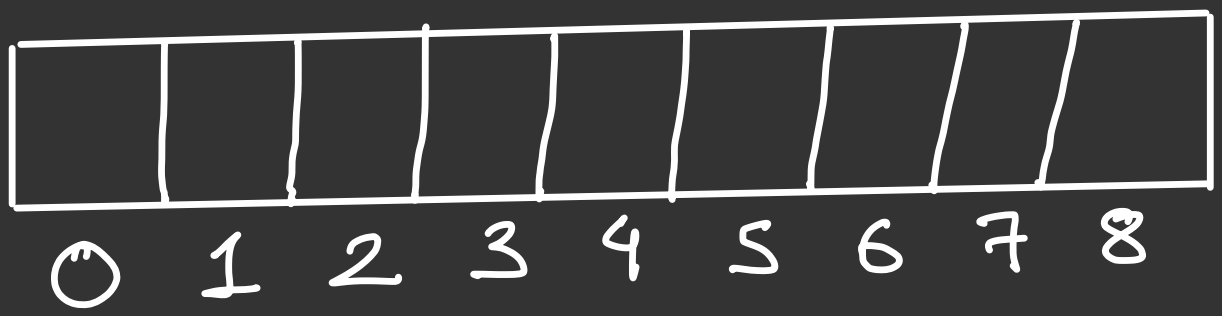
Lecture 9

Arrays

↳ can store values of same data type

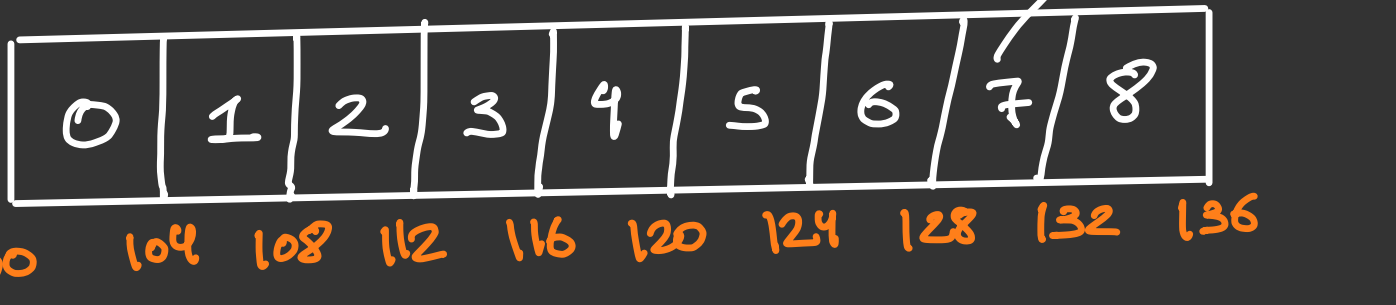
⟶ Contigous location of values in memory



✱ starts from 0 ⟶ index

Declaration

↳ int arr[10];



↳ int = 4 bytes

★ array size = n

index ⟶ 0 to (n-1)

eg:

int arr[3] = {0, 5, 3}

(or)

int arr[1000] = {0}

↳ all are initialised with value zero

★ only works with zero

★ to initialise other than zero

```
int arr[100];
// filling with value (1)
fill_n(arr, 100, 1);
```

int arr[10] = {1};

→ this will only initialise arr[0] = 1

→ and rest will initialised as 0.

Function for printing of array

```
void printArray (int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        cout << arr[i] << endl;
    }
    cout << "Printing done" << endl;
}
```

Size of Array

```
int arr[10];
int size = sizeof(arr) / sizeof(int)
```

40 / 4 = 10

4 bytes x 10

But if starting few elements are only stored, it will still tell the size of the whole array

```
char ch[5];
double d[10];
bool firstbool[5];
```

} Different datatypes

☆ Dont use variable as size while declaring the array

```
int size;
cin >> size;
int arr[size];
```
✗ wrong practice

☆ Integer range
→ -2^{31} to $2^{31}-1$

Formula: n = no. of bits

Range = $-2^{(n-1)}$ to $2^{(n-1)}-1$

☆ Can also use

```
maxi = max(arr[i], arr[i+1])
```
→ variable name cannot be max.

mini = min(arr[i], arr[i+1])

Scope

```
void update (int arr[], int size)
{
    arr[0] = 120;
}
int main()
{
    int arr[5] = {0,1,2,3,4};
    printArray(arr, 5);
    update(arr, 5);
    printArray(arr, 5);
}
```

0
1
2
3
4

120
1
2
3
4

SUPER IMPORTANT

int arr[3];

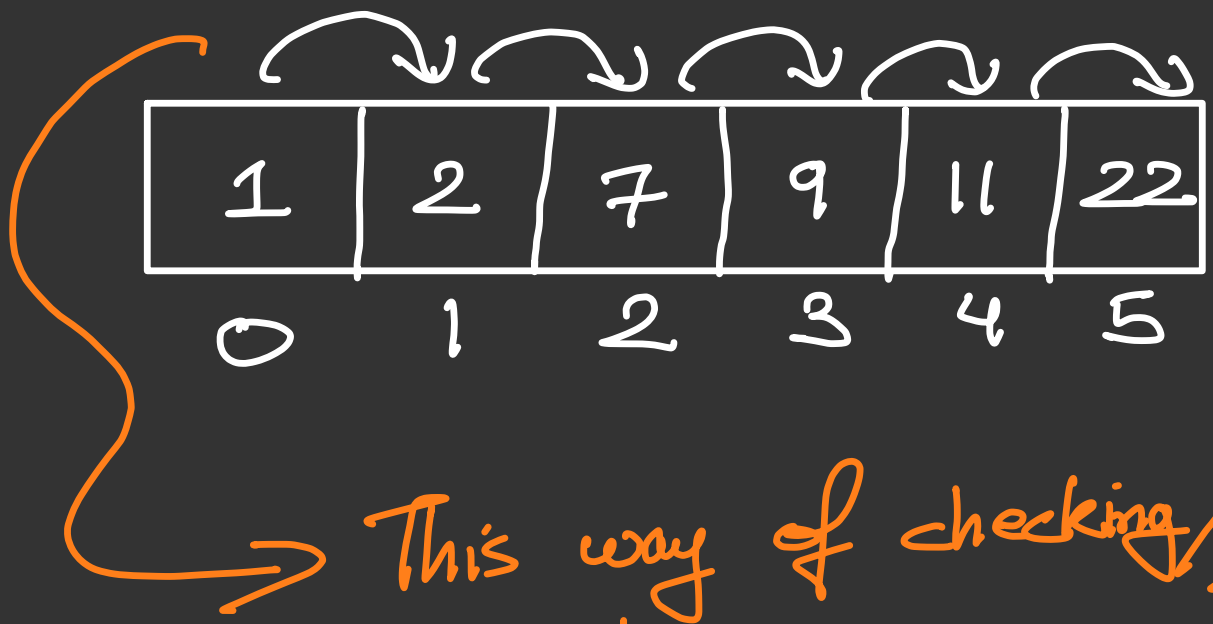
→ location

update(arr[], 5);

→ Similarly this receives the address not an actual array, thus once changed at the address, same is changed when accessed through main() unlike variables.

☆ copy variable is made in case of variable so it doesn't affect the actual variable in main(), as values are passed.

