DAA

Assignment-02

1> int linearsearch (int arr[], int key, int size) {
for (int i=0; i<size; i++) {
if (arr[i] == key)
return i
} else if (arr[i]
return -1;
}

2> 1) Insertion Sort iterative

void insertioniter (int arr[], int size) {
for (int i=1; i<size; i++) {
int key = arr[i];
int j = i-1;
while (j>0 && arr[j]>key) {
arr[j+1] = arr[j];
j = j-1;
}
arr[j+1] = key;
}
}

ii) Insertion Sort recursive

void insertionrec (int arr[], int n) {
if (n<=1)
return;
insertionrec (arr, n-1);
int key = arr[n-1];
int j = n-2;

```
while(j>=0 && arr[j] > last){
    arr[j+1] = arr[j];
    j--;
}
arr[j+1] = last;
}
```

Insertion Sort is called online sort as it can sort list of elements as they are received one at a time, without having to wait for the entire list to be received or processed first

3)

	Best	Avg	Worst	Space
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$

4) ① Inplace :- Sorts i/p array by rearranging elements within array itself
   - → Bubble Sort
   - → Selection Sort
   - → Insertion Sort

② Stable :- Preserves the relative order of equal elements in the array Elements with the same value are sorted in same order
   - Bubble Sort
   - Insertion Sort
   - Merge Sort
   - Count Sort

Date
Page No.

3) Online ~~Sort~~ :- Sorts the stream of elemet as they arrive
- Insertion Sort

5) ① Recursive

```
int binarySearch (int arr[], int l, int r, int x){
    if ( r >= l) {
        int mid = l+r/2;
        if (arr[mid] == x)
            return mid;
        else if (arr[mid] > x)
            return binarySearch(arr, l, mid-1, x);
        else
            return binarySearch (arr, mid+1, r, x);
    }
    return -1;
}
```

② Iterative

```
int binarySearch (int arr[], int n, int x ){
    int l=0, r=n-1;
    while( l <= r){
        int mid=(l + (r) / 2
        if (arr[mid] == x)
            return mid
        else if (arr[mid]<x)
            l=mid+1;
        else
            r = mid-1;
    }
    return -1;
}
```

	Space Complexity
Binary Search (Recursive)	O(log n)
Binary Search (Iterative)	O(1)
Linear Search (Recursive)	O(n)
Linear Search (Iterative)	O(1)

6) Recurrence Relation expresses the time complexity of binary search algo in terms of sub problems the algo divides the i/p array in half each iteration & solves a sub problem of size n/2

$T(n) = T(n/2) + O(1)$;

$T(n)$ → array size (n)

$T(n/2)$ → New array size $(n/2)$

$O(1)$ → Extra constant Complexity

7) Algo:-

Step 1:- Start the i/p array in non-decreasing order

Step 2: Initialize two pointers i & j, to point to the first & last elements of the array

Step 3: While $i < j$, Compute the sum of $A[i] + A[j]$

Step 4: If Sum == K, return i & j

Step 5: If Sum < K, increment i by 1

Step 6: If Sum > K, increment j by 1

Time Complexity = O(n)

Date
Page No.

8) Quick Sort is widely used sorting algo. that has avg. time complexity of $O(n \log n)$ & is often faster than other popular sorting algo. QuickSort is particularly efficient for large data sorting & can be easily implemented in place to save memory resource. However, it's worst time complexity is $O(n^2)$ when i/p is already sorted.

9) arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5}

```
int getInvCount(int arr[], int n){
    int inv = 0;
    for (i = 0 to n-2, i++){
        for(j = i+1 to n-1, j++){
            if (arr[i] > arr[j])
                inv++;
        }
    }
    return inv;
}
```

10) * Best Case → The first element chosen should be the median of the array. If the pivot is chosen as the median at each step, then the partiong step will divide the array into 2 sub-arrays of equal size, resulting in balanced tree of recursive calls. In this case, the theme complexity of Quick Sort is $O(\log n)$

* Worst Case → In worst case the pivot element chosen at each step is either largest or smallest of sub array

Date
Page No.

11) Recurrence Relation:

Merge Sort:

Best case : $T(n) = 2T(n/2) + O(n)$

Worst case : $T(n) = 2T(n/2) + O(n \log n)$

Quick Sort:

Best case: $T(n) = 2T(n/2) + O(n)$

Worst case: $T(n) = T(n-1) + O(n)$

12) Yes, We can implement Stable version of selection Sort

```
void SelectionSort(int arr[], int n){
    for(int i=0; i<n-1; i++){
        int min=i;
        for(int j=i+1; j<n; j++){
            if(arr[j]<arr[min])
                min=j;
        }
    }
    int temp = arr[i];
    arr[i] = arr[min];
    arr[min] = temp;
  }
}
```

Date
Page No.

(d) It is not possible to load the entire array into the memory for sorting using internal sorting. In this case we will need to use external sorting algo. that operate on disk rather than memory.

External sorting is the techniache used to sort large data sets that can not be held in memory address. It involves a combination of internal & external sorting techniques.

The most commonly used external sorting algo is external merge sort. In this algo the data is split into smaller parts that can fit into memory each chunck is sorted using internal sorting, where the data is read from the disk, merged & written back to disk