

UPSKILL WITH US

Searching 2



Gyanu Mayank

17th February, 2023



Agenda

1. Advanced Problems Based on Binary Search

Problems

1. Aggressive Cows

There is a new barn with N stalls and C cows. The stalls are located on a straight line at positions x1,...,xN (0 <= xi <= 1,000,000,000). We want 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?

Examples:

Input: No of stalls = 5 Array: {1,2,8,4,9} And number of cows: 3

About Bosscoder!

Bosscoder Academy is a platform for ambitious engineers who want to upskill and achieve great heights in their careers.

Our world-class program provides the learners with:

- Structured curriculum designed by experts.
- Covers Data
 Structures & Algorithms,
 System Design & Project
 Development.
- Live Interactive Classes from Top PBC engineers.
- ✓ 1:1 Mentorship
- Quick Doubt Support
- Advanced Placement Support

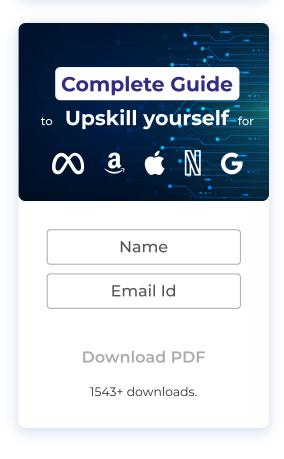
Want to upskill to achieve your dream of working at



UPSKILL WITH US

Learn now we can neip.

After sorting the array, we set a minimum distance, then we keep on increasing until accommodation of all cows is not possible. We stop *just* before that to get our answer, which in this example is 3. For checking if the cows can fit or not, we are simply iterating over our n stalls, and for every stall with the said minimum distance, we place our cow. After we are done, if all cows have been accommodated, we return true, otherwise false. Let's observe the time complexity of our brute force algorithm here, we are increasing distance in each step (which in the worst case of two cows gets as high as m = array[n-1]array[0]), and in that step, we are checking if our cows can "fit", so we are iterating again for each step to check.





```
K = Inp[I];
        }
    return false;
}
int main()
{
    int n, m;
    cin >> n >> m;
    vector<int> inp(n);
    for (int i = 0; i < n; i++)
        cin >> inp[i];
    sort(inp.begin(), inp.end());
    int maxD = inp[n - 1] - inp[0];
    int ans = INT_MIN;
    for (int d = 1; d \leftarrow maxD; d++)
    {
        bool possible = isCompatible(inp, c
        if (possible)
        {
             ans = max(ans, d);
        }
    }
    cout << ans << '\n';</pre>
}
```

Optimised Approach

Make sure to sort the array first, because only then it makes sense to use binary search. For the BS approach, we set low = 1 because the minimum distance is 1 and high =array[n-1] – array[0]. because that's the maximum possible distance between two stalls. Let's calculate our mid-value after this.mid = low + (high-low)/2.



means we can set our upper bound as high-1, and if it is possible, we store it in an answer variable and set our lower bound as mid+1. We keep on doing this until high and low pointers are equal.

```
#include <bits/stdc++.h>
using namespace std;
bool isPossible(int a[], int n, int cows, i
{
    int cntCows = 1;
    int lastPlacedCow = a[0];
    for (int i = 1; i < n; i++)
    {
        if (a[i] - lastPlacedCow >= minDist
        {
            cntCows++;
            lastPlacedCow = a[i];
        }
    }
    if (cntCows >= cows)
        return true;
    return false;
}
int main()
{
    int n, cows;
    cin >> n >> cows;
    int a[n];
    for (int i = 0; i < n; i++)
        cin >> a[i];
```



UPSKILL WITH US

```
int iow = i, nign = a[n - i] - a[0];
    while (low <= high)
    {
        int mid = (low + high) >> 1;
        if (isPossible(a, n, cows, mid))
        {
             low = mid + 1;
        }
        else
        {
             high = mid - 1;
        }
    }
    cout << high << endl;</pre>
    return 0;
}
Time Complexity: O(N*log(M)).
```

Time Complexity: O(N*log(M)). Space Complexity: O(1)

2. Smallest Good Base

Given an integer n represented as a string, return the smallest good base of n.We call k >= 2 a good base of n, if all digits of n base k are 1's.

Example:

Input: n = "13"

Output: "3"

Explanation: 13 base 3 is 111.

