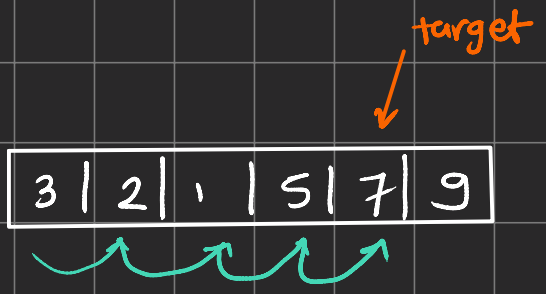
SEARCHING

* Linear search

TC $O(n)$



* Binary search

→ condition = element should be in monotonic order

(sorted → increasing
decreasing)

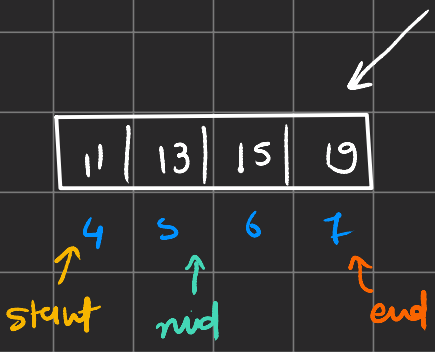
target = 15

→ start = 0th index

→ end = (n-1) th index

→ mid = $\left(\frac{\text{start} + \text{end}}{2}\right)$ index

→ mid : 9 value
and target = 15 so
9 ≠ 15 and
9 < 15
→ go to right side
while searching



→ here mid = 13 and 13 ≠ 15

and 13 < 15

→ neglect left side search on right

→ if 13 > target so move on left side

15	19
6	7

start → 6 (mid), 7 ← end

→ here mid = 15 and

15 = 15

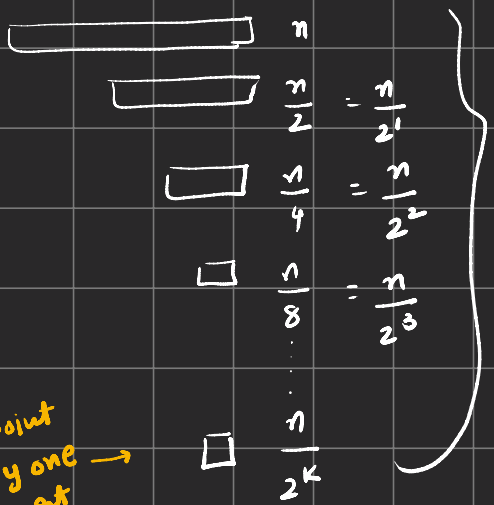
got index = 6

→ if value doesnot exist so return (-1)

(start <= end) condition

$mid = \frac{s+e}{2}$ } issue of INT overflowing sol → $mid = s + \frac{(e-s)}{2}$

→ Tc



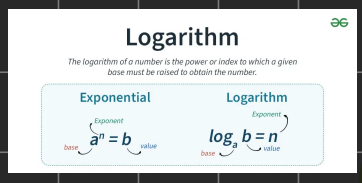
at this point there is only one element present in array

$\frac{n}{2^k}$ = 1 element

$n = 2^k$

$\log(n) = k$

$O(\log n) = O(k)$




```cpp
// #include <algorithm>

// on vector ------------------------
std::vector<int> v{1,2,3,4,5,6} ;

if (binary_search(v.begin(), v.end(), 3)) {
    cout << "3 is found in the vector." << endl;
} else {
    cout << "3 is not found in the vector." << endl;
}
```

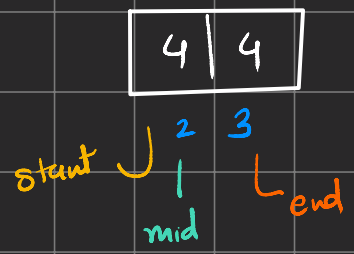
```cpp
// on array -------------------------
int arry[] = {1,2,3,4,5,6};
int sizeOfarry = 6;

// binary_search(intialAddress, lastElementAddress, target)
if (binary_search(arry, arry+sizeOfarry, 3)) {
    cout << "3 is found in the vector." << endl;
} else {
    cout << "3 is not found in the vector." << endl;
}
```

→ Find First occurance

target = 4

→ mid = 4 = target → ans store
→ search in left

→ here mid → 3 < 4 → search in right



→ here mid → 4 = 4
→ store ans
→ search left

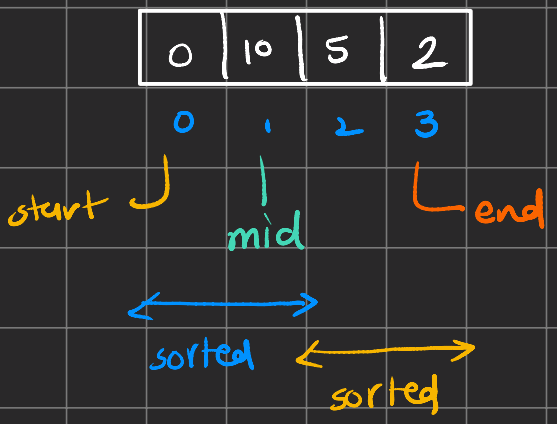
stop condition { start = 2 and end = 1
(s <= e)

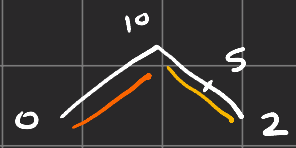
→ Find Last occurance

→ mid = 4 = target → ans store
→ search in right

→ Total number of occurance = lastocc − firstocc + 1

→ Find peak element






```
while (s<e)
{  if (mid < mid+1) ⟶ right search
   else ⟶ e = mid
   update mid.
return s
```

☆ Find Sqrt using Binary Search

$n = 25 \rightarrow \sqrt{25} = 5$

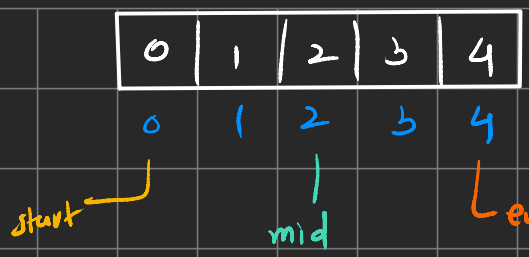
$n = 30 \rightarrow \sqrt{30} = 5.... = 5$

