FREEMIND

Date

Page

Lecture-47

(Binary Search Algo)

⭐ Binary Search Code:-

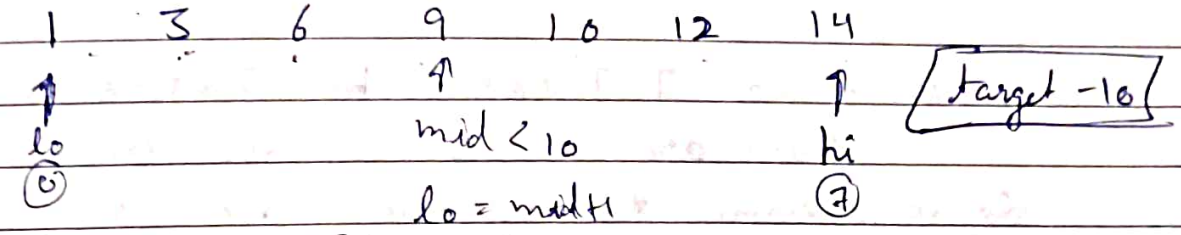
```
int BinarySearch(Vector<int>& input; int target){
    //define search space
    int lo = 0;          //start of search space
    int hi = input.size() - 1; //end of search space

    while (lo <= hi){
        //calc the midpoint for the search space
        int mid = (lo + hi)/2;
        // mid = lo + (hi - lo)/2

        if (target == input[mid]){
            return mid;
        } else if (target < input[mid]){
            hi = mid - 1  // Discard the right of mid.
        }
        else {            // Discard the left of
            lo = mid + 1      mid.
        }
    }
    return -1;
}
int main(){
Vector<int> V{1, 3, 6, 9, 10, 12, 14};
    int target = 10;
    cout << binarySearch(V, target) << "\n";
    return 0;
}
```

FREEMIND

Date

Page

① Time complexity Analysis



$$N + \frac{N}{2} + \frac{N}{4} + \frac{N}{8} + \cdots\cdots 1$$

$$\frac{N}{2^0} + \frac{N}{2^1} + \frac{N}{2^2} + \frac{N}{2^3} + \cdots\cdots \frac{N}{2^k}$$

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

$$N = 2^k$$

$$\log_2 N = \log_2 2^k$$

$$\log_2 N = k \log_2 2$$

$$k = \log_2 N$$

$O(\log N)$

10 12 14
lo mid hi
mid > target
hi = mid - 1

10 12
lo hi
mid = lo
mid = lo
target found

② Space complexity :- O(1)

③ Recursive Binary Search Implementation

f(input, target, lo, hi) =

[lo, hi] denotes search space

finds the target inside input [lo, hi]

```
int mid = (lo+hi)/2; if(lo>hi) return -1;
if (input[mid] == target) return mid;
if (input[mid] < target) {
return f(input, target, mid+1, hi)
}else{
return f(input, target, lo, mid-1)
}
```

FREEMIND

Date

Page

⇒ Time complexity ⇒ $O(\log n)$
Space complexity ⇒ $O(\log n)$

⇒ When ; lo = INT_MAX , hi = INT_MAX then there are chances of overflow
So, to overcome this problem we

Use $mid = \frac{lo + hi + lo - lo}{2}$

$= \frac{2\,lo + (hi - lo)}{2}$

$\boxed{mid = lo + \frac{(hi - lo)}{2}}$ // Modified mid to tackle overflow

Q-VI) Find the first occurance of a given element x, given that the given array is sorted. If no occurance of x is found then return -1.

input :- arr = [2, 5, 5, 5, 6, 6, 8, 9, 9, 9]
x = 5 → m → arr[m] == target
index m might be a possible ans.
this m might be our ans. or we can find something better than this.

output :- 1

Method 1 :- Linear Search

Method 2 :- Binary Search ⇒

```
# include <iostream>
# include <vector>
  Using namespace std;
```

→ Time complexity → O(log n)
→ Space " → O(1)