Linear Search



This way of checking/searching one by one

↳ Linear Search

→ linear_search.cpp

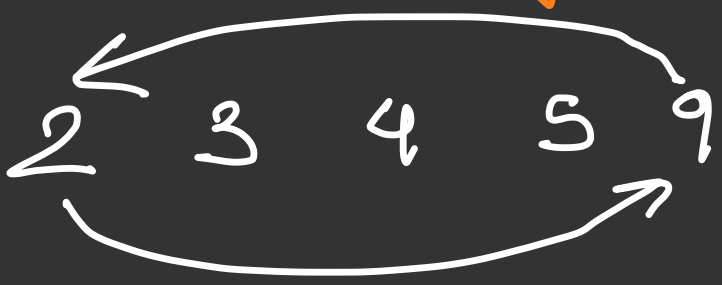
Homework Question

(find sum of the elements of the array)

↳ homework.cpp

Reverse an Integer

eg:



9 3 4 5 2

9 5 4 3 2

9 5 4 3 2

1	4	5	6	2
0	1	2	3	4

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

$n = 5$

1	4	5	6	2	1
0	1	2	3	4	5

$n = 6$

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

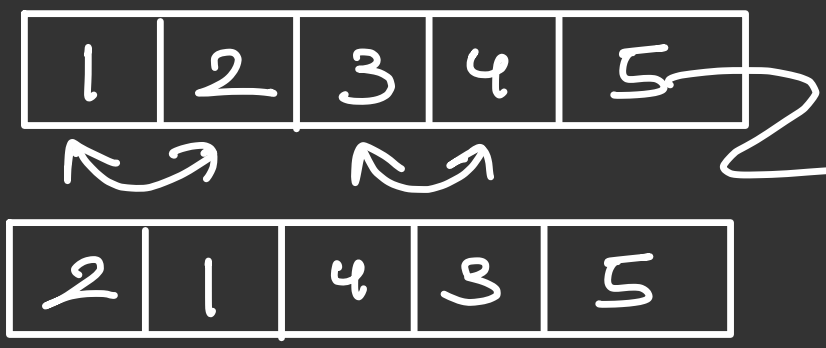
→ reverse_an_Array.cpp

```
swap(arr[i], arr[j]);
```

↳ In built function to swap values b/w variables.

Lecture 10

Alternate Swapping



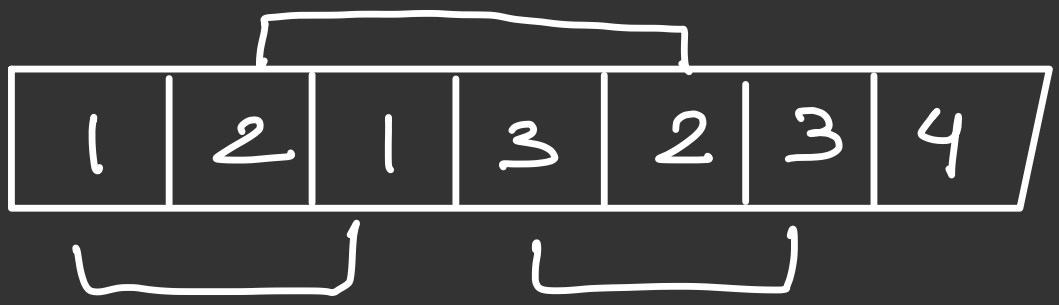
→ stay same if odd no. of elements

→ swap_alternate.cpp

Find Unique Element

Number of elements = $2m+1$

↳ odd



m elements appear twice

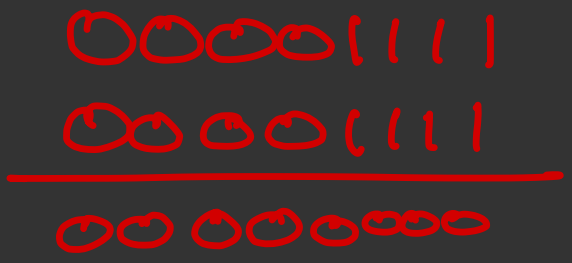
1 element appear once

↳ find this unique_element.cpp

* XOR → 1 only when odd no. of 1's

$a \wedge a = 0$

$0 \wedge a = a$



So,

$x \wedge 2 \wedge 3 \wedge 2 \wedge 3 \wedge x \wedge 4$

$\Rightarrow 0 \wedge 0 \wedge 0 \wedge 4 = 4$

Leetcode

Unique Number of occurances

#1207

→ unique_no_of_occurance.cpp

Duplicates in Array

→ array of size n

→ containing numbers between 1 to n-1.

but there is one integer present in the array twice.

n

1, 2, 3, 4, x - - , n-1, x

↳ we have to find x.

a^a = 0

0^a = a

→ we know n

So, 1^2^3^x__^n-1^x

1^2^3^x___^n-1

x^x^x = 0^x

= x

Leetcode #442

Find all duplicates in an Array

- array of length n
- all elements b/w 1 to n
- Each integer appear once or twice

{4, 3, 2, 7, 8, 2, 3, 1}

= {2, 3}

Approach

- first sort
- if i = i+1
  - push_back(i)

- find_all_duplicates.cpp

Intersection of Two Arrays

→ if nothing common return -1.

→ Arrays are sorted in non-decreasing order

{1, 2, 2, 2, 3, 4} → s1

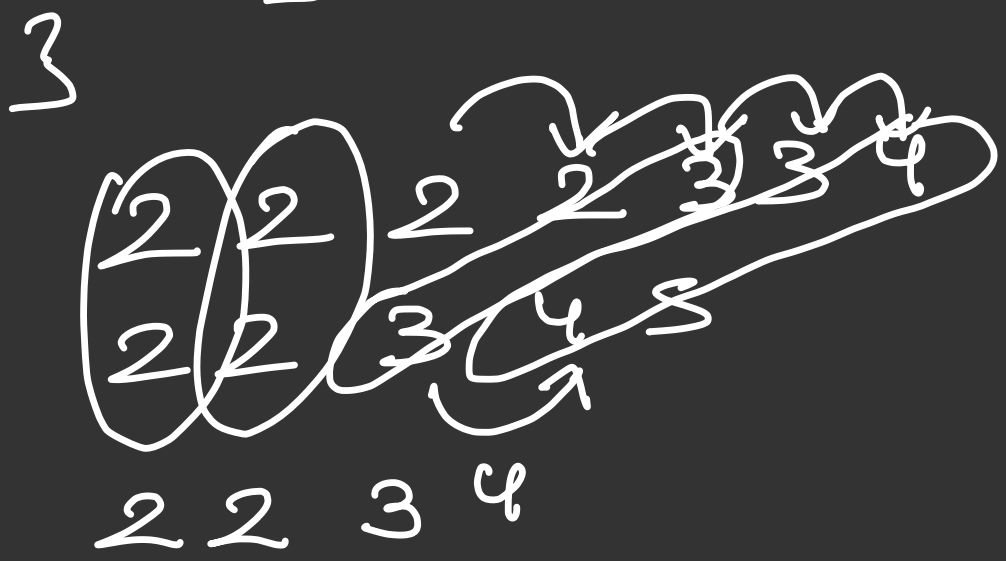
{2, 2, 3, 3} → s2

→ 2, 2, 3

```
int i=0; vector<int> ans;
int j=0;
for( ; i<s1 && j<s2; )
{
    if(arr1[i] == arr[j])
    {
        ans.push_back(arr[i]);
        i++;
        j++;
    }
```

→ Two Pointer Approach

```
else if (arr[i] > arr[j])
{
    j++;
}
else if (arr[i] < arr[j])
{
    i++;
}
```



→ array – intersection.cpp

TLE 2 → Time limit exceeded

Pair Sum

→ return the pairs of elements from the array that add up to the total.

