1B) Given N array elements, All elements repeat twice except 3, repeated elements are adjacent to each other and unique

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
arr[] :	3	3	1	1	8	8	10	19	19	6	6	2	2	4

Sol: If mid lands here, → we go to right
If mid lands here → we go to left

Sol: 1 odd elements
TC: O(N)
SC: O(1)

Idea: All three occurrence are at even positions
Idea: All three occurrence are odd positions

→ Make sure mid lands on first occurrence.

if(arr[mid] == arr[mid+1]){ mid = mid+1 }

0 1 4 7 10 → lands on 4th element

if(mid % 2 == 0){
// I am in left side
// go to right side
}

if(mid % 2 == 1){
// I am in right side
// go to left side
}

for indep(int arr[], int n){
int lo = 0, hi = n-1;
while(lo < hi){
for mid = (lo+hi)/2
if(arr[mid] != arr[mid+1] && arr[mid] != arr[mid-1]){
return arr[mid]
}
if(arr[mid] == arr[mid-1]){
// mid is landing on 2nd occurrence
mid = mid-1;
}
if(mid % 2 == 0){
// we are on left side
lo = mid+1; lo = mid+2
else{
// we are on right side
hi = mid-1
}
}

if (n==1) return arr[0]
// if one or 0th index we get our ans
if (arr[0] != arr[1]) return arr[0]
// if one at n-1 index we get our ans
if (arr[n-1] != arr[n-2]) return arr[n-1]

Trace code for above example

lo	hi	mid
0	14	7 != 6 → we are left side → lo = mid+1
8	14	11

2A) Given an rotated sorted array by K times, search if given key is present. Note: All Elements are distinct

	0	1	2	3	4	5	6	7	8	9	10
arr[] :	-15	-9	-6	0	2	5	8	10	14	18	23

K = 6

	0	1	2	3	4	5
rev[] :	5	8	10	14	18	23

K = 6

sorted array 1 (6 elements) — sorted array 2 (5 elements)

q = 14 ✓
q = 25 ✗

q ≥ 25 — 1st array
q < 25 — 2nd array

Solution:
→ Iterate on rev[]:
TC: O(N) SC: O(1)

Observe: All elements in sorted 1 are >= rev[0]

Solution2: Apply BS on 2 sorted arrays
Sol1: lo = 0 hi = k-1
Sol2: lo = k hi = N-1
TC: (log N + log N)
SC: O(1)

Observe2: All elements in sorted2 are < rev[0]

Sol3: q >= rev[0]
we do BS on 1st array
q < rev[0]
we do BS on 2nd array
TC: log(N) SC: O(1)

Note: K is not given search for given q:

	0	1	2	3	4	5	6	7	8	9	10
arr[] :	5	8	10	14	18	23	-15	-9	-6	0	2

Rotated parts — valid part

Idea:
Step find K:
Iterate & find K
TC: O(N)
OBS 1: K is index of min element

lo = 0, hi = n-1
while(lo < hi){
for mid = (lo+hi)/2
if(arr[mid] >= arr[0]){
lo = mid+1
}
else{
k = mid
hi = mid-1
}
return mid

Idea:
1st part:
→ goto right
→ l = mid+1

2nd part:
→ k = mid
→ goto left
→ hi = mid-1

// Condition to check
if(arr[mid] < arr[0]){
mid is on right side
}
else{
mid is on left side
}

2B) Given N find sqrt(N)

N = 16 → 4
N = 20 → 5

for sqrt(int N){
for ans = 1;
for (i = 1; i*i <= N; i++){
if(i*i <= N){
ans = i
else break;
}
}

TC: O(N) *
SC: O(1)

→ Define your search space?
your left have your ans

lo	hi	mid
1	N	
1	50	25, 25*25 > 50 — 21 22 23 28 .. — goto left side, hi = m-1
1	24	12, 13, 14, 15 — hi = m-1
1	11	6, 6*6 <= 50, ans = 6 — 3 4 5 6 — goto right for better ans, l = mid+1
7	11	9 10 11, hi = mid-1
7	8	7, 7*7 <= 50, ans = 7 — goto right, lo = mid+1
8	8	8, 8*8 > 50 ✗ — left side
8	7	return ans] 7

for sqrt(long N){
long lo = 1, hi = 10^9, ans = N
while(lo <= hi){
long mid = (lo+hi)/2
if(mid*mid <= N){
ans = mid; // look for a better ans
lo = mid+1
}
else{
hi = mid-1
}
}
return ans;
}

TC: log(N)
Binary Search Iteration
Due → log(N)