<u>Approach</u>

The input can be stored in a long long int, use



smallest possible pase is $\kappa=2$, with has the longest possible representation, i.e., largest d. So, to find the smallest base means to find the longest possible representation "11111....1" based on k. As $n<=10^18$, so n<(1<<62). We iterate the length of the representation from 62 to 2 (2 can always be valid, with base=n-1), and check whether a given length can be valid. For a given length d, we use binary search to check whether there is a base k which satisfies $1+k^1+k^2+...k^d=n$. The left limit is 1, and the right limit is pow(n,1/d)+1, i.e., the d-th square root of n.

```
#include <bits/stdc++.h>
using namespace std;
unsigned long long mysolve(unsigned long lo
{
    double tn = (double)n;
    unsigned long long right = (unsigned lo
    unsigned long long left = 1;
    while (left <= right)</pre>
    {
        unsigned long long mid = left + (ri
        unsigned long long sum = 1, cur = 1
        for (int i = 1; i <= d; i++)
        {
            cur *= mid;
            sum += cur;
        }
        if (sum == n)
            return mid;
```



```
erse
             left = mid + 1;
    return 0;
}
string smallestGoodBase(string n)
{
    unsigned long long tn = (unsigned long
    unsigned long long x = 1;
    for (int i = 62; i >= 1; i--)
    {
        if ((x << i) < tn)
        {
             unsigned long long cur = mysolv
             if (cur != 0)
                 return to string(cur);
        }
    return to_string(tn - 1);
int main()
{
    string s;
    cin >> s;
    cout << smallestGoodBase(s);</pre>
    return 0;
}
```

Time Complexity: O(logN)
Space Complexity:O(1)

3. Minimum Size Subarray Sum

Given an array of positive integers nums and a positive integer, target return the minimal



UPSKILL WITH US

suparray, return v instead.

Example

Input: target = 7, nums = [2,3,1,2,4,3]

Output: 2

Explanation: The subarray [4,3] has a minimal length under the problem constraint.

Sliding Window-Based Approach

We can use a sliding window in this question. Initialise two variables and increment the second one when the sum exceeds the target else increment the first one and find the minimum range for desired subarray sum.

```
#include <bits/stdc++.h>
using namespace std;
int minSubArrayLen(int target, vector<int>
{
    int l = 0, h = 0;
    int s = 0, ans = nums.size();
    for (int i : nums)
        s += i;
    if (s < target)</pre>
        return 0;
    s = 0;
    for (int i = 0; i < nums.size(); i++)</pre>
    {
        s += nums[i];
        while (s - nums[l] >= target)
             s -= nums[1++];
        if (s >= target)
             ans = min(ans, i - l + 1);
    }
```



```
int main()
{
    int n;
    cin >> n;
    int target;
    cin >> target;
    vector<int> nums(n);
    for (int i = 0; i < n; i++)
    {
        cin >> nums[i];
    }
    cout << minSubArrayLen(target, nums);
    return 0;
}</pre>
```

Binary Search-Based Approach

So if we clearly see despite not having a sorted array we can do a binary search on the answer because our answer can lie between 1 and n otherwise that will not be a subarray present. So do a binary search from 1 to n and check for each mid if it will give us a subarray having a sum target and continue decreasing it by doing hi=mid-1.

```
#include <bits/stdc++.h>
using namespace std;

bool checker(vector<int> &arr, int k, int n
{
   int sum = 0;
   for (int i = 0; i < mid; i++)
      sum += arr[i];</pre>
```

Searching 2



```
wniie (r < arr.size())
    {
        sum -= arr[1++];
        sum += arr[r++];
        maxi = max(sum, maxi);
    }
    return maxi >= k;
}
int minSubArrayLen(int k, vector<int> &arr)
{
    int n = arr.size();
    int low = 1, high = n;
    int ans = 0;
    while (low <= high)
    {
        int mid = low + (high - low) / 2;
        if (checker(arr, k, mid))
        {
            ans = mid;
            high = mid - 1;
        else
            low = mid + 1;
    }
    return ans;
}
int main()
{
#ifndef ONLIINE JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif
    int n;
    cin >> n;
    int target;
```

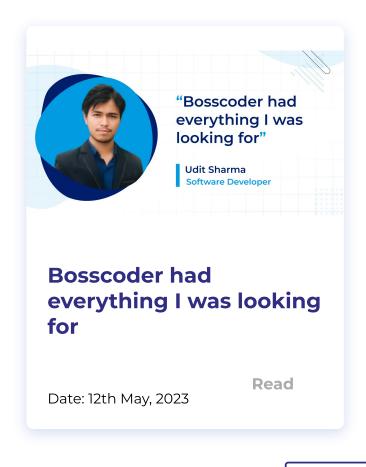


UPSKILL WITH US

```
Tor (Int 1 = 0; 1 < n; 1++)
{
      cin >> nums[i];
}
cout << minSubArrayLen(target, nums);
return 0;
}</pre>
```

Time Complexity: O(nlogn)
Space Complexity: O(1)







View All blogs



UPSKILL WITH US

ACADERY

wiio are we

CUIILACL US

Email: ask@bosscoderacademy.com

ruliuw us uli

Helping ambitious

learners upskill

themselves

& shift to

top product based

companies.

Lets hear all about it.

About us

Blog

Attend a FREE Event

Privacy Policy

Terms & Condition

Pricing and Refund Policy

in LinkedIn

Youtube O Instagram

⊘ Telegram

Q Reviews on Quora