→ in the pair, the first value has to be smallest one.

To store pair

→ `vector <vector<int>> ans;`

```
vector<int> temp;
temp.push_back(min(arr[i], arr[j]));
temp.push_back(max(arr[i], arr[j]));

ans.push_back(temp);
```

first sort before printing

↳ `sort(ans.begin(), ans.end());`

For Printing ⟶

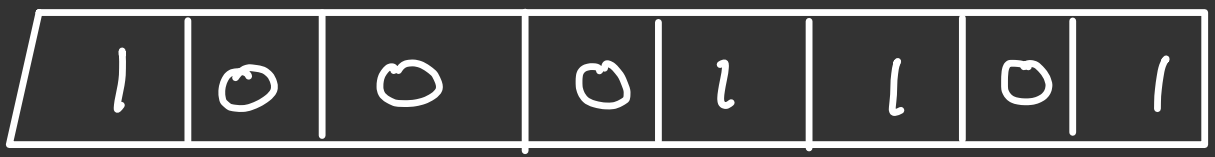
```
int L = ans.size();
for(int i=0; i<L; i++)
 {
   cout << ans[i][0] << ", "
        << ans[i][1] << " ";
 }
cout << endl;
```

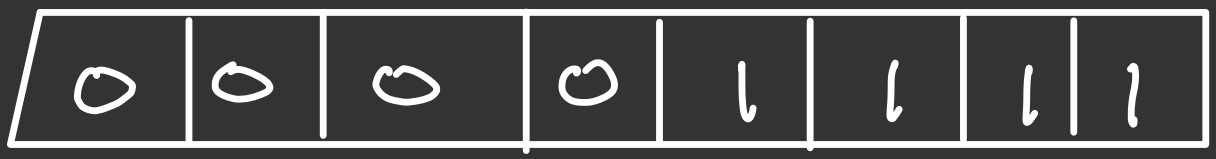
3 Sum

⟶ 3Sum.cpp

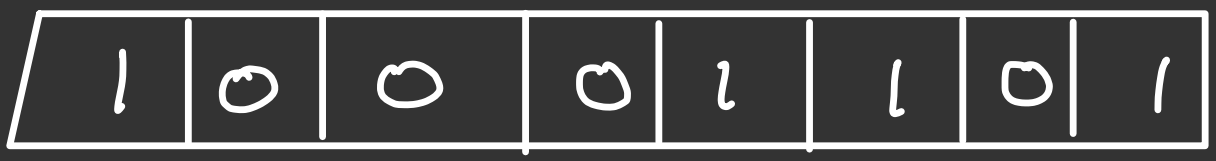
* Couldn't find how to remove duplications in vector.

Sort 01





⟶ first all the zero's followed by all the one's.



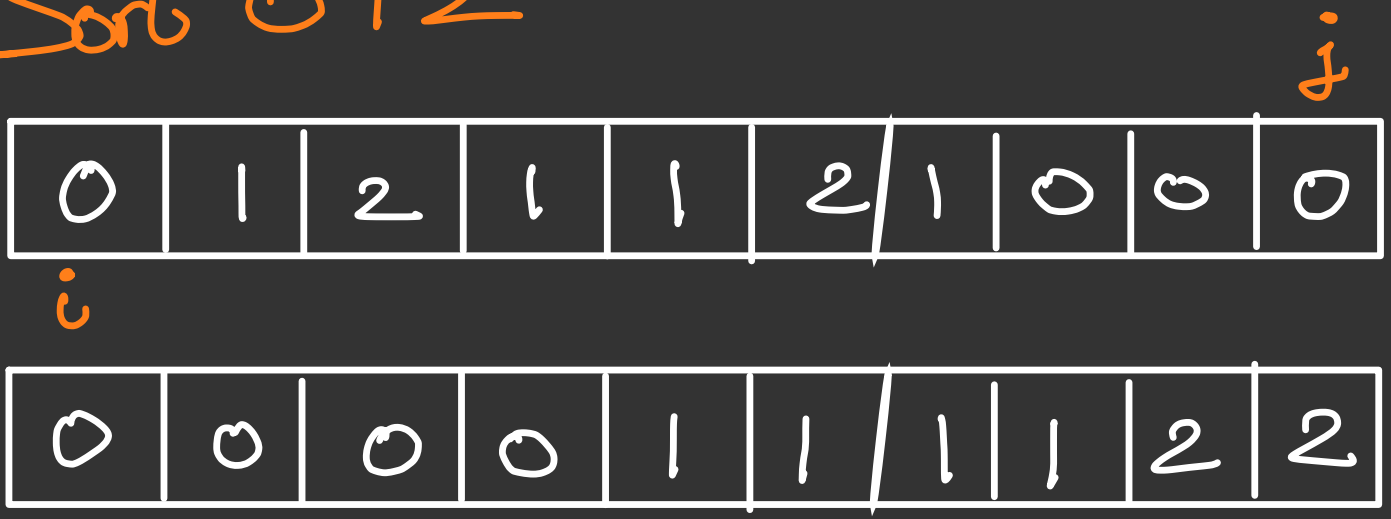

```
if (arr[i] == 0)
    i++
if (arr[j] == 1)
    j--;
```

```
if(arr[i]==1 && arr[j]==0)
{
    arr[i]=0;
    arr[j]=1;
    i++;
    j--;
}
```

→ While (i<j)

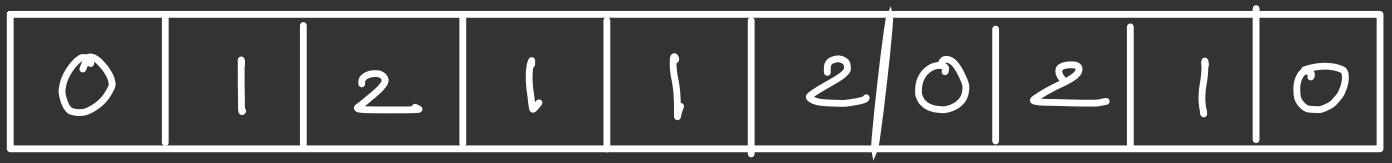
→ sort_0_1.cpp

Sort 0 1 2



k2 → will keep counts of number of ones.

→ sort_0_1_2.cpp




```
low = 0;
mid = 0;
high = n-1;

while (mid <= high)
{
    switch (arr[mid])
    {
        case 0:
            swap(arr[low], arr[mid]);
            low++;
            mid++;
            break;
        case 1:
            mid++;
            break;
```

Case 2:

```
swap(arr[high], arr[mid]);
high--;
break;
}
}
```

0	1	2	3	4	5	6	7	8	9
0	1	2	1	1	2	0	2	1	0

n = 10

low = ~~0 1 2~~ 3

mid = ~~0 1 2 3 4 5 6~~ 7

high = ~~9 8 7~~ 6

0	0	0	1	1	1	1	2	2	2

Lecture 11

Time Complexity

→ amounts of time taken by an algorithm to run, as a function of the length of the inputs.

