**DSA -3**

**DAY 1- (Orientation and Introduction to BS)**

**Difference between Data Structures And Algo.**

* Basically data structures refers to storage and retrieval of data while algorithm refers to the set of instructions

**What topics we will be learning in this module.**

* Binary Search
* Stack
* Queue
* Hashing
* Binary Tree

**Points that will be very helpful in your coding career**

* Try to submit all the question in leetcode which you are submitting in acciojobs

Portal , this will help you to build your profile and you have a profile which you can add in your resume as well.

* Try to SPEAK LOUD WHILE READING THE QUESTION.

**BINARY SEARCH**

SEARCHING ALGO

IT IS A searching algo used in sorted array.

O(Log2N)= O(logN)

**ALGORITHM**

Divide the array into two halves

Find Mid

Key = Target Number

Compare the middle element with key

If the key is found at middle = return middle

if key is not at middle

if key is smaller than the middle

it will go to left part to search

if key is greater than the middle

it will go to right part to search

**To find mid why we are not using arr.length/2**

1. Binary search is a dynamic process, where the portion of the array being searched changes with each iteration
2. Changing search bounds = binary search divides the array in repeatedly half adjusting the search bounds low or high the middle index need to be calculated based on current bound
3. Using arr.length/2 would always give the middle index of the entire array

mid = low + (high-low)/2

**Pseudo Code Of**

**Q1. Binary Search**

****

**DRY RUN**

**arr = [ 2, 4, 7, 10, 14,18,21,25,30,34]**

low = 0

high = 9

mid = 0+(9-0)/2 = 4

Mid index is 4

arr[mid] = 14 target = 18

18> 14

new low = mid+1 = 4+1 = 5

mid = 5 + 9-5/2 = 7

arr[mid] = 25

18 < 25

l = 5

h = mid-1 = 7-1 = 6

mid = 5 + 6-5/2 = 5

mid = 5

arr[mid] = 18

target value = 18

**Q2. SEARCH INSERT POSITION**

**1, 3 , 4, 5 , 6 target value is 2 so index 1**

**target value is 6 so index is 2**

**APPROACH**

Low And High

Low will Be At Beginning

High will Be At End

Find Mid

If(arr[mid] >= x)

eliminate right

if(arr[mid] < x)

eliminate the left part

**When we have the target in array**

[1, 3, 5, 6]

x = 5

low = 0

high = 3

mid = 0 +(3-0)/2 = 1

arr[1] = 3

3<5 high

low = mid +1 = 2

mid = 2

arr[mid]

arr[2]= 5

ans = 2 is the answer

**When we don’t have the target in array**

low =0

high = 3

mid =1 ar[mid] = 3

target = 4

3<4

low = 2(mid +1) 1+1

mid = 2

arr[mid] = 5

4<5 left part = high =mid-1 = 2-1 = 1

low = 2

high = 1

return low = 2 which is the position where 4 would be inserted.

**TIME COMPLEXITY OF BINARY SEARCH**

The time complexity of an algo is determined by number of operations it performs relative to the size.

Dividing the array.

Imagine the array has n elements, after first comparison = n /2

then n/4

then n/8……….

Number of steps : The number of times you can halves the array before reducing it to a single element will be equal to log of base 2 of n

log2(n)

Best case = O(1) = The Target Found At Middle Only O(1)

Worst case = O(logn)

* Time complexity: O(logn) — Since binary search reduces the search space by half at each step, the maximum number of iterations required to find the target is log base 2 of n, where n is the size of the array. Therefore, the time complexity of binary search is O(logn).
* Space complexity: O(1) — Binary search only uses a constant amount of additional space for the two pointers and the middle index variable, so the space complexity is O(1).

**DAY -2**

**Q2. Find First and Last Occurence of the element**

N= 7 , target = 13

[3,4,13,13,45,56]

output = 2,3

**Brute Force Approach**

That is linear search in the array by updating the first variable, if we got our first variable, then update only second one till you reach the last occurrence.

Pseudo Code:

First = -1

second = -1

start traversing the array

when we first counter the target

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

if(arr[i] == k)

if(first == -1) first =i

last = i

Time complexity = O(N)

Space Complexity = O(1)

**OPTIMIZED APPROACH**

**Pseudo Code**

We will basically implement 2 binary search to find the first and last occurrence.

firstOccur()

First = -1 (ifd we don’t find any index will return -1)

Place low and high

Low = 0

High = last index

Find mid

Compare arr[mid] with k(target value)

Case 1 : if(arr[mid] == k)

With the help of this condition we can say mid may be our answer

So we will update first with mid and try to search in left part if there any smaller index available, as we have to find smallest index as first occurence

Case 2: if(arr[mid] <k)

Mid cannot be our answer , just eliminate left part

search in right side

Case 3: if(arr[mid] >k)

Mid cannot be our answer , just eliminate right part

search in left side

The process continues till low cross high.

secondOcc()

Last = -1

Place low and high

Low = 0

High = last index

Find mid

Compare arr[mid] with k(target value)

ase 1 : if(arr[mid] == k)

With the help of this condition we can say mid may be our answer

So we will update last with mid and try to search in right part if there any larger index available, as we have to find largest index as first occurence

Case 2: if(arr[mid] <k)

Mid cannot be our answer , just eliminate left part

search in right side

Case 3: if(arr[mid] >k)

Mid cannot be our answer , just eliminate right part

search in left side

The process continues till low cross high.

Dry Run

arr[1,2,2,2,3,4,5]

firstOcc target = 2

low = 0

high= 6

first = -1

mid = 3

arr[mid]= arr[3] = 2 which is k

first = 3

high = mid -1 = 3-1 = 2

check for smaller index in left part

mid = 1

arr[mid] = 2

set first = 1

high = mid -1 =1-1=0

mid = 0

arr[0] = 1 1<2

low =mid +1 = low = 1

high =0, low =1

low<high flow stop

Last Occ

low = 0

high = 6 last =-1

mid = 3

arr[3] = 2 = set last = 3

low = mid +1 = 4

mid = 5

arr[5] = 4 >2

high = mid-1 5-1= 4

mid = 4

arr[4] = 3 3>2

high = mid-1 = 3

last = 3

So ans is[1,3]

**TIME COMPLEXITY = O(2\*logN) where N is size of array**

**Space Complexity = O(1).**

**Q4. Find min in a rotated sorted array.**

ALGO for brute force approach

start by assuming first element to be min

min = arr[i]

loop through the array starting second index 1

for each index

compare min value with each index

int min = arr[0];

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

if(nums[i] < min){

min = nums[i]

return min

Time and space complexity will be O(N) and O(1)

**Optimised Approach using Binary Search**

**Algo**

left = 0

right = n-1

mid

arr[mid] > arr[right] the min is in the right half

left = mid+1

if (arr[mid] <= right)

the min should be in the left part (including mid)

right = mid

Loop will go till when the (left < right) otherwise if this condition not there than loop stop and return arr[left]

**Dry Run**

[3,4,5,1,2]

left =0

right = 4

mid = 2

array[mid] = array[2] = 5

array[right] = array[4] = 2 5>2 left = mid+1 = 3

left =3 right =4 mid = 3

array[3] = 1

array[4] = 2 ...................1<=2 right = mid = 3

left = 3 right =3......loop terminates( base case : if left is equal to right , it means that the search space has been narrowed down to a single element . Return arr[left]

Ans is 3