$n = 35 \rightarrow \sqrt{35} = 5.... = 5$

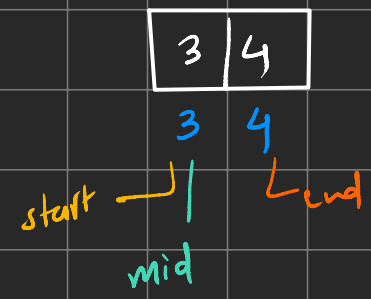
→ n = 10

$25 \neq 10$, $5 \times 5 = 25 > 10$

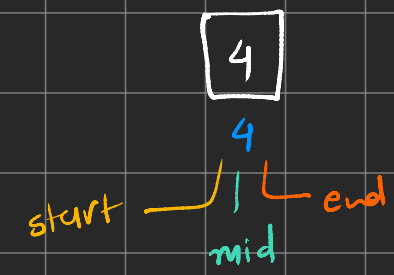
← left



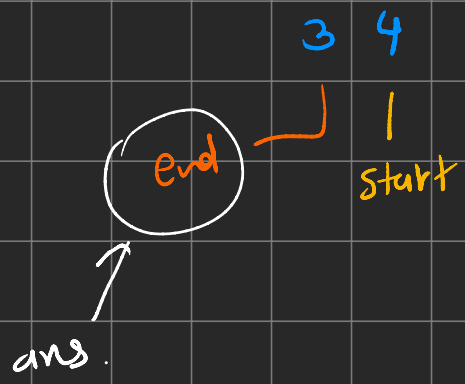
$2 \times 2 = 4 \neq 10$, $4 < 10$ → right



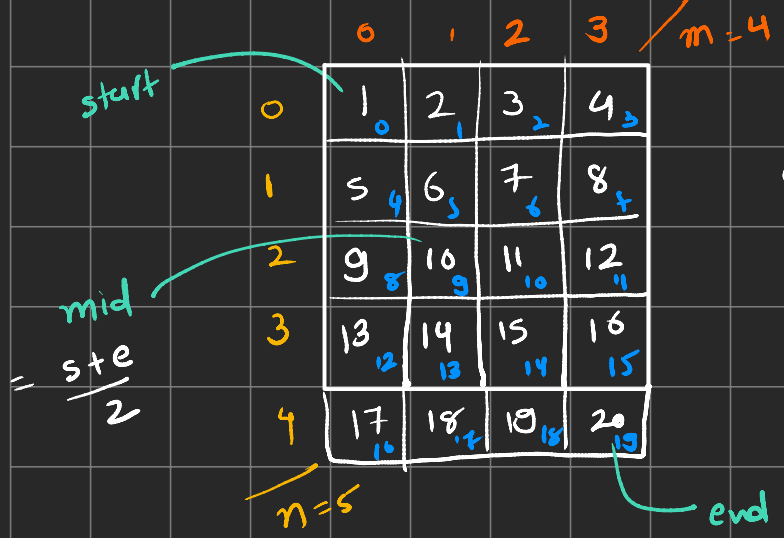
$3 \times 3 = 9 \neq 10$, $9 < 10$ → right



$4 \times 4 = 16 \neq 10$, $16 > 10$ ← left.



→ BS in 2D array



element = arr[rowIndex][colIndex]

mid / totalCol mid % totalCol

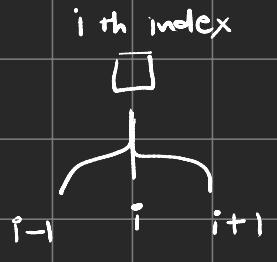
→ as we know underthehood, 2D stores as form of 1D Array so we can apply method that we are doing on 1D Array's BS.

BS in nearly sorted array

target =

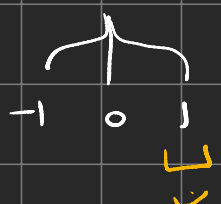
nearly sorted

arr: 10 | 3 | 40 | 20 | 50 | 80 | 70 (indices 0 1 2 3 4 5 6)



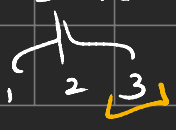
sorted arr: 3 | 10 | 20 | 40 | 50 | 70 | 80 (indices 0 1 2 3 4 5 6)

→ in sorted array, 3 → 0th index

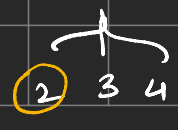


→ on these index 3 will be there in array

→ 20 → 2 index



→ 40 → 3 index



→ 70 → 5 index

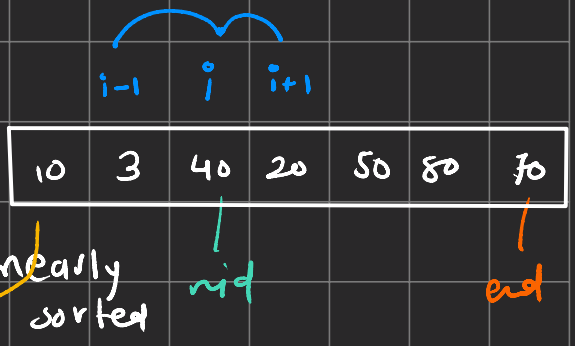
① linear search = n

② sort + BS

$n \log n + \log n = \log n(n+1)$, $(n+1) \approx n$

$= n \log n$

③



→ If (mid == target)

↳ return mid

→ elseif (mid < target)

↳ right search

s = mid + 1

→ else

↳ left search

e = mid - 1

→ IF (mid == target)

↳ return mid

ELSE IF (mid-1 == target)

↳ return (mid-1)

ELSEIF (mid+1 == target)

↳ return (mid+1)

→ elseif (mid < target)

↳ right search

s = mid + 2

→ else

↳ left search

e = mid - 2

O(log n)

★ Divide 2 numbers using BS

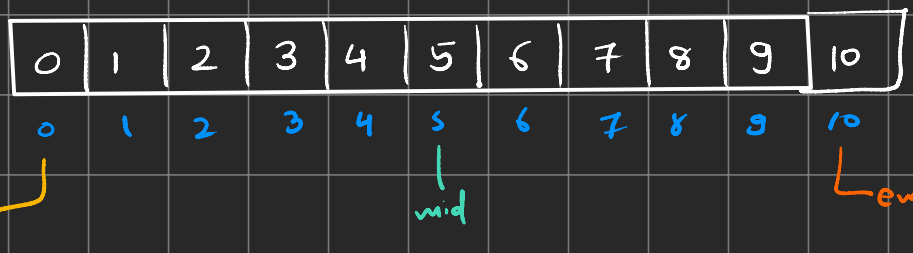
dividend = 10, divisor = 2, quotient = ?

```
    5
  ____
2 | 10
    10
   ---
    00
```

* Here quotient can be relay between 0 → dividend
* quotient * divisor + reminder = dividend
  (reminder ≥ 0)

quotient * divisor <= dividend

★ 0 → 10



→ 5 * 2 <= div

10 <= div

10 <= 10

True.

★ 22/7

7 * ans <= 22

→ 0 / s 11 / mid 22 / e

→ 7 * 11 <= 22

77 <= 22

77 < 22

← left search

→ 0 / s 5 / m 10 / e

→ 7 * 5 <= 22

35 <= 22

35 < 22

← left search

→ 0 (s) 2 (m) 4 (e)

→ 7 * 2 <= 22

14 < 22

→ right search

store this ans = 2

→ 3 (s) 4 (e)

s → m

→ 7 * 3 <= 22

21 <= 22

21 < 22

store ans

→ right search

→ 4 (s, e)

s → mid, e

→ 7 * 4 <= 22

28 < 22

← left search

→ e (mid) 4 (s)

e > s } break in this case

Here end = 3

ans = 3

mid = 3

} same.