Why?

↳ the machine on which the program runs could be also fast or slow, thus affecting the total performance of the algorithm.

So, time complexity gives a metric to compare the algorithms without having to run them.

Notations

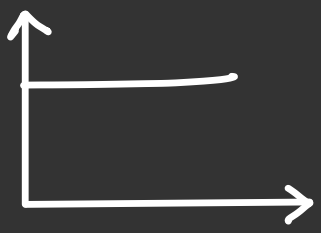
→ Big O notation (upper bound (or) worst case)

→ Omega Ω (lower bound)

→ Theta Θ (for avg case complexity)

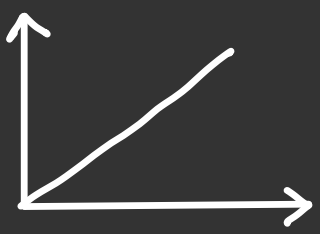
Big O notation

→ Constants time – O(1)



```
for(1->10)
  cout<<"hello";
```

→ Linear time – O(n)

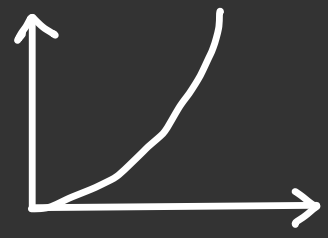


```
for(1->n)
  cout<<"hello";
```

→ Logarithmic time – O(log n)

Binary Search

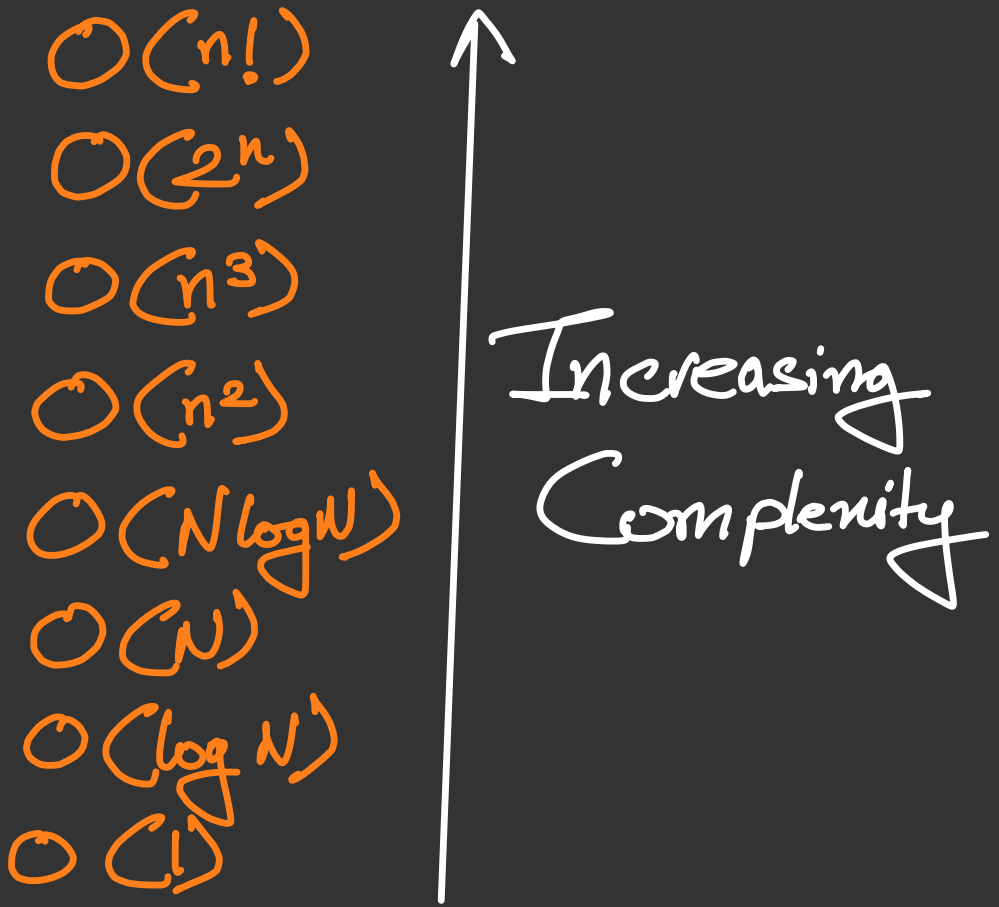
→ Quadratic Time – $O(n^2)$



```
for(1 → n)
{
    for(1 → n)
}
```

→ Cubic Time – $O(n^3)$

```
for(1 → n)
{
    for(1 → n)
    {
        for(1 → n)
    }
}
```



→ ignore lower degree

→ ignore constants

→ when another loop outside
↳ ADD

→ when another loop inside (or nested loop)
↳ MULTIPLY

Questions

$f(n) \longrightarrow 2n^2 + 3n$ $O(n^2)$

$f(n) \longrightarrow 4n^4 + 3n^3$ $O(n^4)$

$f(n) \longrightarrow N^2 + \log N$ $O(N^2)$

$f(n) \longrightarrow 12001$ $O(1)$

$f(n) \longrightarrow 3n^3 + 2n^2 + 5$ $O(n^3)$

$f(n) \longrightarrow \frac{n^3}{300}$ $O(n^3)$

$f(n) \longrightarrow n/4$ $O(n)$

$f(n) \longrightarrow \frac{n+4}{4}$ $O(n)$

} Linear time

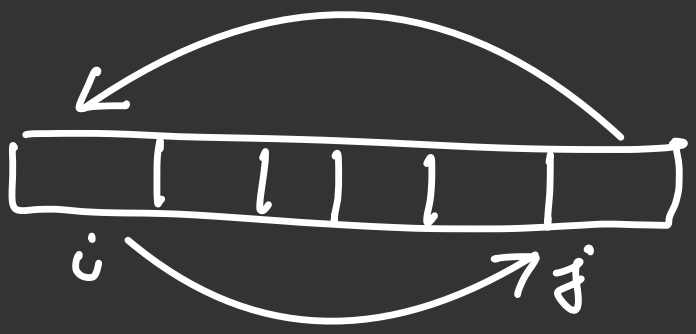
Examples

(1) Printing of array

```
for (0 → n)
{
    couts << "_"
}
```

$O(n)$

(2) Reversal of array



$n \rightarrow$ array length

$\frac{n}{2} \rightarrow$ swap

$O(n/2) = O(n)$

(3) Linear Search

```
for(1 → n)
```

$O(n)$

What is the time, space complexity of following code :

```
int a = 0, b = 0;
for (i = 0; i < N; i++) {
    a = a + rand();
}
for (j = 0; j < M; j++) {
    b = b + rand();
}
```

Assume that rand() is O(1) time, O(1) space function.

