Name:- Ishu Verma
Section CST SPL 2
Roll no:- 04

Tutorial 3

Q1.) Write a linear search pseudo code to search an element in a sorted array with minimum comparisons.

```
int linear_search (int A[], int n, int t)
{   if(abs(A[0]-t) > abs(A[n-1]-t))
        for(i= n-1 to 0 ; i--)
            if (A[i]== t){ return i; }.
    else
        for (i=0 to n-1 ; i++)
            if(A[i]==t)
                return i;
}
```

Q2.) Iterative Insertion Sort

```
void insertion (int A[] ,int n)
{
    for(i=1 to n)
    {   t= A[i];
        j = i;
        while (j>=0 && t < A[j])
        {
            A[j+1] = A[j];
            j--;
        }
        A[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 algorithm. 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 algorithms like bubble, sort, insertion sort, heap sort etc are considered external sorting technique ~~or~~ as they need the data to be ~~stored~~ sorted in advance.

Q3:) Complexity of all sorting algorithms that have 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	$O(n \log n)$	$O(n \log n)$

(Q4.)

	Inplace	Stable	Online
Bubble	✓	✓	X
Selection	✓	X	X
Insertion	✓	✓	✓
Count	X	✓	X
Quick.	✓	X	X
Merge.	X	✓	X
Heap	✓	X	X

(Q5.) Recursive / Iterative pseudo code for 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 :- Time complexity = $O(n)$
Space complexity = $O(1)$

Recursive :- Time Complexity = $O(n)$
Space complexity = $O(n)$

Binary Search

Iterative :- Time complexity = $O(\log n)$
Space complexity = $O(1)$

Recursive :- Time complexity = $O(\log n)$
Space complexity = $O(\log n)$

Q6.)

$T(n)$

$\downarrow$

$T(n/2)$

$\downarrow$

$T(n/4)$

$\downarrow$

$\vdots$

$\downarrow$

$T(n/2^k)$

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

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

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

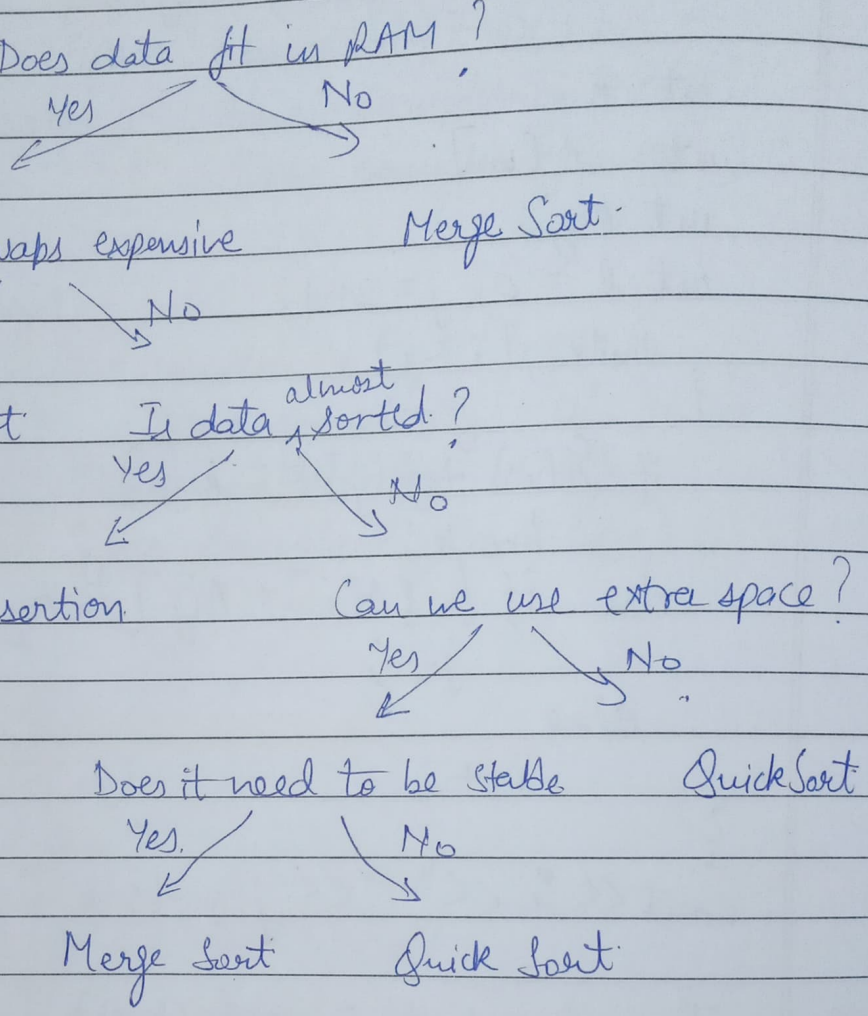
Time Complexity = $O(n \log n)$

Q8.) Which sorting is best for practical use? Explain.

A8.) Factors affecting on deciding whether a sorting algorithm is good ~~better~~ or not:-

1.) Run Time
2.) Space
3.) Stable
4.) No. of swaps
5.) ~~Is there a need of~~ or will the data fit in RAM.

