DAA Tutorial 3

1. Write linear search pseudocode to search an element in a sorted array with mini. comparisions

pseudocode:

```
int linearSearch (int arr, int n, int key)
  for i<0 to n-1
    if arr[i] = key
      return i
    return -1
```

2. Write pseudocode for iterative & recursive insertion sort. Insertion sort is called online sorting. Why? What about other sorting algo that has been discussed in lectures?

Iterative insertion sort

```
void insertion sort (int arr[], int n)
{
      int i, temp, J;
      for (i=1 to n)
            t = arr[i];
            J = i;
            while (J>=0 && t<[i])
            {
                  arr[J+1] = arr[J];
                  J--;
```

```
            arr[j+1]=t;
        }
    }
```

Recursive Insertion sort:-

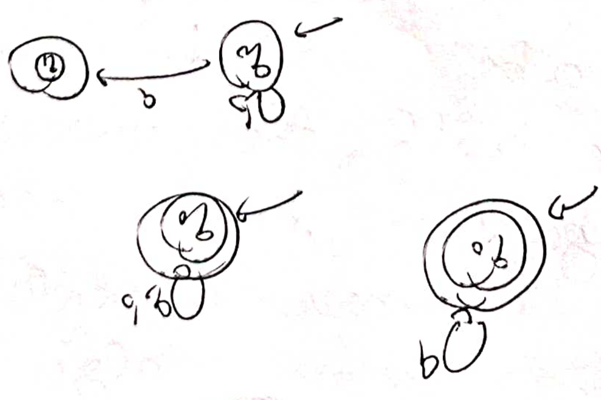
```
void Insertion(int A[], int n)
{
    if(n <= 1)
        return;
    insertion(A, n-1);
    int last = A[n-1];
    int j=n-2;
    while( j >= 0 && A[j] > last)
    {
        A[j+1] = A[j];
        j--;
    }
    A[j+1] = last;
  }
}
```

Insertion sort is also called online sorting algo. because it will work if the elements to be sorted are provided one at a time with the understanding that the algorithm must keep the sequence sorted as more elements are added in, other sorting algo, like bubble sort, insertion sort etc are considered external sorting technique as they need the data to be sorted in advance.

3. Complexity of all the sorting algo. that has been discussed in lectures.

	Best case	worst case
Bubble sort	$O(n^2)$	$O(n^2)$
Selection sort	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n)$	$O(n^2)$
Count sort	$O(n)$	$O(n+k)$
Quick sort	$O(n \log n)$	$O(n^2)$
Merge sort	$O(n \log n)$	$O(n \log n)$
Heap sort	$O(n \log n)$	$O(n \log n)$

4. Divide all sorting algo into inplace/ stab online sorting.



	Inplace	Stable	Online
Bubble	✓	✓	✗
Selection	✓	✗	✗
Insertion	✓	✓	✓
Count	✗	✓	✗
Quick	✓	✗	✗
Merge	✗	✓	✗
Heap	✓	✗	✗

5. Write recursive/iterative pseudocode for binary search. What is Time & Space complexity of linear & Binary Search.

Iterative

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

Recursive

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

Linear Search

Iterative : TC : O(n)
SC O(1)

Recursive : TC : O(n)
SC : O(n)

Binary Search

Iterative : TC = O(log n)
SC = O(1)

Recursive : TC = O(log n)
SC = O(log n)

6. Write recurrence relation for binary recursive search.

$T(n)$
$\downarrow$
$T(n/2)$
$\downarrow$
$T(n/4)$
$\vdots$
$T(n/2^k)$

Recurrence Relation $= T(n/2) + O(1)$

7. Find two indexes such that $A[i] + A[j] = K$ in mini. time complexity.

```
int n;
int A[n];
int key;
int i=0,j=n-1;
while (i<j)
{
    if((A[i] + A[j]) = key))
        break;
    elseif ((A[i]+ A[j]))>key)
        j--;
    else
        i++;
}
cout <<i <<" " <<j;
```

Time Complexity $= O(n \log n)$

8. Which sorting is best for practical use? Explain:

Factors affecting on deciding whether a sorting algo is good/not?

1. Run Time.
2. Space
3. Stable
4. No. of swaps
5. will the data fit in RAM.

9. What do you mean by number of inversion in an array? count the no. of inversions in array arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5} using merge sort.