→ $O(N) + O(M)$

$= O(N+M)$

What is the time, space complexity of following code :

```
int a = 0, b = 0;
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        a = a + j;
    }
}
for (k = 0; k < N; k++) {
    b = b + k;
}
```

→ $O(N^2) + O(N)$

$= O(N^2+N)$

$= O(N^2)$

What is the time complexity of the following code :

```
int a = 0;
for (i = 0; i < N; i++) {
    for (j = N; j > i; j--) {
        a = a + i + j;
    }
}
```

for (i = 0 → N)
{
  for (j = N → i)
  {
    → N → 0
    (worst case)
  }
}

✱ Take the worst case

→ O(N) x O(N)

= O(N²)

Question

① isPrime

```
for (2 -> n)
{

}
```

→ O(n-2)

= O(n)

Stuck in TLE?

TLE - Time Limit Exceeded

10^8 operation rule

↳ most of the modern machine can perform 10^8 operation/second.

Constraints Chart

$1 < n < 10^6$

$< 10, 11$	$O(n!), O(n^6)$
$< 15, \ldots 18$	$O(2^n * n^2)$
< 100	$O(n^4)$
< 400	$O(n^3)$
< 2000	$O(n^2 * \log n)$
$< 10^4$	$O(n^2)$
$< 10^6$	$O(n \log n)$
$< 10^8$	$O(n), O(\log n)$

these will show TLE

will work

Space Complexity (memory)

What is the time, space complexity of following code :

```
int a = 0, b = 0;
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        a = a + j;
    }
}
for (k = 0; k < N; k++) {
    b = b + k;
}
```

⇒ O(1)

What is the time complexity of the following code :

```
int a = 0;
for (i = 0; i < N; i++) {
    for (j = N; j > i; j--) {
        a = a + i + j;
    }
}
```

⇒ O(1)

arr[5]; ⇒ O(1)

```
int n;
cin >> n;
vector <int> v(n);
```

↳ length = n

⇒ O(n)

```
for(0 → n)
{
    vector <int> v(n);
    for(0 → n)
    {
    }
}
```

Still,
space complexity
⇒ O(n)

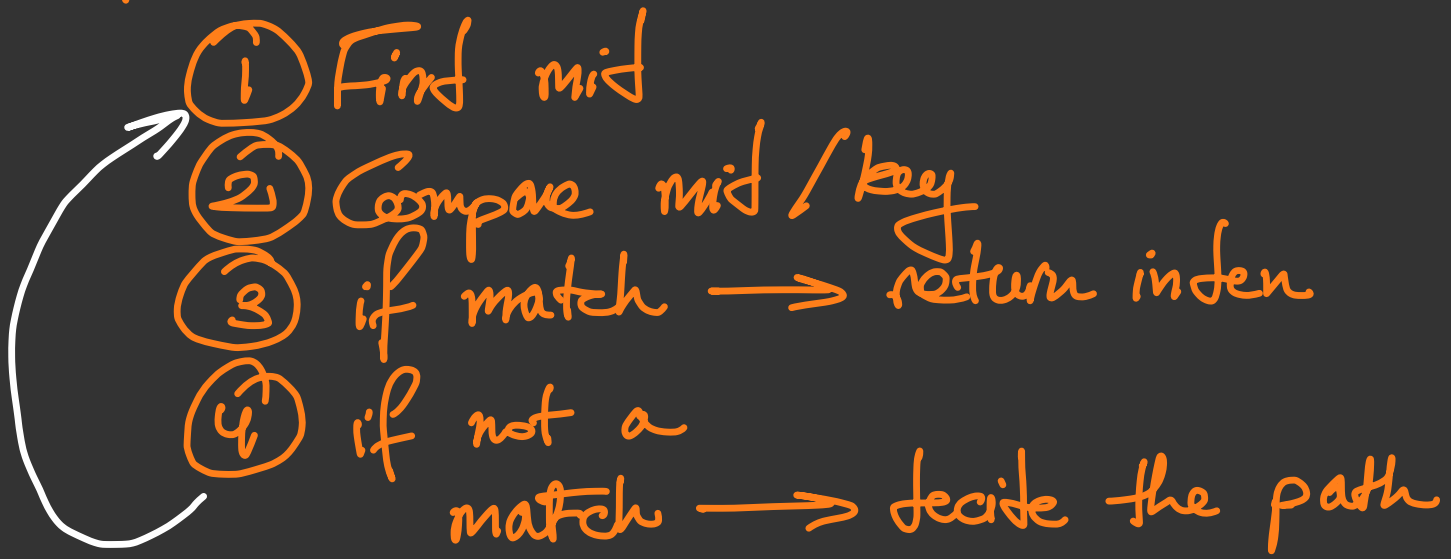
Lecture 12

Binary Search

→ work only on monotonic functions

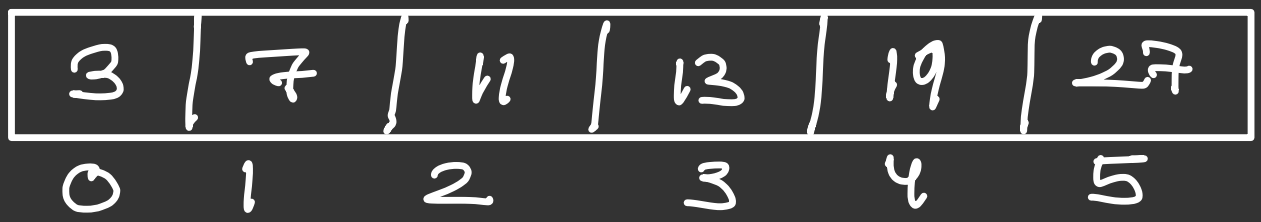
(either values should be in increasing order or decreasing)

Steps →



$$mid = \frac{\text{starting index} + \text{ending index}}{2}$$

example:



Key = 27

→ mid = $\frac{0+5}{2}$ = 2

→ 11 ! = 27

↳ 27 > 11

→ mid = $\frac{3+5}{2}$ = 4

→ 19 ! = 27

↳ 27 > 11

27

→ mid = $\frac{5+5}{2}$ = 5

→ 27 == 27 → return 5;

example:

5	11	13	17	19	27	30
0	1	2	3	4	5	6

Key = 25

→ $mid = \frac{0+6}{2} = 3$

→ 17 != 25

↳ 25 > 17

19	27	30

→ $mid = \frac{4+6}{2} = 5$

→ 27 != 25

↳ 25 < 27

19

→ $mid = \frac{4+4}{2} = 4$

→ 19 != 25 → NO LEFT PART

↳ return -1;

Integer max value = $2^{31}-1$

Just in case,

start = $2^{31}-1$

end = $2^{31}-1$

Then, mid = $\frac{\text{start} + \text{end}}{2}$

⟶ will give error

Thus,

$$mid = s + \frac{(e-s)}{2}$$

⟶ binary_search.cpp

☆ Comparison Linear Search vs Binary Search

→ Linear Search

1000 values

1000 comparison

↳ O(n)

→ Binary Search

1000 values

Compare 1000 = N

↓

500 = N/2

↓

250 = $N/2^2$

↓

125 = $N/2^3$

↓

$N/2^4 = 62$

$\downarrow$

31

$\downarrow$

15

$\downarrow$

7

$\downarrow$

$N/2^8 = 3$

$\downarrow$

$N/2^9 = 1$

Total 10 Comparisons

$\frac{N}{2^k} = 1 \rightarrow N = 2^k$

$k = \log N$

Thus, $O(\log n)$

* even if $k+1$ comparisons

$O(\log N + 1)$

$= O(\log N)$

Lecture 13

Binary Search Questions

Question 1:

first and last position of an element in a sorted array.