→ For negative numbers → +ve/+ve | -ve/-ve | +ve/-ve | -ve/+ve

⊕ | ⊕ | ⊖ | ⊖

do all things with +ve and in end according to above just change ans's sign.

* Find odd occuring element in array

1	1	2	2	3	3	4	4	3	600	600	4	4
0	1	2	3	4	5	6	7	8	9	10	11	12

→ all elements occurs even no of times (2n), except one element

→ all repeating occurance of elemets exists in pair

and pair are not adjacent (આવા same જેવું પાસે પાસે નહીં પડે છે)

(there cannot be more than 2 consecutive occurance of any elemet)

→ Find the element that appeared odd no of times

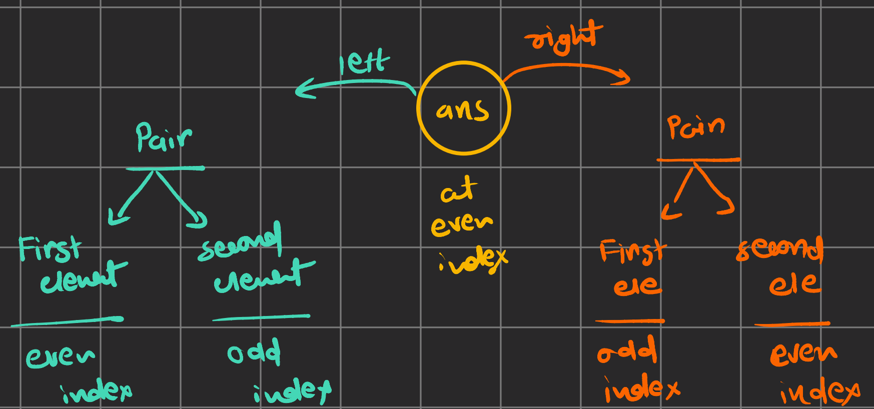
1 → 2, 2 → 2, 3 → 3, 4 → 4, 600 → 2

① XOR approach → 1^1^2^2^3^3^4^4^3^600^600^4^4 → 3

②

left ← → right

1	1	2	2	3	3	4	4	3	600	600	4	4
0	1	2	3	4	5	6	7	8	9	10	11	12
even	odd	even	odd	even	odd	even	odd	Ans even	odd	even	odd	even



→ s = 0, e = n-1, $mid = \frac{s+e}{2}$

mid = even index

- mid == mid+1 (mid: even, mid+1: odd) ⟶ left માં પડાડું તો ans right માં છે
  - → right search
  - s = mid + 2
- mid ≠ mid+1 (mid: even, mid+1: odd) ⟶ right માં પડાડું તો ans left માં હોવા
  - → left search
  - e = mid
  - this might be Answer ← that's the reason

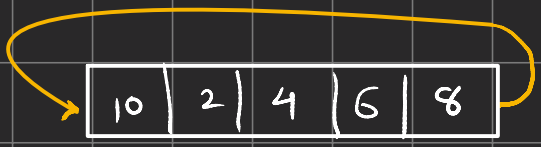
mid = odd index

- mid-1 == mid (mid-1: even, mid: odd) ⟶ left side માં પડાડું ans right માં હોવા
  - → right search
  - s = mid + 1
  - because we have not checked mid+1
- mid-1 ≠ mid (mid-1: even, mid: odd) ⟶ right side માં પડા ans left માં હોવા
  - → left search
  - e = mid - 1
  - this not be ans because ans's index always be even

Pivot Element

→ rotated and sorted array
→ no duplicate element

→ ascending (sorted)

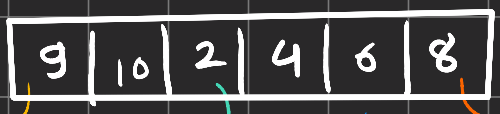


rotated

rotated

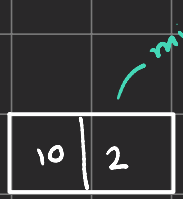
order break

pivoted ele. { max element



* 

→ IF (arr[mid] > arr[mid+1])
└ return mid

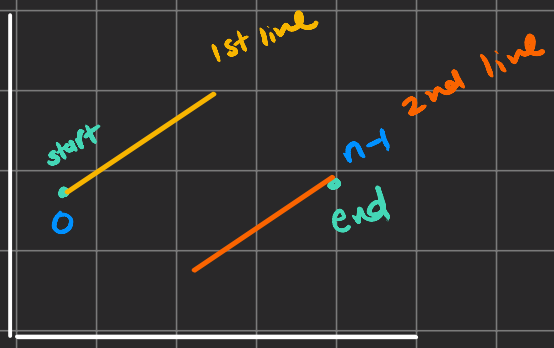
* 

→ IF (arr[mid-1] > arr[mid])
└→ return (mid-1)

* 

IF (arr[s] > arr[mid]
└→ so mid is in lineB
left search
e = mid - 1

OR



ELSE
↳ right search
s = mid + 1

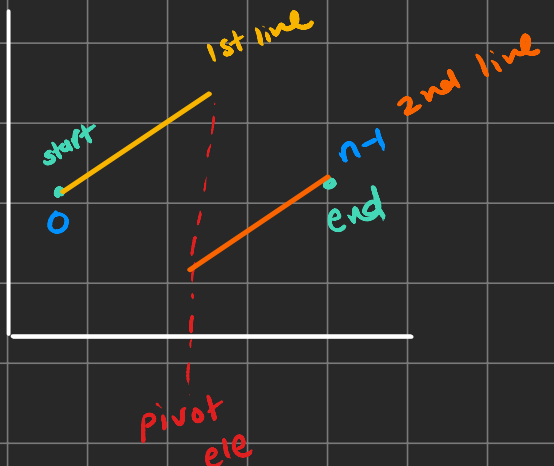
arr[mid] >= arr[0]

↳ shows first line
so move to right side

→ Search in rotated and sorted array target =

9	10	2	4	6	8
0	1	2	3	4	5

10 is pivot element



If (arr[pivot] ≤ target ≤ arr[n-1])

↳ use BS on second line

↳ s = pivot index
e = n-1
then do regular BS.

else
↳ use BS on First line

s = 0
e = pivot index

☆ K-diff pairs in array

difference

3	1	4	1	5
0	1	2	3	4

→ $0 <= i$, $j <$ nums.length

→ $i != j$

→ $abs(nums[i] - nums[j]) = K$

① Brute Force

② two pointer approch (sorted and then 2pointer app)

1	1	3	4	5
0	1	2	3	4
i	j			

① IF (i-j = target)
↳ i++ , j++

② i-j < target
↳ j++

③ else (i-j > target)
↳ i++

③ BS (sorted + BS)

K = 2

1	1	3	4	5
0	1	2	3	4
i				

→ a[i] = 1 , K + a[i] = 3 → now search on rightside that is there 3 available ?