FREEMIND
Date
Page

```
Void firstOccurance(Vector<int> &input, int target){

    int lo = 0;
    int hi = input.size() - 1;
    int ans = -1;

    while (lo <= hi){

        mid = lo + (hi-lo)/2

        if (a[mid] >= target){
            ans = mid
            hi = mid-1
        }

        if (input[mid] = target){
            ans = mid;
            hi = mid-1
        }else if( input[mid] > target){
            hi = mid - 1;
        }else{
            lo = mid + 1;
        }
    }
    return ans;
}
int main(){
    n. → enter
    Vector → enter
    target → enter

    cout << firstOccurance(Vector, target) << "\n";
    return 0;
}
```

Imp. for Interview

FREEMIND

Date

Page

Q② Find the square root of the given non-negative value x. Round it off to the nearest floor integer value.

input x = 4	11
output :- 2	3

$\sqrt{x} \rightarrow [1, x]$

within this range find the biggest value whose square is less than or equal to x.

Method 1 ⇒ [linear search]

```
ans = -1
for (i = 1; i <= x; i++) {

    if (i * i <= x) {
        ans = i;
    } else {
        break;
    }
}
```

Time complexity $O(\sqrt{n})$

Method 2 ⇒ Binary Search

[1, n]

lo = 1, hi = n

```
int ans = -1;
while (lo <= hi) {
    int mid = lo + (hi - lo)/2;
    if (mid * mid <= target) {
        ans = mid;
        lo = mid + 1;
    } else {
        hi = mid - 1;
    }
}
```

sqrt (int n)

```
int main() {
    cout << sqrt(24) << "\n";
```

FREEMIND

Date

Page

Lecture - 45

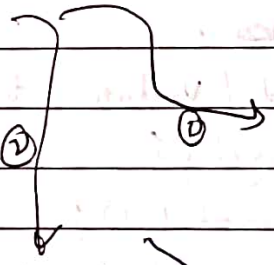
(Problem Solving on Binary Search - 1)

Q1 → Given an array of integers 'a' that is sorted in non-decreasing order, Find the first and the last position of the given 'target' element in the sorted array. Followed zero-based indexing.
if target is not found, return [-1, -1].
input :- arr[] = [1, 2, 3, 3, 3, 5, 11] , target = 3

output :- [2, 4]

Sol :- ~~Same as~~ ① For 1st occurance → Same as Q1 previous lec

② For last occurance → 1st occurance vale funct. ko hi use karke usse ek bade element ki 1st position (x) nikala and (x-1) will be the last occurance of target element.



Approach :-

```
if (a[mid] <= target) {
    lo = mid+1;
    if (a[mid] == target) {
        ans = mid;
    }
} else {
    hi = mid-1;
}
```

```
if (input[mid] > target) {
    ans = mid;
    hi = mid-1;
} else lo = mid+1;
```

Time complexity
O(logn)

Space complexity
O(1)

Date
Page

```
Void lower_bound (vector<int> & input, int target) {
    | Same as prev. ans.
}
Void Upperbound (Vec                    n )
    | Same {but target will be changed
    |      acc. to approach (1) }
}
int main () {
        n → enter
        Vector → enter
        target → enter
        Vector <int> result [ ];
        int  lb = lowerbound (vector, target)
        if (input [lb] != target) {
                result.push_back (-1);
                result.push_back (-1);
        }else{
                result.push_back (lb);
                result.push_back (Ub -1);
        }
        Cout << result[0] << " " << result[1] << "\n";

        return 0;
}
```