There is no best sorting algorithm. It depends on the situation or the type of array provided.



Q9.) What do you mean by number of inversions in an array ? Count the number of inversions in an array arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5} using merge sort.

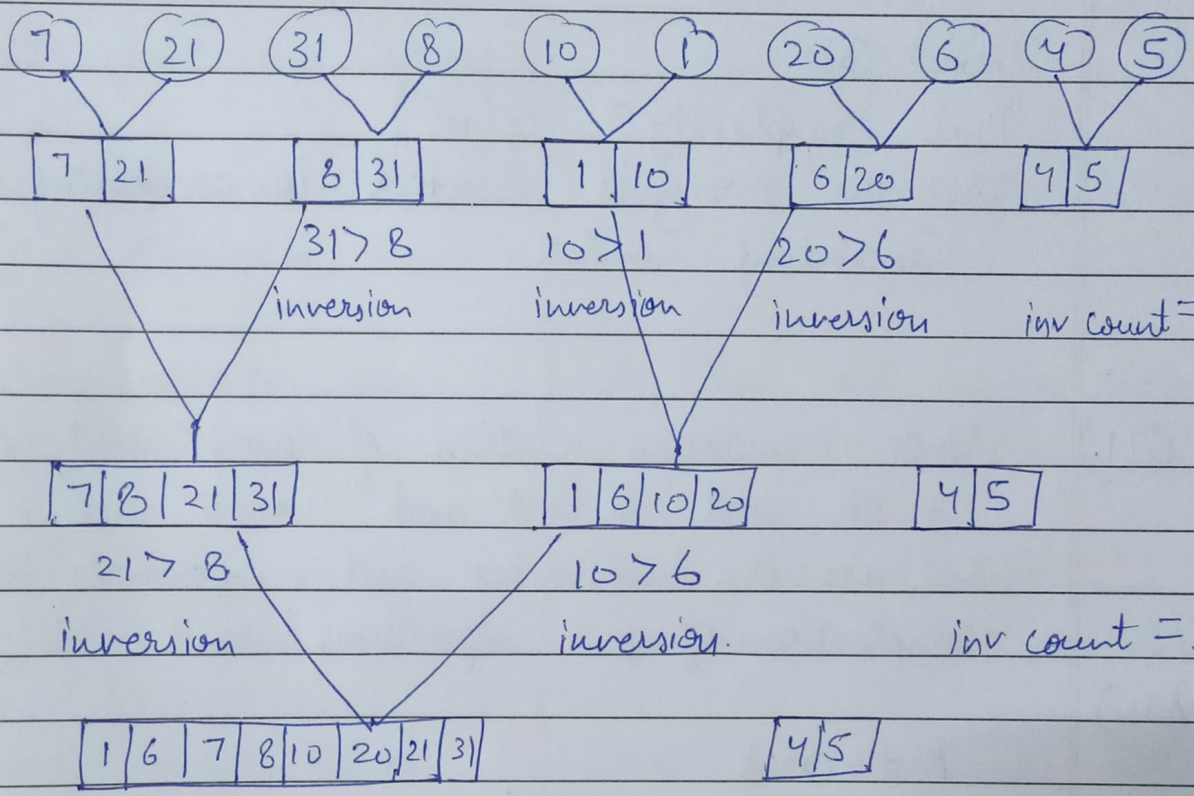
Ans. Inversion in an array ~~means~~ indicates how far the array is from being sorted. If the array is already sorted, the inversion count is 0, but if the array is sorted in reverse order, then the inversion count is maximum.

Condition for inversion:-

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

7	21	31	8	10	1	20	6	4	5	

Dividing the array



7>1, 7>6, 8>1, 8>6, 21>10, 21>20, 31>1, 31>6, 31>10, 31>20
21>1, 21>6 total inv in this step = 12

inv count = 17

1	4	5	6	7	8	10	20	21	31

6>4, 6>5, 7>4, 7>5, 8>4, 8>5, 10>4, 10>5, 20>4, 20>5,
21>4, 21>5; 31>4, 31>5 total inv in this step = 14

inv count = 31 Ans

Q10.) Which case quick sort will give the best and worst case complexity?

A10.)

Best Case
Time Complexity = $O(n \log n)$
The best case occurs when the partition process always picks the middle element as pivot.

Worst Case.
Time Complexity = $O(n^2)$
When the array is sorted in ascending or descending order.

Q11.) Write recurrence relation of merge sort and quick sort in best and worst case.
What are the similarities and differences between complexities of two algorithms and why?

A11.)

Best cases
Merge Sort = $2T(n/2) + n$
Quick Sort = $2T(n/2) + n$

Worst case
Merge sort = $2T(n/2) + n$
Quick Sort = $T(n-1) + n$

Similarities:- They both work on the concept of divide and conquer algorithm.
Both have best case complexity of $O(n \log n)$.

Differences :-

	Merge Sort	Quick Sort
1.)	The array is divided into just two halves.	The array is divided in any ratio.
2.)	Worst case complexity is $O(n \log n)$.	Worst Case complexity $O(n^2)$.
3.)	It requires extra space ie NOT inplace.	It does not require extra space ie inplace.
4.)	It is external sorting algorithm and stable.	It is internal sorting algorithm and NOT stable.
5.)	Works consistently on any size of data set.	Works fast on small data sets.

Q12.) Selection sort is not stable by default but you can write a version of stable selection sort.