2 + 1 = 3

(1,3)

→ a[i] = 1 , — — —

→ a[i] = 3 , 2+3 = 5 → search 5 on right side.

(3,5)

★ Find K closest Element

K = 4 → elements to return
x = 3

1	2	3	4	5
0	1	2	3	4

- $|a-x| < |b-x|$ or
- $|a-x| == |b-x|$ and $a < b$

→ K = 4, x = 35

12	16	22	30	35	39	42	45	48	50	53	55	56
0	1	2	3	4	5	6	7	8	9	10	11	12
23	19	13	5	0	4	7	10	13	15	18	20	21

diff = |x − a[i]|

22 < 48
when diff is same take small → big

① sort according to diff like

35	39	30	42	45	22	48
					13	13

k = 4 etc.

② two pointer

12	16	22	30	35	39	42	45	48	50	53	55	56
0	1	2	3	4	5	6	7	8	9	10	11	12
23	19	13	5	0	4	7	10	13	15	18	20	21

diff = |x − a[i]|

l → → →
← ← ← ← h

IF (diff at low > diff at high)
└ l++

ELSE
└ h--

N → K
O(n−k)

h − l ≥ K → break condition

7 − 5 ≥ 4
2 ≥ 4

③ BS + 2 pointer

12	16	22	30	35	39	42	45	48	50	53	55	56
0	1	2	3	4	5	6	7	8	9	10	11	12
23	19	13	5	0	4	7	10	13	15	18	20	21

① Find smallest ele. in array which is $\geq x$
- lower bound
- closest ele to x

② H → index of closest's element
L → H-1

③ [L, H] window form, size → K
↳ expand window to K

if (x - arr[low] > arr[high] - x)
↳ high++
else
↳ low--

12	16	22	30	35	39	42	45	48	50	53	55	56
0	1	2	3	4	5	6	7	8	9	10	11	12
23	19	13	5	0	4	7	10	13	15	18	20	21

L L H

→ diff low > diff high → high++
else → low++

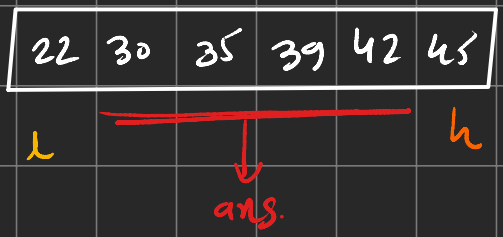
→ 5 > 0 → high++

→ 5 > 4 → high++

→ 5 < 7 → low++

→ 13 > 7 → high++

} 4 movements



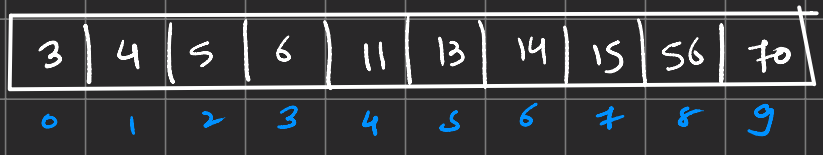
✱ Exponential Search

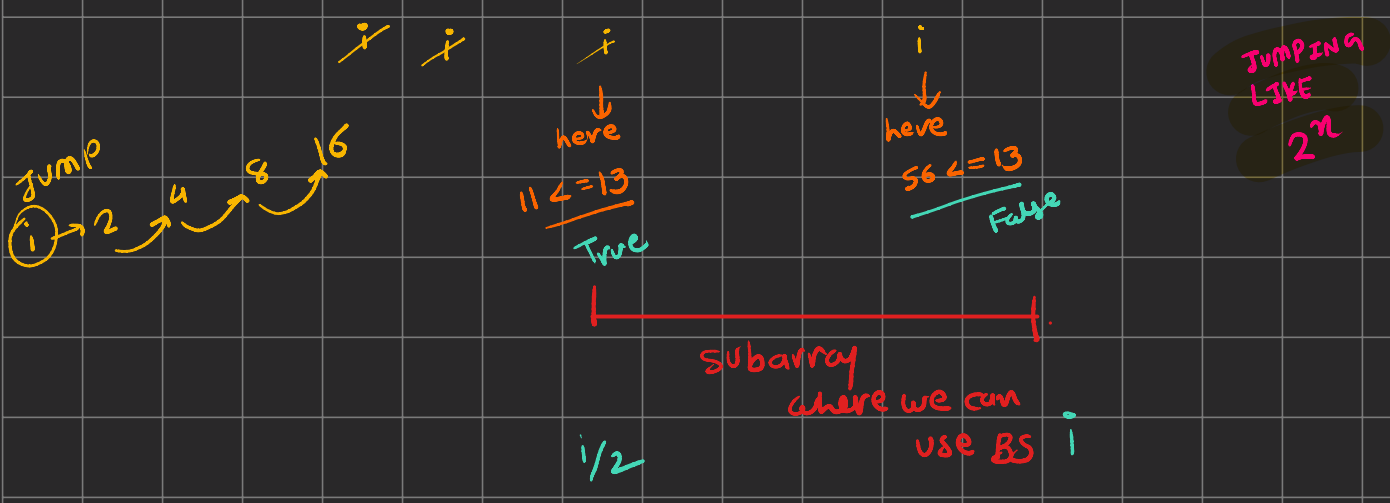
→ Array should be sorted

└ doubling / galloping / strazic search

→ Application : when u have a too long array so during that time just think like u have array from 0 → ∞ something so during that time TC becomes O(n) even if u are using BS to solve this and to reduce TC we are using this method

let array 0 → ∞ big size arr.





```
if (a[0] == x) → return 0;
int i = 1;
while (i<n && a[i] <= x)
{
    i = i * 2
}
return Bs(a, i/2, min(i, n-1), x)
```

TC: $\log(2^{\log m - 1})$

while loop { $\log 16 \rightarrow \log(i) \rightarrow \log_2 2^4 \rightarrow$ 4 steps } $O(\log m)$

Bs in subarray $\rightarrow 2^4 - 2^3 \rightarrow 16 - 8 = 8$

$\rightarrow 2^{\log m} \rightarrow 2^{\log_2 16} \rightarrow 2^{\log_2 2^4} \rightarrow 2^4$