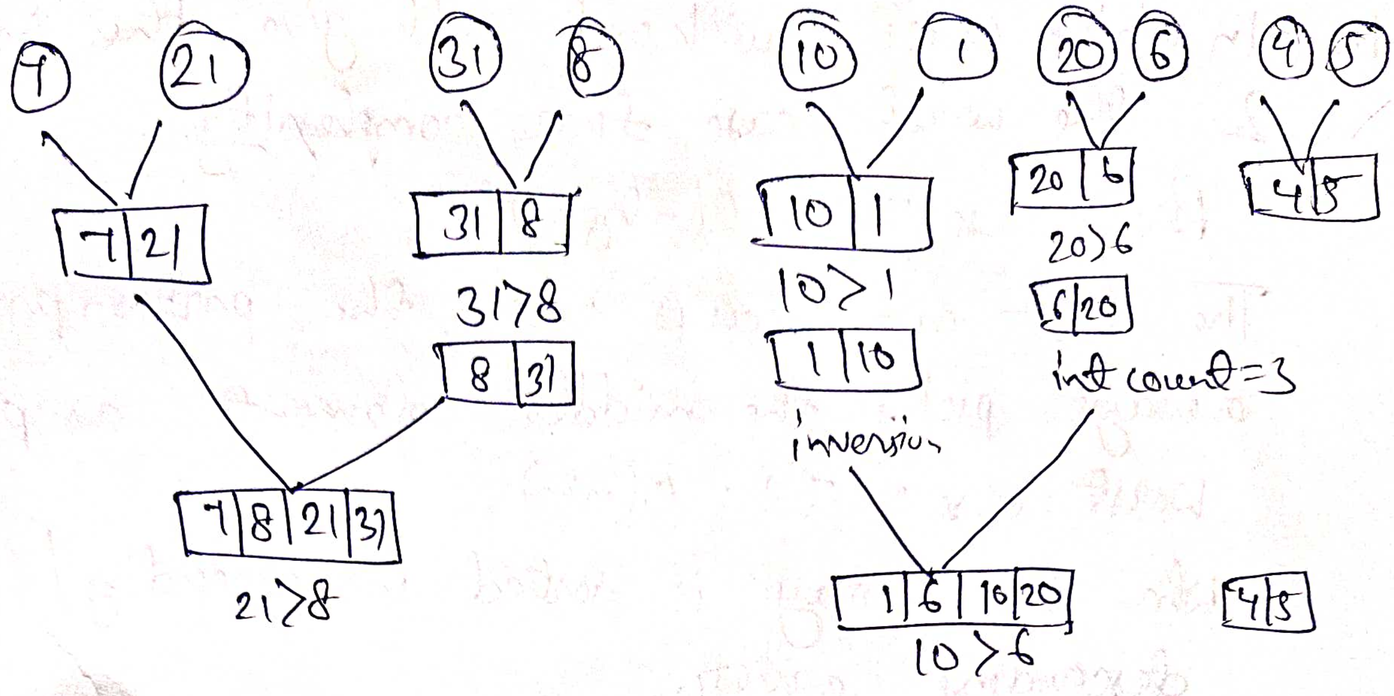
Inversion in an array indicates how far the array is from being sorted. If the array is already sorted, the inversion count is 0, but if the array is sorted in reverse order, then the inversion count is max.

Condition for inversion:-

$a[i] > a[j]$ & $i < j$

7	21	31	8	10	1	20	6	4	5

Dividing the array:



int count = 5

1	6	7	8	10	20	21	31

4	5

7>1, 7>6, 8>1, 8>6, 21>10, 21>20, 31>1,
31>6, 31>10, 31>20, 21>1, 21>6

1	4	5	6	7	8	10	20	21	31

int count A = 17

6>4, 6>5, 7>4, 7>5, 8>4, 8>5, 10>4,
10>5, 20>4, 21>4, 21>5, 31>4, 31>5

inversion count = 31

10. In which cases Quicksort will give the bes
& the worst case time complexity.

Best case TC : $O(n \log n)$

The best case occurs when the partion proc
always picks the middle element as piv

Worst case : TC : $O(n^2)$

When the array is sorted in ascending/
descending order.

11. Write Recurrence Relation of merge &
Quicksort in best & worst case ? what
are the similarities & diff. b/w
complexities of ~~two~~ algo. & why ?

Best cases

Mergesort = $2T(n/2) + n$

Quicksort = $2T(n/2) + n$

Worstcases

Mergesort = $2T(n/2) + n$

Quicksort = $T(n-1) + n$

Similarities: They both work on the concep
of divide & conquer algo,
Both have the best case complexity
of $O(n \log n)$.