2/3/2024

PAGE No.
DATE / /

Day 5 OF DSA

Tasks.

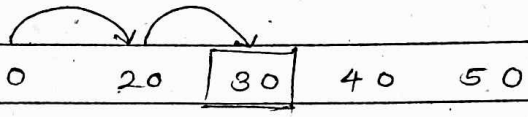
Checkbox	Arrays :.
☑	Searching Algorithms: linear search, binary search
☑	Sorting Algorithms: Bubble sort, selection sort
☑	Implement searching algorithms: linear search, binary search
☑	Implement sorting algorithms: bubble sort, selection sort.

Theory Part

PAGE No. / DATE 11

* Linear Search:

Linear search is defined as sequential search algorithm that starts at one end & goes through each element of a list until the desired element is Found.



key = 30

* Implementation :-

```
For( int i=0; i<n; i++){
    iF(arr[i] == x)
     return i;

return -1;
}
```

Time complexity : O(N)
space complexity : O(1)

PAGE No.
DATE / /

* Binary search :-

Binary search is defined as a searching algorithm used in a sorted array by repeatedly dividing the array in half.

steps :-

1) Divide the search space into two halves.

mid = low + (high - low) / 2

2) Compare mid element with key

3) If found then print.

4) else.

key < mid → go in left side

key > mid → go in right side

5) & continue this process

* Types of Binary search :-

(1) Iterative Binary search

(2) Recursive Binary search.

PAGE No.
DATE 11

* Iterative Binary Search Algorithm

Function:-

```
binarys( int arr[], int left, int right,
                                int x)
{
  while( left <= right)
  {
    int m = left + (right - left) /2;
    if (arr[m] == x)
      return m;
    if (arr[m] < x)
      left = m + 1;
    else
      right = m-1;
  }
  return -1;
}
```

mid == x → (1)

mid < x → (2)

mid > x → (3)

Time Complexity :	O(logN)
space complexity :	O(1)

PAGE No.
DATE 1 1

* Recursive Binary search Algorithm:

```
int bs( int a[], int l, int r, int x)
{
    if (r >= l) {
        int mid = l + (r-l)/2;

(1)     if(a[mid] == x)
            return mid; ✓

(2)     if(a[mid] > x)
            return bs(a, l, mid-1, x); ✓

(3)     if( a[mid] < x)
            return bs( a, mid+1, r, x);

    }
    return -1;
```

Time complexity : O(log N)
space complexity : O(1)

PAGE No.
DATE / /

Bubble sort:-

It is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

steps:-

1. Traverse from left to right.
2. Compare adjacent elements & swapped it.
3. Continue this until data is sorted.

Eg.

1st

6 3 0 5

3 6 0 5

3 0 6 5

3 0 5 6

2nd

0 3 5 6 ✓ Sorted

Total No. of passes : n-1

Total No of comparisons : n * (n-1) / 2

PAGE No.
DATE / /

* Implementation :-

Function :-

```
void bubbleSort(int ar[], int n)
{
    int i, j;
    bool swapped)
    For(i=0; i<n-1; i++){
      swapped = False)
      For(j=0; j<n-i-1; j++){
        if(ar[j] > ar[j+1])
        {
          swap(ar[j], ar[j+1]);
          swapped = true;
        }
      }
      if (swapped == False)
          break;
}
```

Time Complexity : $O(N^2)$
Space Complexity : $O(1)$

PAGE No.
DATE 11

* Selection Sort -

Selection sort is a simple and efficient algo that works by repeatedly selecting the smallest element from the unsorted portion of the list & moving it to the sorted portion of the list.

(1) 64 25 12 22 11

(2) 11 | 25 12 22 64
Sorted | Unsorted

(3) 11 12 | 25 22 64
Sorted | Unsorted

(4) 11 12 22 | 25 64 ✓

PAGE No.
DATE / /

* Implementation:-

```
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    For(i=0; i<n-1; i++) {

    min_idx = i;
    For(j=i+1; j<n; j++) {

        if (arr[j] < arr[min_idx])
            min_idx = j;
    }

    if (min_idx != i)
        swap(arr[min_idx], arr[i]);
    }
}
```

Time Complexity : $O(N^2)$
Space complexity : $O(N)$

Problems :-

PAGE NO.
DATE / /

(1) Finding an element in an Array :-

Approach :-

```
For( int i=0; i<n ; i++)
{
    if (arr[i] == target)
    { return i; }
    else
        return -1;
}
```

Time Complexity : O(n)
Space Complexity : O(1)

Input - 5 10 15 20 25, t=15

Output - 2

PAGE No.
DATE / /

(2) Finding the first occurence of character in a string :-

Approach :-

```
For( int i=0; i< str.length; i++)
{
    if( str[i] == character)
    {
        return i;
    }
else
    return -1;
}
```

Time complexity : O(n)
Space complexity : O(1)

Input = " hello world ", character = 'o'.
01234

Output = 4

PAGE No. / DATE / /

③ Searching in sorted Array:

Input - 2, 5, 7, 10, 15, 20,
target = 7.

Approach:

```
while( l <= r)
{
   int mid = l + (r-l)/2

  if (arr[mid] == target)
{
     return mid;
}
else if( arr[mid] > target)
   {
     l = mid + 1;
   }
else if (arr[mid] < target)
   {
     r = mid - 1;
   }
else
   return -1;
}
```

Time Complexity : O(log n)
Space complexity : O(1)

PAGE No.
DATE / /

④ Finding peak element in an Array :-

Input = [1, 3, 20, 4, 1, 0]

Output = 2

Approach :-

```
int Peak ( int arr[], int low, int high)
{
    if ( low == high) return low;

    int mid = low + (high-low) / 2;
```

m-1	m	m+1
4	5	6

→ if(mid>0 && arr[mid] > arr[mid-1] &&
mid < high-1 && arr[mid] > [mid+1])
{ return mid;
}

m-1	m
6	5

→ if(mid > 0 && arr[mid-1] > arr[mid])
{ return peak(arr, low, ~~high~~ mid-1);
}

m	m+1
5	6

→ else { return Peak(arr, mid+1, high);
}
}

Time Complexity: $O(n^2)$
space complexity: $O(1)$

PAGE NO.
DATE / /

(5) Sorting an array :-

Input :- 5, 3, 8, 2, 1
Output :- 1, 2, 3, 5, 8

Approach :-

```
void bubbleSort( int arr[], int n)
{
    int i, j;
    bool swapped ~~= false~~;

    For( i=0; i<n; i++)
    {       swapped = false;
      For( j = i; j<n-i-1; j++)
        {  if (arr[j] > arr[j+1])
           { swap( arr[j], arr[j+1]);
             swapped = true;
           }
        }

    if ( swapped == false)
        break;
}
```

Print Array Using For loop

PAGE No.
DATE / /

(6) Couting inversions in an Array :-

Approach :-

(1) Use bubble sort ;
& take variables as
inversion_count = 0

(2) & in inner For loops if condit^n add
this variable with
increment

(3) & return the inversion_count ;

Time complexity : O(n)
Space Complexity : O(n)

PAGE NO.
DATE / /

(7) Selection sort :-

Sorting Array :-

Approach :-

(1) take min index for calculating min element

(2) then swap it with 1st element

Time complexity : O(n log n)
Space complexity : O(1)

PAGE No.
DATE 11

(8) kth smallest element :-

Approach :-

1) Write selection sort as it is

2) Only at the end of Funct^n add

arr[k-1] ✓

means if

arr.size = 5

& k = 3

then arr[3-1]

arr[2]

Time Complexity : $O(n^2)$
Space Complexity : $O(1)$