$\rightarrow 2^{\log m} - 2^{\log m - 1}$

$\rightarrow 2^{\log m} - 2^{\log m} \cdot 2^{-1}$

$\rightarrow 2^{\log m}(1 - 2^{-1})$

$\rightarrow \frac{2^{\log m}}{2} \rightarrow 2^{\log m - 1}$

$m = 16$

$\rightarrow 2^{\log 16 - 1}$

$\rightarrow 2^{4-1}$

$\rightarrow 2^3$

$\rightarrow$ 8 steps

Application

→ search in ∞ array

unbounded array

→ Better than binary search when x is near beginning

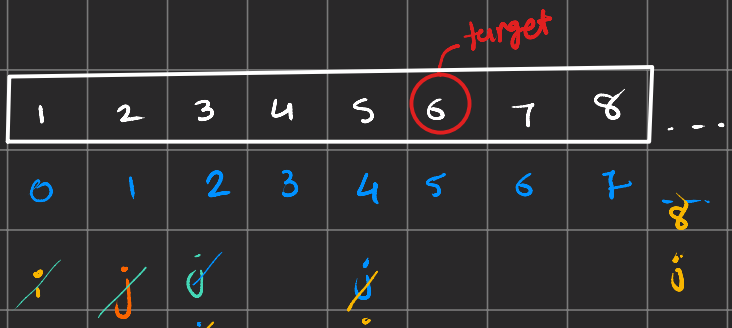
→ unbounded search

find the element in ∞ array sorted 1,2,3 ---- ∞

```
i=0
while (1) {
    if ( a[i] > x ) → break
    if (a[i] == x) → ans = i
    i++
}
```

Brute Force approach

→ Exponential search is optimized



that's in which we can apply Bs:

```
while ( a[j] < x)
{
    i = j
    j = j * 2
}
```

Book Allocation Problem

book = 4 m = 2

10, 20, 30, 40 → pages

Allocate books in such a way that:

1. Each student gets at least one book.
2. Each book should be allocated to only one student.
3. Book allocation should be in a contiguous manner.

You have to allocate the book to 'm' students such that the maximum number of pages assigned to a student is minimum.

10	20	30	40
0	1	2	3

max

go: 10 | 20 30 40 → 10, 90

70: 10 20 | 30 40 → 30, 70

60: 10 20 30 | 40 → 60, 40

find min from max (go, 70, 60) → ans : 60

(min)

① | 25 | 46 | 28 | 48 | 29 |

students = 4

low = max(arr) or 0

49

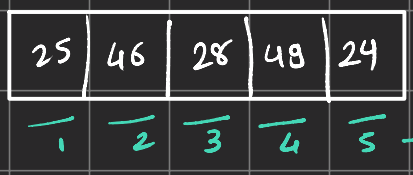
high = sum(arr)

171

```
for (pages = low → high)
{
    countStudent = funct (arr, pages)

    if (countStudent == m)
        return pages
}
```

```
fun (arr, Pages)        // mid (maximum Allowed Pages)
{
    stu = 1, pageStudat = 0

    for ( i=0 → n-1 )
    {
        if (pageStudat + arr[i] <= pages)

            pagesStudat = pagesStu + arr[i]

        else
            stu ++ ;
            pagesStu = arr[i]
    }
return stu
```

→ pages = 49



i=0 ⟶ 0+25 <=49 ⟶ ps = 0+25 = 49

i=1 ⟶ 49+46 ≤ 49 ✗ → stu = 1, ps = 46

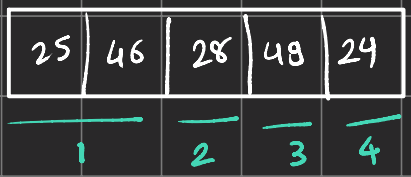
i=2 ⟶ 46+28 ≤ 49 ✗ → stu = 2, ps = 28

i=3 ⟶ 28+49 ≤ 49 ✗ → stu = 3, ps = 49

i=4 ⟶ 49+24 ≤ 49 ✗ → stu = 4, ps = 24

total students = 5

→ pags = 71



i=0 ⟶ 0+25 ≤ 71 ⟶ ps = 25

i=1 ⟶ 25+46 ≤ 71 ⟶ ps = 46

i=2 ⟶ 46+28 ≤ 71 ✗ → stu = 1, ps = 28

i=3 ⟶ 28+49 ≤ 71 ✗ → stu = 2, ps = 49

i=4 ⟶ 49+24 ≤ 71 ✗ → stu = 3, ps = 24

total stu = 4 done

TC → O(sum − max + 1) × n

WHY BS ? MAX-MIN જ્યાં એવા હોય ત્યાં apply કરી શકાય
and linear search TC બહાર આવતા હોય

(2)

s	m	e
49	110	172

↓

Func (arr, 110) → student = 2

we need more students

2 < 4

← move to left search

s	m	e
49	78	109

↳ 3 students

3 < 4

← again left search

s	m	e
49	63	77

↓

5 students 5 < 4 ✗

→ right search

→

s	m	e
64	68	77

↓

5 student

→ right student

↙

s	m	e
70	73	77

store 73

4 studet

try to reduce it because we need minimum

4<=4

left search

s	m	e
70	71	72

4 studet

4<=4

left search

s	m	e
70	70	70

5 studet

last store ans = 71

s	e
71	70

break here

* Painter's partition

Split array - largest sum

→ split the array into k subarrays such that the max subarray sum is minimum

↓

min (max)

ans: [10 20 30] [40]

10	20	30	40

k = 2 painters

max

	10	20 30 40		
90	① 10		② 90	
	10	20	30	40
30	① 30		② 70	
	10	20	30	40
60	60 ①			② 40