→ first_last_occurance.cpp

```
if(arr[mid] == key)
{
    ans = mid;
```

for first occurance:

```
end = mid - 1;
```

for last occurance:

```
start = mid + 1;
```

☆ Pair

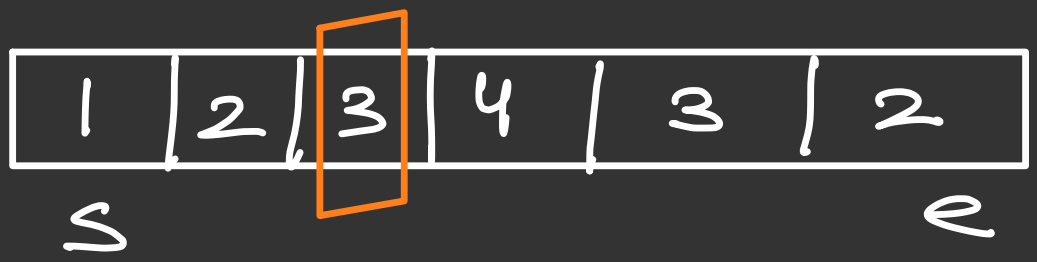
```
pair<int, int> p;
p.first = firstOccurance
                (arr, size, 3);

p.second = lastOccurance
                (arr, size, 3);
```

☆ If total number of occurance?

$$\Rightarrow [\text{last index} - \text{first index}] + 1$$

Peak Index in a Mountain Array

LeetCode – 852




```
m = ets/2;
if (arr[mid+1] > arr[mid])
{
    start = mid+1;
}
else if (arr[mid-1] > arr[mid])
{
    end = mid-1;
}
else
    return mid;
```

⟶ peak_index_in_mountain.cpp

Find Pivot Index

LeetCode - 724

1	7	3	6	5	6

→ first find sum of all the elements in the array in sum.

→ ans = 0

→ for (0 → n)

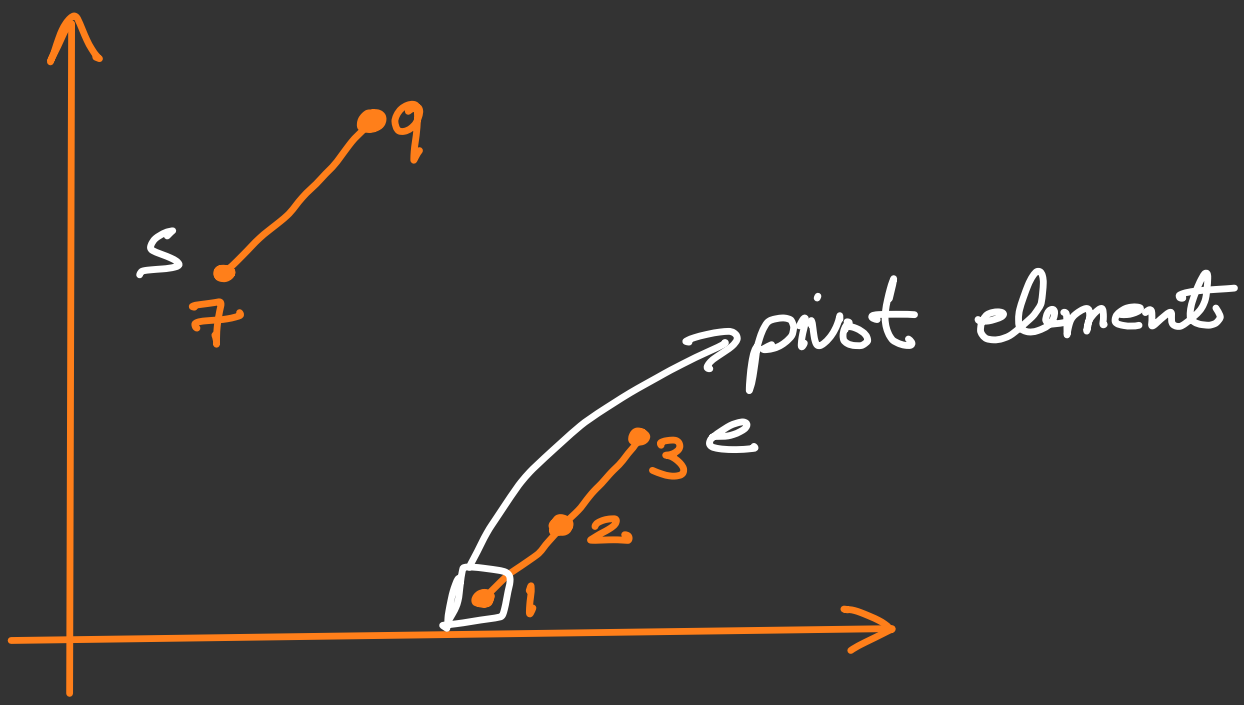
```
{
    sum = sum - arr[i];
    if (sum == ans)
        return i;
    ans = ans + arr[i];
}
```

Lecture 14

Pivot Elements

i/p ⟶ arr[] ⟶ {1,2,3,7,9} sorted

i/p ⟶ {7,9,1,2,3} rotated



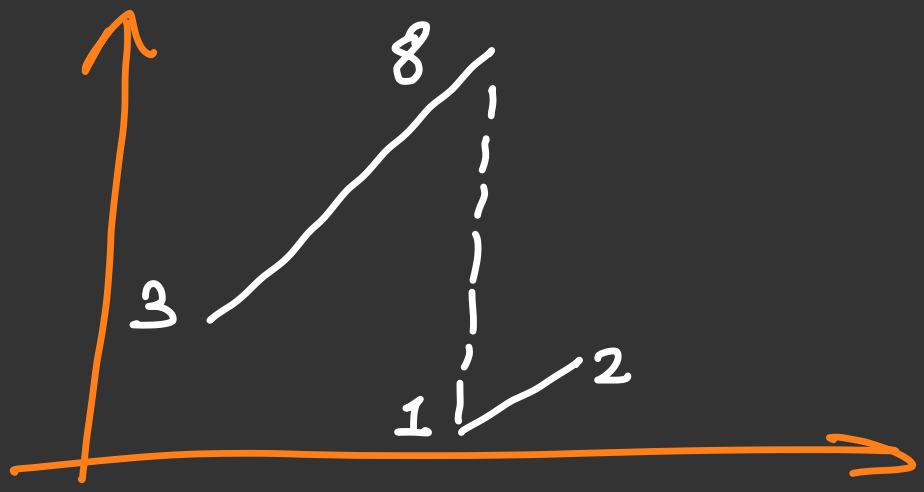
⟶ mid = s + (e-s)/2

⟶ if(arr[mid] >= arr[0])

s = mid + 1;

→ else if (arr[mid] < arr[0])
e = mid;
else if (arr[mid-1] > arr[mid])
return mid;