FREEMIND

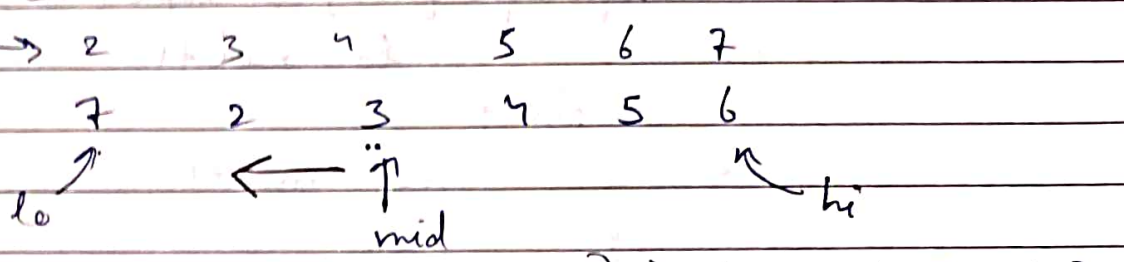
Date

Page

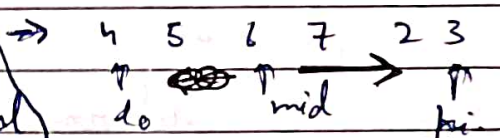
Q:- A rotated sorted array is a sorted array on which rotation operation has been performed some no. of times. Given a rotated sorted array, find the index of the minimum element in the array. Follo 0-based indexing.
It is gaurantee that all the elements in the array are unique.

- input :- Arr[] = {3, 4, 5, 1, 2}
- output :- 3

Sol:- eg:-



a[lo] > a[hi] → sorted Rotated



↳ if a[mid] > a[lo]
lo = mid + 1; //min. element is at right
if a[mid] < a[lo]
hi = mid - 1; //min. element is at left

terminals → a[mid] > a[mid+1]
ans = mid + 1
a[mid] < a[mid-1]
ans = mid

```
int findMinimumInSortedRated(Vector<int>&input)
    if (input.size() == 1)  return input[0];
    int lo = 0 , hi = input.size() - 1;
    if (input[lo] < input[hi]) { // Sorted array.
        return lo;
    }
```

Date

Page

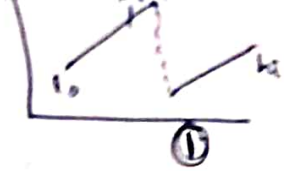
```
while (lo <= hi) {
    int mid = lo + (hi-lo)/2

    if (input[mid] > input[mid+1]){
        return mid+1;
    } if (input[mid] < input[mid-1]) {
        return mid;
    }
    if (input[mid] > input[lo]) {
        lo = mid+1
    } else {
        hi = mid-1
    }
}
return -1;
}
int main(){
    n -> enter
    vector -> enter
    cout << funct. <<"\n";
    return 0;
}
```

Q3> Given the rotated sorted array of integers, which contains distinct elements, and an integer target, return the index of target if it is in the array. Otherwise return -1.

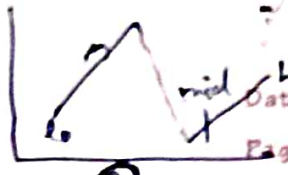
Input:- Arr[] = {3, 4, 5, 1, 2} , target = 4

output:- 1

Method 1:- Apply Binary search 3 times
① To find min. element
② From [lo to min.]
③ From [min. to hi] and get target value



①



②

FREEMIND

Date

Page

Method 2 :-

```
#include <iostream>
#include <vector>
using namespace std;

int binarySearchSortedRotated(Vector<int>& input[], int target)

    int lo = 0 , hi = input.size()-1;

    while (lo ≤ hi) {
        int mid = lo + (hi - lo)/2 ;

        if (input[mid] == target)  return mid;

        if ( input[mid] > input[lo] ) {
            if ( target ≥ input[lo] && target ≤ input[mid]) {
            ~~if (target >= lo && target <= mid)~~
                hi = mid - 1;
            } else {
                lo = mid + 1;
            }
        } else {
            if (target ≥ input[mid] && target ≤ input[hi
                lo = mid + 1;
            } else {
                hi = mid - 1;
            }
        }
    }
    return -1;
}

int main() {
    n → enter ; vector → enter ; target → enter;
    cout << funct << "\n";
```

Case ①

Case ②

Time complexity → O(log n)

Space → O(1)

(H.W)

Q> Search element in Rotated Sorted Array with duplicate elements. Return 1 if the element is found, else return -1.

input :- [0, 0, 0, 1, 1, 1, 2, 0, 0, 0] , target = 2

lo++ lo hi hi--

output :- 1

Logic -> Terminal elements ko remove karenge so that lo or hi me compare kar sakein or binary search apply kar sakein (for Sorted Rotated array)

```
if (lo = hi)
    lo++ , hi--
```

FREEMIND

Date

Page

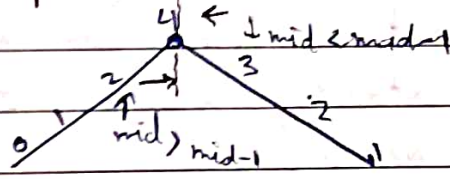
Lecture -46

(Binary Search Problems - 2)

Q:- Given a mountain array 'a' of length greater than 3, return the index 'i' such that arr[0] < arr[1] < ... < arr[i-1] < arr[i] > arr[i+1] > ... > arr[arr.length - 1]. This index is known as the peak index.