60 → min

Same as Book allocation

0 | m 10 | sum(arr) = 20
s | | e

I → 5 <= 10
5+5 <= 10
5+5+5 <= 10 X

II → 5 <= 10
5+5 <= 10

0	4	9
s	m	e

I → 5 <= 4 X

cannot do

↳ right search

$\frac{s}{s}$ $\frac{m}{7}$ $\frac{g}{e}$

I → $s < 7$, $s+5 < = 7$ ✗

II → $s < 7$

III → ✗ no solution

$\frac{8}{s}$ $\frac{g}{e}$

mid

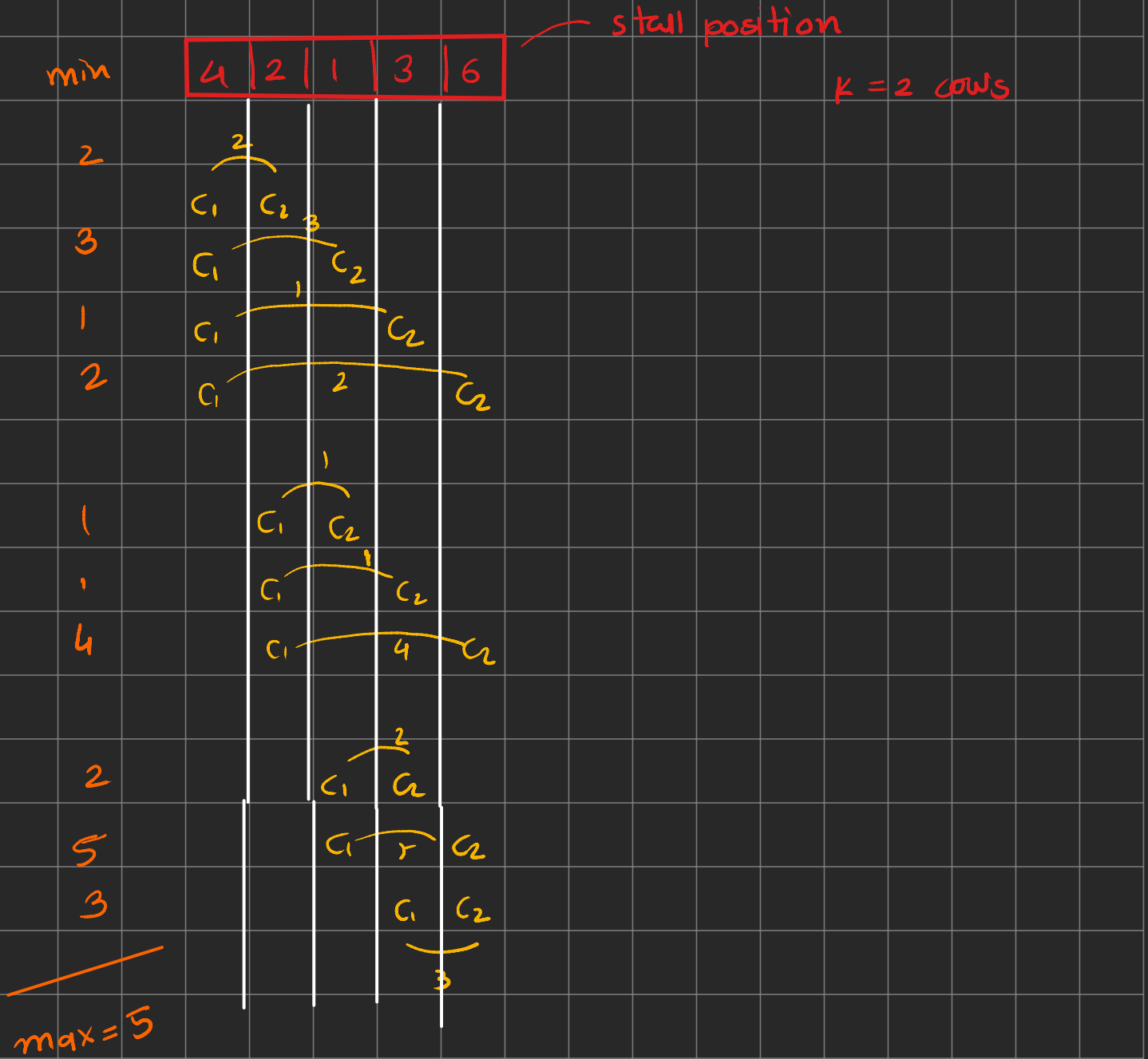
I → $s < 8$, $10 < = 8$ ✗

II →

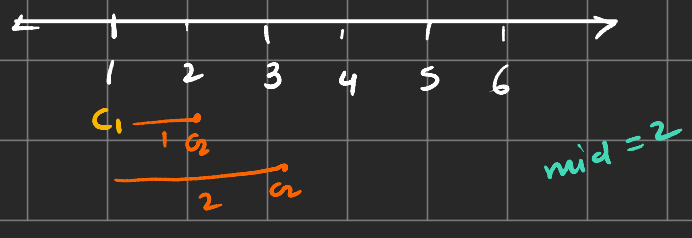
III → ✗ no sol.

$\frac{g}{se,m}$ } break.

✯ Aggressive cows



search space



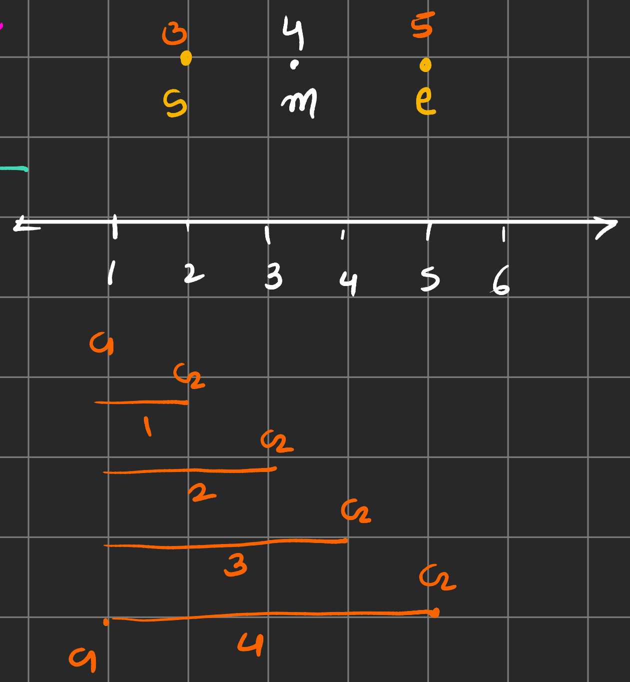
3-1 >= mid
>= 2

} અગર 2 ગાય 2 distance પર નહીં લડ
રહી તો ઉસકે આગે કે distance પર
ભી નહીં લડેગી

→ right search now

isPossible (stalls, mid, n) ← minimum allowed distance

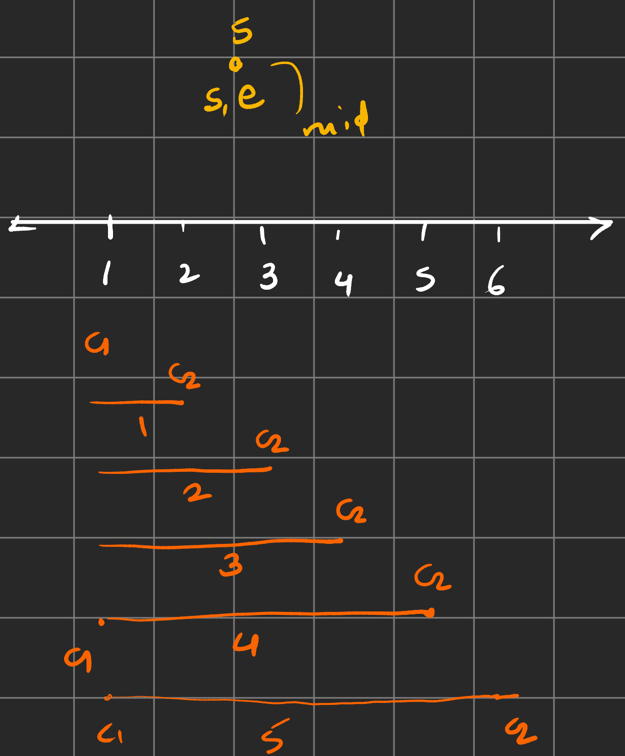
```
cowCount = 1
lastPos = stalls[0]

for (i = 0 → n)
  if (stalls[i] - lastPos >= mid)
  {
    cowCount++

    if (cowCount == k)
      return true

    lastPos = stall[i]
  }
return false
```



Sort the array that given in que.

