Todays Content

1. Aggresive Cows
2. When to apply BS.
    a. When can we discard search space : Monotonic 
3. Smallest subarray with sum >= k.

20) Given N Cows & M stalls, all stalls are on x-axis at different locations, Place all N cows in such a way that min distance between any 2 cows is maximized. Maximize min distance

#Note1 In a stall only 1 cow can be present

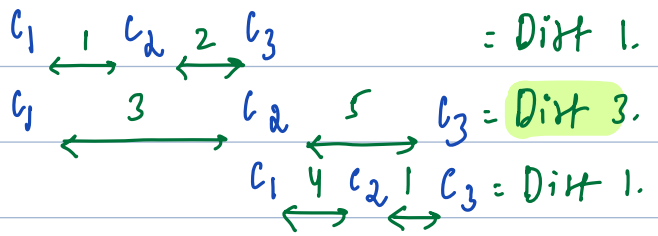
#Note2 All cows have to placed, stalls >= N cows

#Note3 All stall positions are sorted, if not sorted we can sort them.

#Eg1: x-axis[]

#Stalls = 5 0 1 2 3 4

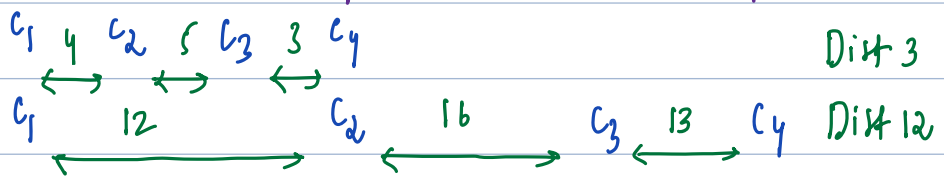
Cows = 3 {1 2 4 8 9} Inc Min Dist Between cows.



#Eg2: x-axis[]

#Stalls = 9 0 1 2 3 4 5 6 7 8

Cows = 4 {2 6 11 14 19 25 30 39 43}



Search:

Target: Maximum, minimum distance between 2 cows

SearchSpace: # gt should be m distance

#Note: Search space, make sure ans is in space.

[l ... h] l = smallest value h = greatest value

Min distance among cows l = Min diff among adj elements, 1

Max distance among cows h = ar[N-1] - ar[0]; # last - first

Final distance among M cows should be in search space [l..h]

Discard:

#Eg2: n-cows()

#Stalls = 9 0 1 2 3 4 5 6 7 8

#Cows = 4 {2 6 11 14 19 25 30 39 43}

Dist = 21 * C_1 C_2 C_3 C_4 *

Dist = 10 ✓ C_1 C_2 C_3 C_4

l h

#Search space: {1, 41}

l h m : ----- T T T T T F F F F F F F .. F

1 41 21 : Check if min dist among cows is atleast >= 21.

If we cannot keep them at 21 dist apart.

21 22 23 . . .

We cannot keep them at 22 23 .. apart as well

#goto left ; h = m-1;

1 20 10 : Check if min dist among cows is atleast >= 10

If we can keep them at 10 dist apart

7 8 9 10,

We can keep them at 9, 8, 7 . . . apart as well

#update ans, l = m+1

Dry Run:

Eg2: n-axis[]

Stalls = 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Cows = 4 { 2 | 6 | 11 | 14 | 19 | 25 | 30 | 39 | 43 }

d = 13	C_1	C_2 C_3	# C = 3 < 4 *
d = 12	C_1 12 C_2 16 C_3 13 C_4		# C = 4 = 4 ✓
d = 15	C_1	C_2 C_3	# C = 3 < 4 *
d = 10	C_1	C_2 C_3 C_4	# C = 4 ✓
d = 21	C_1	C_2	# C = 2 < 4 *