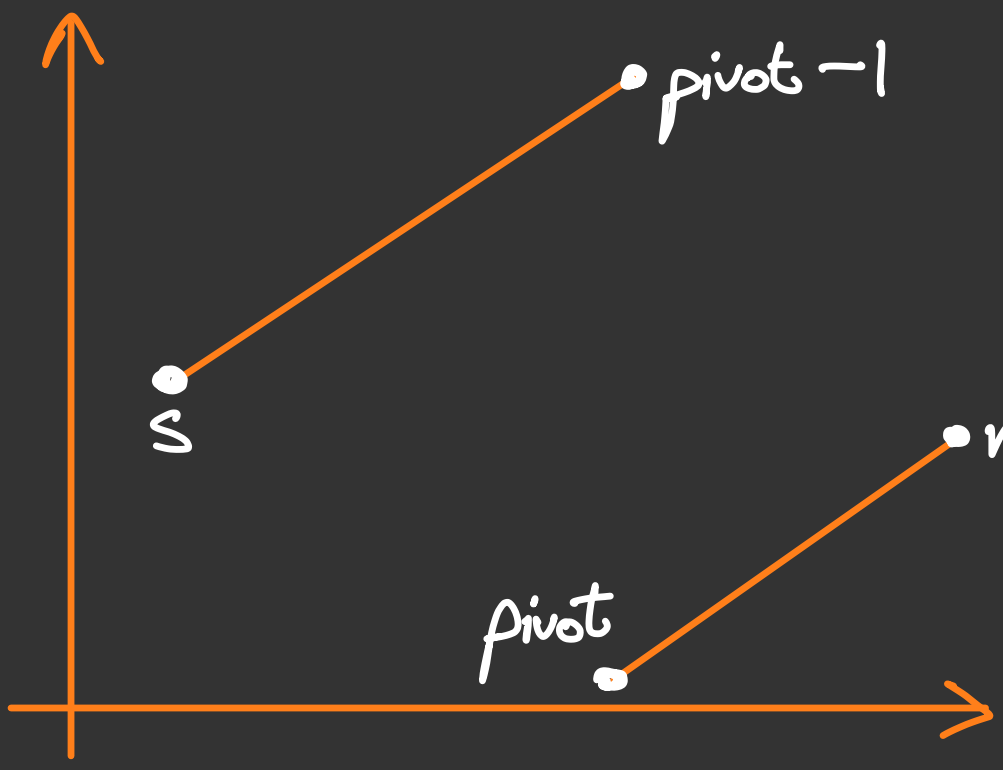
3	4	5	6	7	8	1	2
0	1	2	3	4	5	6	7



mid = ~~3~~ ~~5~~ 6
start = ~~0~~ ~~4~~ 6
end = ~~7~~ 6

→ pivot_index_in_array.cpp

Search in rotated, sorted Array



→ first find the pivot value

→ Check

if (key > arr[0])

→ Binary search in

I = s=0, e=pivot-1

else

→ Binary search in

II = s=pivot, e=n-1

→ search_rotated_sorted_array.cpp

Square Root of a number using binary search

eg: 98

s = 1 ~~7~~ 10

e = ~~97~~ ~~48~~ ~~23~~ 11

mid = ~~49~~ ~~24~~ ~~12~~ ~~6~~ 9

* Added a function for more precision ie. for decimal values.

Check for 0.1

↳ 6.1, 6.2, 6.3

then 0.01

⋮

till the precision counts

⟶ sqrt_binary.cpp

Lecture 15

① Book Allocation Problem

→ each element of the array represents the number of pages in the book.

→ 'm' students

→ each student should gets atleast one book

→ Book allocation should be in continous manner as given in the array.

Allocate such that the maximum number of pages assigned to a students is minimum.

eg:

10 | 20 30 40 = 10 (90)

10 20 | 30 40 = 30 (70)

10 20 30 | 40 = (60) 40

min of all these max = 60 ✓

☆ Not necessarily that a sorted array is given.

Example:

10	20	30	40

take min as 0. = s

max as sum of all

= 10 + 20 + 30 + 40

= 100 (no. of pages)

= e

Search space

While s<=e

1. 0 ——s——————————— 100

mid = $\frac{0+100}{2}$ = 50

isPossibleSolution ( ) → 2 → m=2

10 20 | 30 40

I → 10 + 20 + 30 ✗

II → 30 + 40 ✗

Thus, can divide into m equal parts when max is 50.
As we would need a third student.

If it couldn't satisfy for 50, then it wont satisfy for values less than 50.

What is happening?

⟶ We get the max allocation of number pages

⟶ We have m students
↓
m number of allocations

AND each allocation shouldn't be more than the max allocation number.

(i) When cannot allocate then

s = mid + 1;

mid = 75

10 20 30 | 40

I — 10 + 20 + 30 + (40) × } < 75

II — 40

⟶ store the solution

Now, we know all the values will be true for more than 75 also, BUT

We want the minimum of the maximum value

(ii) When successfully allocated,

e = mid − 1;

3. 51 ——— 74 (e)

$mid = \frac{51+74}{2} = 62$

10 20 30 | 40

I → 10 + 20 + 30 + (40) ✗

II → 40

→ Store this solution

4. 51 (s) ——— 61

$mid = \frac{51+61}{2} = 56$

10 20 | 30 40

I → 10 + 20 + (30) ✗

II → 30 + (40) ✗

→ cannot allocate

5. 57 ——— 61
s
mid = 59

⟶ cannot allocate

6. 60 ——— 61
mid = 60

10 20 30 | 40

I ⟶ 10 + 20 + 30 ≠ (40) ×

II ⟶ 40

Correct ✓

⟶ Store the allocation

⟶ e = mid − 1
= 60 − 1
= 59

s = 60

Condition $s <= e$

↳ false, out of while

```
isPossibleSolution(                    )
{
    c = 1;  // student counter
    sum = 0;  // sum of number of pages

    for(i = 0; i < size; i++)
    {
        if( sum + ar[i] <= mid)
            sum = sum + ar[i];
        else
        {
            c++;
            if(c > m || ar[i] > mid)
                return false;
            sum = ar[i];
        }
    }
    return true;
}
```

(2) Aggressive Cows

Array → position of stalls

Search space

→ min = 0

→ max = max_value − min_value

= 6 − 1 = 5

example:

{4, 2, 1, 3, 6}

→ {1, 2, 3, 4, 6}

k = 2

⚠ Where Binary Search ?

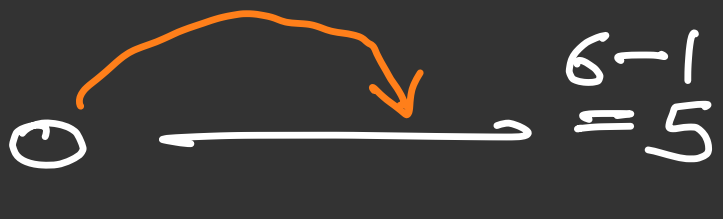
→ If after finding a possible solution, you can neglect some part of the search space

→ Binary Search.