1-5 >= mid
>= 4

→ right search



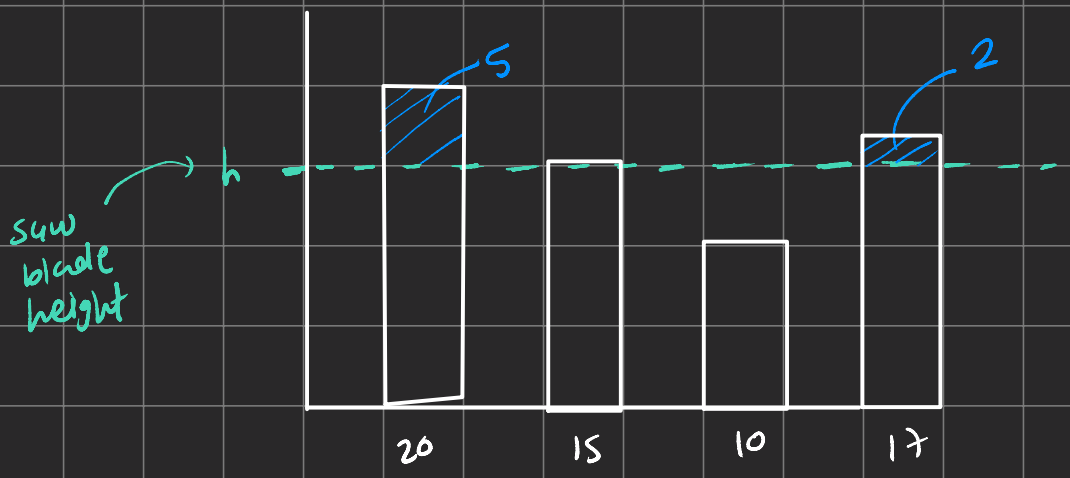
ans = 5

1-6 >= 5

✡ EKO SPOJ

M = 7 meter (min)

20	15	10	17



= 5 + 2
= 7

→ so when blade height is 15 meter we can get 7 meter of total wood

① BruteForce (i = 0 ⟶ i = max(arr) = 20)

i = 1 → 19 + 14 + 9 + 16 =

i = 2 → 18 + 13 + 8 + 15 =

⋮

i = 10 → 10 + 5 + 0 + 7 =

⋮

i = 15 → 7 meter → got ans

②

h = 10 → 10 + 5 + 0 + 7 = 22

22 > target

22 > 7 → move right search

stove ans

h=15 → 7 7 = target

7 == 7 → got ans

store ans

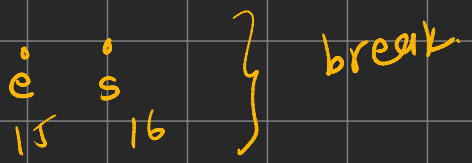
right search

h = 18 → 2 7 = target

(not) → left search

h=16 → 5 7= target

(not) → left search



✱ Parata - rotiparata spoj

n=5 cooks

total Parata = 10

min time to make order=?

R → 1 parata in R min

2R → 2 parata in 2 * R min

3R → 3 parata in 3 * R min

⋮

6 max

s m e 220 → total time taken by cook with highest rank when all parathas are given to him to make

110

→ C1 = 1 + 2*1 + ... 10*1

= 55 min

here → 4 rank

= 4 + 2*4 + 3*4 + ... + 10*4

= 4 * [1+2+3 ... +10]

= 4 * n x (n+1) / 2

= 220

is Possible to make → Yes

← left search

locz decrease time for cook

s m e

0 54 109

cooks can work parallel

→ C1 → 45 min for 9 paratha

→ C2 → 2 min for 1 paratha

10 paratha

is Possible to make → Yes

← left search

0 m 53

s 26 e

→ C1 = 1+2+3+4+5+6+7 → 6 parathas

21 min

28 >= mid >= 26

→ C_2 = 2 + 4 + 6 + 8 → 4 parathas

20 min

isPossibletomake → yes

← left search

0 m 25

s 12 e

→ C_1 = 1 + 2 + 3 + 4 + ~~5~~ → 4 paratha

10 min

15 min

15 >= mid

7 = 12

→ C_2 = 2 + 4 + 6 → 3 parata

12 min

12 >= mid

12 >= 12

→ C_3 = 3 + 6 → 2 parata

9 min

→ C_4 = 4 → 1 parata

isPossibletomake → yes

← left seared

0 m 24

s 5 e

→ C_1 = 1 + 2 = 2P

3 min

→ C_2 = 1 P

→ C_3 = 1 P

→ C_4 = 1 P

→ C_4 = 1 P

6 P <= 10 parata → not Possible

→ right search

6 m 24

s 8 e

→ C_1 = 3P

→ C_2 = 2P

→ C_3 = 1P

→ C_4 = 1P

→ C_4 = 1P

8P → not possible

9 m 24

s 10 e

→ C_1 = 4 P 1+2+3+4

→ C_2 = 2P

→ C_3 = 2P

→ C_4 = 1P

→ C_4 = 1P

9

Selection sort

→ different Rounds / passess

↳ smallest element lekake, usko right jagah par place karadete hai

0	1	2	3	4
64	25	12	22	11
i				

Use case
↳ small size arr

Round ①: 64 25 12 22 11 — smallest value — i=0 — swap

Round ②: 11 25 12 22 64 — smallest value — i=1 — swap

Round ③: 11 12 25 22 64 — smallest — i=2 — swap

Round ④: 11 12 22 25 64 — smallest — i=3 — swap

total Rounds = N-1 = 5-1 = 4

last element is already sorted

```
void selectionSort(vector<int>& arr, int n){

    for(int i=0; i<n-1; i++){
        int minIndex = i;

        for(int j=i+1; j<n-1; j++){
            if(arr[j]<arr[minIndex]){
                minIndex = j;
            }
        }

        swap(arr[minIndex], arr[i]);

    }

};
```