Input:- arr = [0, 4, 1, 0]

Output:- 1




```
int PeakInMountainArray(Vector<int>& input){

    lo = 1 , hi = input.size() - 1;
    int ans = -1;
    while (lo <= hi){

        int mid = lo + (hi - lo)/2;

        if (input[mid] > input[m-1]) {
            ans = mid;
            lo = mid + 1;
        } else {
            hi = mid - 1;
        }
    }
    return ans
}
int main () {
    cout.tie -> tie
    Vector enter
    Cout << PeakInMountainArray(Vector) << "
    return 0;
}
```

FREEMIND

Date

Page

Q: A peak element is an element is an element that is strictly greater than its neighbours. Given a 0-indexed integer array nums, find a peak element and return its index. If the array contains multiple peaks, return the index to any of the peaks. You may imagine that nums[-1] = nums[n] = $-\infty$, In other words an element is always considered to be strictly greater than a neighbor that is outside the array.

input:- Arr[] = [1, 2, 1, 2, 6, 10, 3]

output:- Either index 1 or index 5 are the correct output. At index 1, 2 is the peak element and at index 5, 10 is the peak element.

Sol.⇒

```
int findPeak(vector<int>& input){
    int n = input.size()-1
    int lo=0, hi=input.size()-1;
    while(lo<=hi){
        int mid = lo+(hi-lo)/2

        if(input[mid]==0){
            if(input[mid]>input[mid+1]){
                return 0;
            }else{
                return 1;
            }
        }else if(input[mid]==n-1){
            if(input[mid]>input[mid-1]){
                return n-1;
            }else{
                return n-2;
            }
        }
    }
```

Date

Page

```
else {
    if (input[mid] > input[mid-1] && input[mid] > input[mid+1])
        return mid;
    } else if (input[mid] > input[mid-1]) {
        lo = mid + 1
    } else {
        hi = mid - 1
    }
}
}
return -1;
}
int main() {
    n -> enter
    Vector -> enter
    Cout << findpeak(Vector) << "\n";
    return 0;
}
```

Q3) Search the 'target' value in a 2d integer matrix of dimensions "n x m and return 1 if found, else return 0, This matrix has the following properties:

(i) integers in each row are sorted from left to right

(ii) 1st integer of each row is greater than the last integer of previous row.

input :-

Matrix = {[1,3,5,7], [10,11,16,20], [23,30,34,60]},

target = 3

output :- 1

1	3	5	7
10	11	16	20
23	30	34	60

FREEMIND

Date

Page

Sol: ① Method 1 → i = 0 to n
: j = 0 → m
linear search in 1D Array
}

② Method 2 → i = 0 → n
j e → m
Binary search in 1D Array
}

③ Method 3 → Copy 2D array in single 1D Array

{1, 3, 5, 7, 10, 11, 16, 20, 23, 30, 34, 60}
↳ nm-1

Apply Binary search

④ Method 4th →

1	3	5	7
10	11	16	20
23	30	34	60

3×4

mid = 5
↓
(x, y) = (1, 1) = 11

x = mid / m = 5/4 = 1

y = mid % m = 5%4 = 1

#Code

```
Void SearchMatrix (Vector<int> &a, int target)
    n = a.size() - 1
    m = a[0].size() - 1
    int lo = 0, hi = n*m - 1;
    while (lo <= hi) {
        int mid = lo + (hi - lo)/2
        int x = mid / m
        int y = mid % m
```

```
if (a[x][y] == target) {
        return true;
} else if ( a[x][y] < target ) {
        ls = mid+1;
} else {
        hi = mid-1
}
}
return false;
}
int main() {
    ⇒ Vector → input   (Q.T.Q)
      target → 3
      cout << func << "\n";
}
```

Lecture 47

(Binary search Problems-3)

Q3 You have 'n' ($n <= 10^5$) boxes of chocolate. Each box contains a[i] (a[i] <= 10000) chocolates. You need to distribute these chocolates among 'm' students such that the maximum no. of chocolates allocated to a student is minimum.