l	h	m: (l+h)/2	Can we cows at m distance apart
1	41	21	Can we keep all cows at 21 dist apart * h = m-1;
1	20	10	Can we keep all cows at 10 dist apart ✓ ans = 10, l = m+1
11	20	15	Can we keep all cows at 15 dist apart * h = m-1;
11	14	12	Can we keep all cows at 12 dist apart ✓ ans = 12, l = m+1
13	14	13	Can we keep all cows at 13 dist apart * h = m-1
13	12	# Stop process & return ans = 12.	

```
int minDist(vector<int> &arr, int c){ #Search space size:
    int N = arr.size();                    [l..h] = h-l+1
    int l=1, h = arr[N-1]-arr[0], ans=0;   #Binary Search Iteration
    while(l<=h){                           = log2(h-l+1)
        int m = (l+h)/2;                   #Check function
        if(check(arr, c, m)){              = N
            ans = m;                       TC: O(N log2(h-l+1))
            l = m+1;                       SC: O(1)
        }
        else{
            h = m-1;
        }
    }

    return ans;
}

bool check(vector<int> &arr, int m, int c){ TC: O(N)
    int last_c = arr[0], cows = 1;
    for(int i=1; i<arr.size(); i++){
        if(arr[i]-last_c >= m){ # Can place cow at ith stall
            cows++;
            last_c = arr[i];
        }
    }

    if(cows<c){return false;}
    else{return true;}
}
```

For what type of problems we can try binary search?

Questions:

1. Finding some target
2. Finding largest/last/Big/Max
3. Finding smallest/1st/small/Min.

When can we apply BS?

1. Target, 2. Search space 3. When we can discard search space.

When we can discard search space?

#Search space

F F F F T T T T..

If search space can be written as above or below pattern using Idea in that search space we can discard.

T T T T F F F F...

or

If using some idea : Eg: peak

#Monotonicity:

Increasing or Decreasing

Assume T=1, F=0

F F F F T T T T.. = 0 0 0 0 0 1 1 1 1

T T T T F F F F... = 1 1 1 1 1 0 0 0

Note: If search space follows, above pattern in your checkfunction we can apply BS.

28 Given ar[N], calculate length of smallest subarray with sum >= k

Constraints

$1 <= N <= 10^6$

$1 <= ar[i] <= 10^9$

#Eg1:

	0	1	2	3	4	5	6	7	8
ar[10] = {	3	2	5	7	4	8	9	2	6 }

k = 18

#Eg2:

	0	1	2	3	4	5	6	7	8	9
ar[10] = {	4	5	7	3	6	9	8	3	2	4 }

k = 20

#Idea1:

#Idea3:

Search:

Target:

SearchSpace: #Search n of subarray

#Note: Search space, make sure ans is in space.

[l ... h] l = smallest value h = greatest value

smallest subarray length l = 1.

greatest subarray length h = N

length of subarray with sum >= k will be in [l..h]

#Ex2:

	0	1	2	3	4	5	6	7	8	9
ar[10] = {	4	5	7	3	6	9	8	3	2	4 }

k = 20

Search Space: { }

l	h	m
1	10	5 :

#Check if there exists a subarray of len = 5 with sum >= 20 ✓

If subarray of len = 5, with sum >= 20 exists.

Subarray of len >5 with sum >= 20 exists

[5] 6 7 8 ...

$a_1 + a_2 + a_3 + a_4 + a_5 >= k$

If we add + n

$a_1 + a_2 + a_3 + a_4 + a_5 + n >= k$

ans = 5; h = m-1;

l	h	m
1	4	2 :

#Check if there exists a subarray of len = 2 with sum >= 20 ✓

If subarray of len = 2 with sum >= 20 doesn't exist

Subarray of len < k with sum >= 20 doesn't exist

1 [2]

l = m+1;

#DryRun.

#Eg2:

	0	1	2	3	4	5	6	7	8	9
ar[10] = {	4	5	7	3	6	9	8	3	2	4 }

k = 20

#Check if there exists a subarray of len m with sum>=k

l h m

```
int smallest(vector<int> &arr, int k){

}
```

```
bool issub(vector<int> &arr, int m, int k){

}
```