$n-1$

$(n-1), (n-2), \ldots 1$

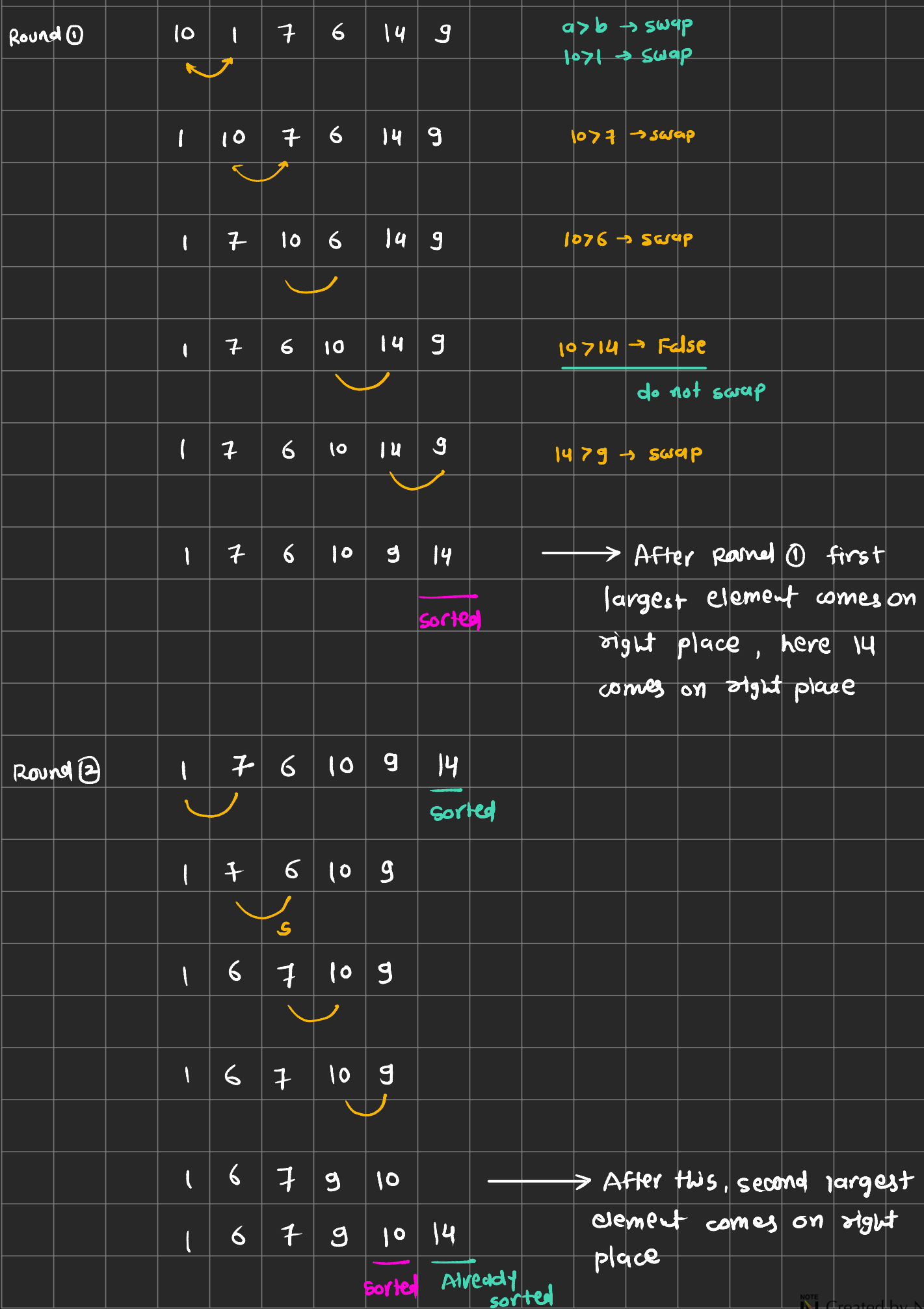
$= 1+2+3+\ldots+(n-1)+(n-2)$

$= \frac{n(n-1)}{2}$

$= \frac{n^2-n}{2} = O(n^2)$

Best case / worst case = $O(n^2)$

Bubble Sort



Round ③

1 6 7 9 10 14

sorted

1 6 7 9

1 6 7 9

1 6 7 9

1 6 7 9 10 14 → third largest ele from array on right place

sorted

Round ④

1 6 7 9 10 14

sorted

1 6 7

1 6 7 9 10 14

sorted

Round ⑤

1 6

1 6 7 9 10 14

sorted

1 6 7 9 10 14

in end automatic sorted bcz all others are sorted

```
void bubbleSort(vector<int>& arr, int n){

    for(int i=0; i<n; i++){


        bool swapped = false;

        for(int j=0; j<n-i; j++){
            // process ele till n-i element
            if(arr[j]>arr[j+1]){
                swap(arr[j],arr[j+1] );
                swapped = true;
            }
        }

        if(swapped==false){
            cout << "here swapped false hitted"<<endl;
            break;
        }

    }
};
```

use case :

i th round mei i th largest element ko uske right place par pahucha dete hai

Tc comparisons SC = O(1)

1 → (n-1)
2 → (n-2)
3 → (n-3)
4
⋮
(n-1)th → 1

= 1+2+3+ ... +(n-2)+(n-1)
= O(n²)

n is size of arr

→ optimised → in any round, when there is no swapping it means array is already sorted

due to ↓

Best case : O(n) , worst case : O(n²)

Insertion Sort → card example

arr[] = { 4 12 11 20 } → compare ele which is in Last

0	1	2	3	4	5	6
10	1	7	4	8	2	11

10	1	7	4	8	2	11

i

Round ① i=1

1 < 10
← left me add

so we need space of one index in left side

10 ને 1 space આગળ ખીસકાવ્યો
0th index પર 1 copy કરી દો

1	10	7	4	8	2	11

sorted i

Round ②

i=2

7 < 10 — left side go

7 > 1 → so right side of 1

10 ની આગળ મૂકાવાનો

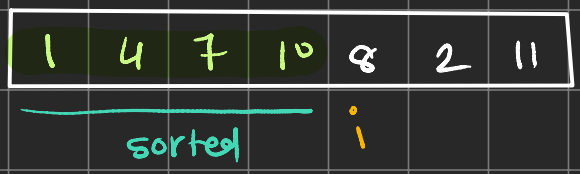
Round ③

4 < 10 → left side

4 < 7 → left

4 > 1 → right

7, 10 ની બધી space right માં મૂકાવાની

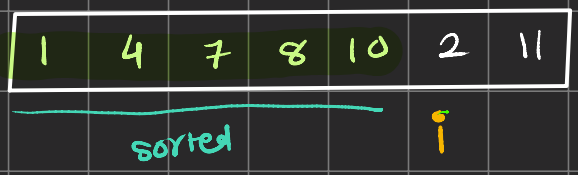


Round ④

8 < 10 → left side

8 > 7 → right side

10 ની 1 space shift થવી



Round ⑤

2 < 10 → left

2 < 8

2 < 7

2 < 4 → left

2 > 1 → right

4, 7, 8, 10 shift all to 1 space right side

Round ⑥ 11 > 10 → right side


```cpp
void insertionSort(vector<int>& arr, int n){

    for(int i=1; i<n; i++){

        int temp = arr[i];

        int j=i-1;
        for(; j>=0; j--){

            if(arr[j]>temp){
                // shift
                arr[j+1] = arr[j];
            }
            else{   // ruk jao
                break;
            }

        }
        arr[j+1] = temp;

    }
};
```

Use case

→ adaptable

→ stable

→ n small so good performance

→ partially sorted array then good

$TC = O(n^2)$

Best case $= O(n)$, worst $= O(n^2)$