(a) One box will be allocated to exactly one student.
(b) All the boxes should be allocated.
(c) Each student has to be allocated at least one box.
(d) Allotment should be in contiguous order, for instance a student cannot be allocated box 1 and box 3, skipping box 2.

⇒ Calculate and return that minimum possible no.
Assume that it is always possible to distribute the chocolates

→ The first line of input will contain the ~~line~~ value of n,

the no. of boxes.
→ The second line of input will contain the n no.s denoting the no. of chocolates in each box.
→ The third line will contain the m no. of students.

- input :-

4
12 34 67 90
2

- Output :-

113

Time complexity :- $O(n \times \log(\sum a_i))$

Sol is

```
#include<iostream>
#include<vector>
using namespace std;

bool canDistChoco(vector<int>&arr, int mid, int s){
    int n = arr.size();
    int studentRequired = 1;
    int currSum = 0;

    for(i=0; i<n, i++){
        if(arr[i] > mid){
            return false;
        }
        if(currSum + arr[i] > mid){
            studentRequired++;
            currSum = arr[i];
            if(studentRequired > s) return false;
        }else{
            currSum += arr[i];
        }
    } return true;
}
```

```
int distChoco(Vector<int>& arr, int s) {
    int n = arr.size();
    int lo = arr[0];
    int hi = 0;
    for(int i=0; i<n; i++){
        hi += arr[i]
    }
    int ans = -1;
    while (lo <= hi) { int mid = lo + (hi-lo)/2;
        if (canDistChoco(arr, mid, s)) {
            ans = mid;
            hi = mid-1;
        } else {
            lo = mid+1;
        }
    } return ans;
}
int main() {
    n -> enter
    Vector -> enter
    s -> enter (no. of students)
    cout << distChoco(Vector, s);
}
```

Q 24 A new racing track for kids is being built in New York with 'n' starting spots. The spots are located along a straight line at positions x1, x2 ... xn ($x_i < 10^9$). For each i, $x_{i+1} > x_i$. At a time only 'm' children are allowed to enter the race. Since the race track is for kids, they may run into each other while runing. To prevent this, we want to choose the starting spots such that the minimum distance b/w any two of them is as large as possible. What

FREEMIND

Date

Page

is the largest minimum distance?

→ [The first line of input will contain the value of n, the no. of starting spots [The second line of input will contain the n no.s denoting the location of each spot [The third line will contain the value of m, no. of children.

- input

5

1 2 4 8 9

3

output

3

Time complexity → $O(n \log(X_n - X_1))$

Sol ⇒

```
bool canPlaceStudents(vector<int>& pos, int s, int mid) {
    int studentReqd = 1;
    int lastPlaced = pos[0];
    for (i = 1; i < pos.size(); i++) {
        if (pos[i] - lastPlaced >= mid) {
            studentReqd++;
            lastPlaced = pos[i];
            if (studentReqd = s) {
                return true;
            }
        }
    }
    return false;
}
int race(vector<int>& pos; int s) {
    int n = pos.size();
    int lo = 1;
    int hi = pos[n-1] - pos[0];
```

```
int ans = -1
while (lo <= hi) {
    int mid = lo + (hi - lo) / 2
    if (canPlaceStudents(pos, s, mid)) {
        ans = mid;
        lo = mid + 1;
    } else {
        hi = mid - 1;
    }
}
return ans;
}
int main() {
    n -> enter
    Vector -> enter
    s -> enter (No. of Students)
    cout << rasc(pos, s) << "\n";
}
```

(H.W) 3 Variations of these two problems

(ii) Aggressive Counts [From Q2]

(i) Book allocation (From Q1)