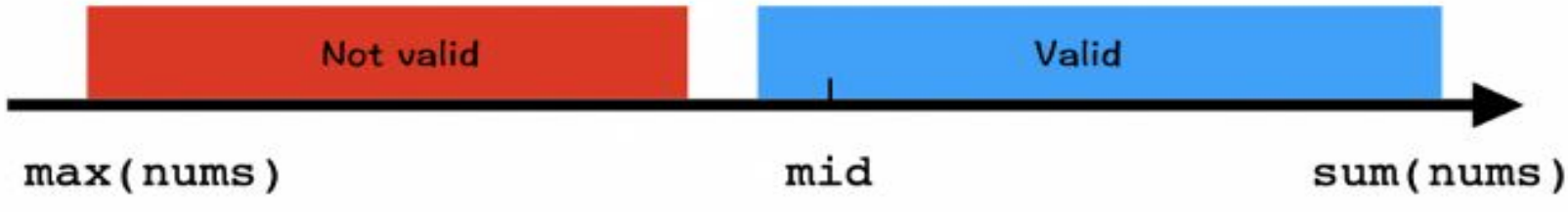
```cpp
#include <bits/stdc++.h>
using namespace std;
int N, C;
long long A[100000];
int main()
{
  int T;
  cin >> T;
  while (T--)
  {
    cin >> N >> C;
    for (int i = 0; i < N; i++)
      cin >> A[i];
    sort(A, A + N);

    // binary search
    long long low = 0, high = 1000000000, mid, pos = 0;
    while (high >= low)
    {
      mid = (high + low) / 2;
      if (chk(mid))
      {
        low = mid + 1;
        pos = mid;
      }
      else
      {
        high = mid - 1;
      }
    }
    cout << pos << endl;
  }
  return 0;
}
```

Introductory Steps to write code in C++

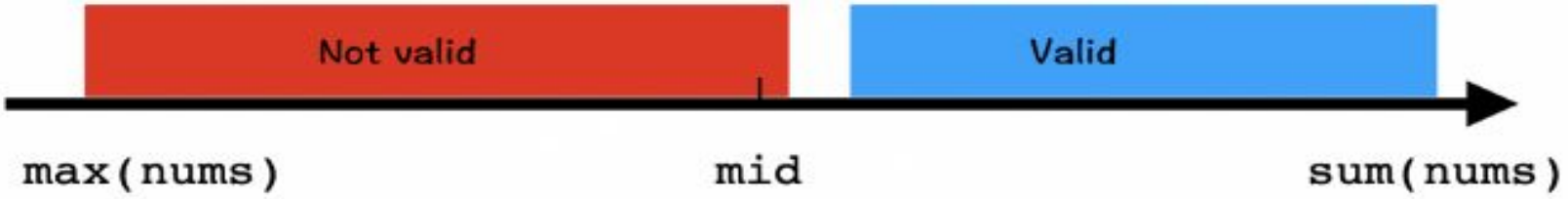
1. Use file name extension **.cpp**
2. Use only one header file for all
   bits/stdc++.h
3. include extra command
   using namespace std;
   just after including header file
4. compile using **g++** instead of **gcc**

Lower Bound



Asking for the lower bound and mid works:

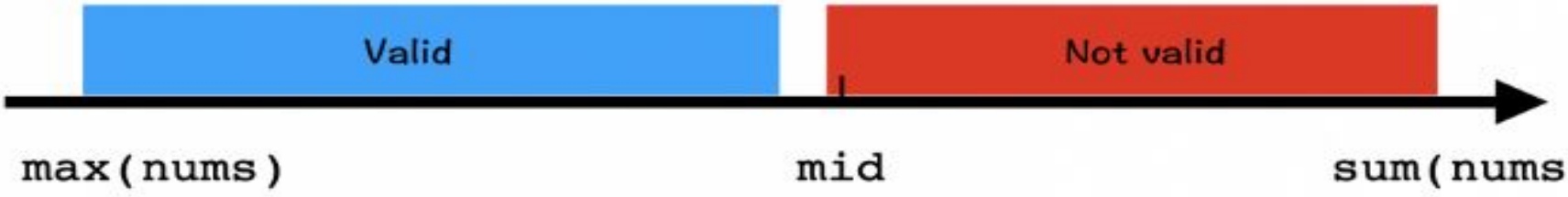
```
r = mid
```



Asking for the lower bound and mid does not work:

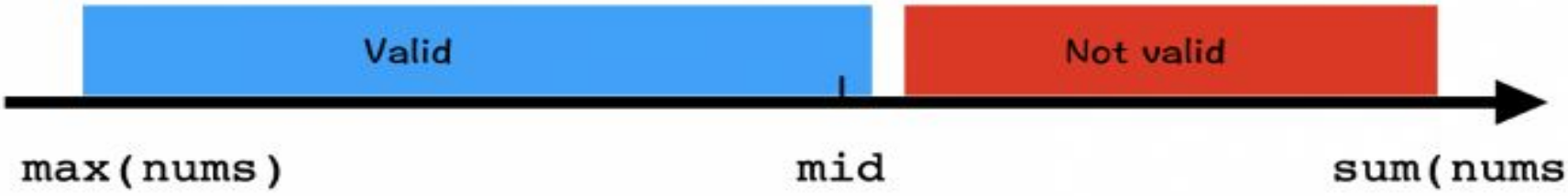
```
l = mid + 1
```

Upper Bound



Asking for the upper bound and mid does not work:

```
r = mid - 1
```



Asking for the upper bound and mid works:

```
l = mid
```

INTRODUCTION TO STL(Standard Template Library)

Vector:

A vector is an array like container that improves on the C++ array types. In particular it is not necessary to know how big you want the vector to be when you declare it, you can add new elements to the end of a vector using the ***push_back*** function. (In fact the ***insert*** function allows you insert new elements at any position of the vector, but this is a very inefficient operation -- if you need to do this often consider using a list instead).

Declaration:

1) vector <data type> vector_name : In this declaration a vector of 0 length is initialized.
2) vector <data type> vector_name(len) : In this declaration a vector of length len is initialized;
3) vector <data type> vector_name(len,val) : In this declaration the vector is of length len is initialized where elements are initialized with a value val.

Access: You can access the elements of a vector by using operator[] i.e. we can access the ith element of vector v as v[i].

Functions of vector:

1. push_back(): It push the elements into a vector from the back
2. empty(): Returns whether the container is empty.
3. size(): Returns the number of elements in the vector.
4. clear(): It is used to remove all the elements of the vector container
5. erase(): It is used to remove elements from a container from the specified position or range
6. insert(): It inserts new elements before the element at the specified position
7. front(): Returns a reference to the first element in the vector.
8. back(): Returns a reference to the last element in the vector