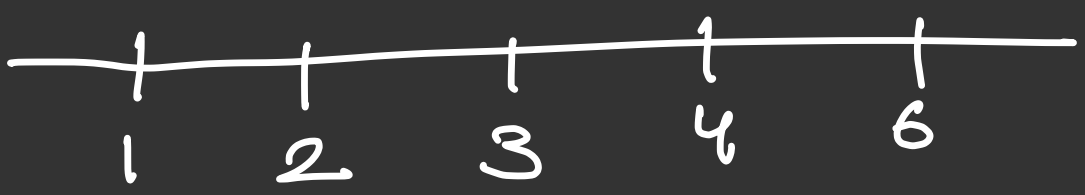
★ Search space contains all the possible values for the final answer, thus max − min.

example: {4, 2, 1, 3, 6}

1.

search space: 

mid = 5/2 = 2



c1 = 1

c2 = ~~2~~ 3 ✓

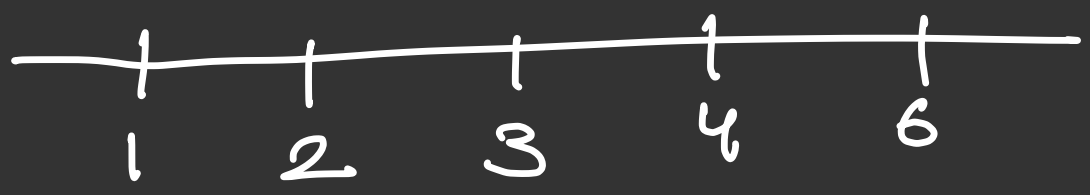
ans = mid;

When correct answer
↳ store
↳ s = mid + 1; (we want maximum possible distance)

2.

3 ——— 5

mid = 4



$c_1 = 1$

$c_2 = \cancel{2}\cancel{3}\cancel{4}\ 6$

ans = mid;

s = mid + 1;

↳ 4

s = e

3. 5 ——— 5

mid = 5

ans = 5;

s = mid + 1; → 6

s > e ✗

Thus, while condition (s<=e)

LeetCode

1011

Capacity to ship packages within D days.

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

day = 5

Let say if day=1 then all need to be shipped

Thus, search space → 1 —— sum

1 —— 55

mid = 28

won't even split for 5 days.

28 is possible but less partitions are made

possible ⟶ c < days

When not possible

↳ more weights left and no of partitions according to days are done i.e. c == days.

⟶ more partitions are made than requ'nd

not possible

↳ c > days

↳ c == days but elements left

⟶ ship_packages.cpp

PRACTICE

① Book Allocation

10 10 20 20 30 40

3 students

0 ——————— 130

mid = 65

10 10 20 20 | 30 | 40

possible, so store in ans but check for more

& for min to be maximum, mid should be less

0 ——— 64

mid = 32

>mid

10 10 | 20 | 20 30

33 ——— 64

mid = 48

>48

10 10 20 | 20 | 30 40

49 ——— 64

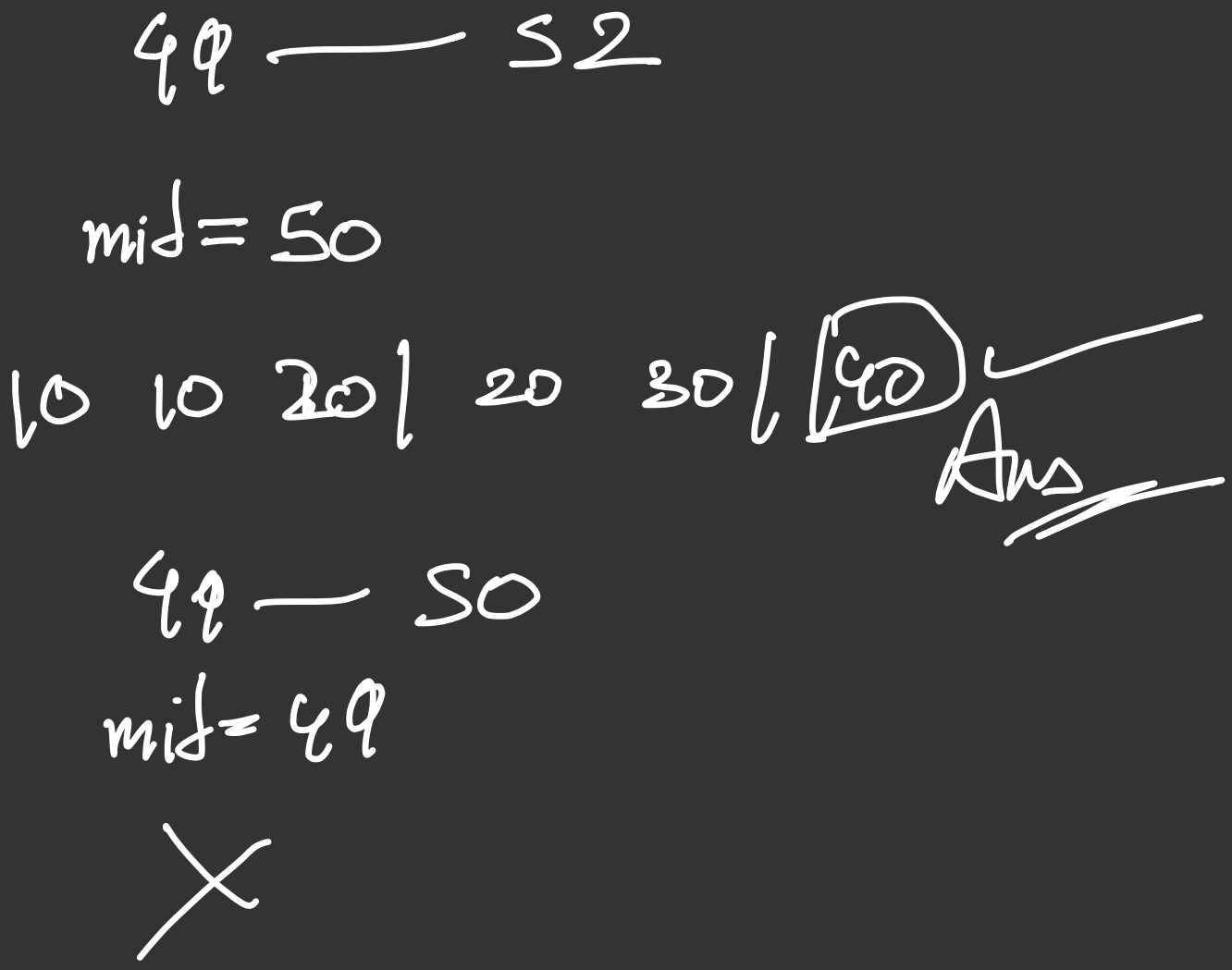
mid = 56

10 10 20 | 20 30 | 40

new max

49 — 55 mid = 52

10 10 20 | 20 30 | 40


49 — 52
mid = 50
10 10 20 | 20 30 | 40
Ans
49 — 50
mid = 49
X

Intuition for Binary Search

→ lines "man of the minimum"

→ When we have a set of range b/w which our answer will lie and by calculating for mid we can neglect either left or the right part.

Then find the range or search space

total = 100

10 | 20 30 40 max = 90

10 20 | 30 40 max = 70

10 20 30 | 40 max = 60

0 ——— 100

mid = 50

10 20 | 30 | X → s = mid + 1;

51 ——— 100

mid = 75

10 20 30 | 40 → store 75 as ans

and
e = mid − 1;

51 —— 74

mid = 62

10 20 30 | 40

51 —— 62

mid = 56

10 20 | 30 | x

57 —— 62

mid = 59

10 20 | 30 | x

60 —— 62

mid = 61 ✓ 60 — 60

mid = 60 ✓