Constraints
1 ≤ N ≤ 10^18
→ N = 10^18
lo = 1, hi = 10^9
mid*mid ≤ N
mid*mid ≤ 10^18 — cannot store the long int
→ avoid overflow
hi = 10^9
Limit* mid* mid ≤ mid
lo = 1, hi = 10^9
(mid)^2 ≤ N

2C) Aggressive Cows:
Given N cows & M stalls, all M stalls are on X-axis at different locations. Place all N cows into a long stall such that distance between any 2 cows is maximized. M > N

Ex:
Stalls: 5
Cows: 3

	1	2	4	8	9
	c_1	c_2	c_3		

min dist = 3 — ans = 3

Ex:
Stalls: 9
Cows: 4

	2	6	11	14	19	25	30	34	38
	c_1		c_2		c_3			c_4	

min = 3
min = 9
min = 12
ans = 12

BF:
Take all combinations & placing our cows
Stalls C cows → N
for each combination iterate & get ans
TC: Stalls C cows → N * C(Stalls C cows)

// ans = 9 — can we place all cows at ans 9
5 6 7 8 9
T T T T T

// ans = 15 — can we place all cows at ans 15
15 16 17 18 ...
F F F F

update ans
Better ans in right side
↑ → F
we go to left side

lo	hi (max - min)	mid
1	41	21 ✗ goto left side, hi = mid-1
1	20	10, ans = 10, goto right, lo = mid+1
11	20	15 ✗ hi = mid-1
11	14	12, ans = 12, lo = mid+1
13	14	13 ✗ hi = mid-1
13	12	ans = 12

Pseudocode:
int dist(int arr[], int s, int c){
int lo = 1, hi = dist[n-1] - dist[0], int ans = 0;
while (lo <= hi){
int mid = (lo+hi)/2
if(check (mid, dist, s, c)){
ans = mid
lo = mid+1
}
else{
hi = mid-1
}
}
return ans;
}

TC: [Binary Search Iteration] * [TC of check function]
lo = 1
hi = [max - min of dist array] → O(1)
Total ops in which we are applying [lo - hi] → log(hi - lo)/2 → [Total Binary Search Iterations]

bool check(int dist, int s, int c){
int cows_placed = 1
int last_placed = dist[0]
for (int i = 1; i < s; i++){
// when will you place a new cow at dist[i]
if (dist[i] - last_placed >= mid){
cows_placed++;
last_placed = dist[i];
}
}
if (cows_placed < c)
return False
else
return True;
}

when dist are which 2 cows should be placed → No. of cows
check(int mid, int dist[], int s, int c)
int cows_placed = 1
int last_placed = dist[0]
for (int i = 1; i < s; i++)
// when will you place a new cow at dist[i]
if (dist[i] - last_placed >= mid){
cows_placed++;
last_placed = dist[i];
}
if (cows_placed < c)
return False
else
return True;

TC: log_2(hi - lo) * O(s)

2D) Given N tasks, K workers & time taken to complete each task. Find min time in which we can complete all tasks.

Note: A single worker can only do a continuous set of tasks.

Ex:
N = 15

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	3	5	1	7	8	2	5	3	10	1	4	7	5	4	6

K = 3 ?

c_1 → 24 — c_2 → 21 — c_3 → 26
c_1 → 24 — c_2 → 25 — c_3 → 22

minimize the max
min of all sum
37
→ 26
25

BF: ans = 30 // we can complete all tasks in 30

30 31 32 33 ....
T T T T

ans = 20 // can not complete all tasks in 20

... 17 18 19 20
F F F F

↑ if mid is invalid → goto right, lo = mid+1

if mid is valid
→ update & look for better ans in left
ans = mid
hi = mid-1

→ Define search space on these
lo: max of arr[]
hi: sum of all array elements

T T T T T .. T — F F F F F
F F F F F .. F — T T T T T
low true — high true

Note:

→ divide as & digits

while(lo <= hi){
int mid = (lo+hi)/2
if(mid*m > N){
high = mid-1
}
else{
ans = mid;
low = mid+1
}
}

→ [if(ans*ans - N ≤ 10^-6) return ans;]

Pseudocode:
int time(int task[], int N, int p){
int lo = max of task[], hi = sum of task[], ans = lo;
while(lo <= hi){
int mid = (lo+hi)/2
if (check (mid, task, N, p)){
ans = mid; // Better ans to left side
high = mid-1
}
else{
low = mid+1
}
}
return ans;
}

bool check(int time, int arr[], int n, int p){
int sum = 0;
int cut = 0;
for (int i = 0; i < n; i++){
sum = sum + arr[i];
if (sum > time){
// Time we should not consider
// The task
cut++; // for a person work is assigned
// Next person will start with task i
sum = arr[i]
}
}
if (sum > 0) { cut++ }
if (cut > p) {
return False
}
else{
return True;
}
}