```
void selection(int A[], int n)
{
  for(int i=0; i<n-1; i++)
  {
    int miis = i;
    for(int j= i+1; j<n; j++)
    {
      if(A[miis] > A[j])
        miis = j;
    int key = A[miis];
    while( miis > i)        //instead of swapping, do
    {                       //shifting of elements.
      A[miis] = A[min-1];
      min--;
    }
    A[i] = key;
  }
}
```

Q13.) Bubble sort scans the whole array even when the array is sorted. Can you modify the bubble sort algorithm so that it does not scan the whole array once it is sorted?

```
void bubblesort(int A[], int n)
{
    int i, j;
    int f=0;
  for(i=0; i<n; i++)
  {
    for(j=0; j<n-1; j++)
```

```
{
    if (A[j] > A[j+1])
    {
        swap (A[j], A[j+1])
        f = 1;
    }
}
if (f == 0)
    break;
}
```

Your computer has 2GB RAM and you are to sort a 4 GB array. Which algorithm you are going to use for this purpose and why? Also explain the concept of external and internal sorting.

When the data set is large enough to fit inside RAM, we ought to use Merge Sort because it uses the divide and conquer approach in which it keeps dividing the array into smaller parts until it can no longer be splitted. It then merges the array ~~divides~~ divided in n parts. Therefore at a time only a part of array is taken on RAM.

External Sorting.

It is used to sort massive amounts of data. It is required when the data does not fit inside RAM and instead they must reside in the slower external memory.

During sorting, chunks of small data that can fit in main memory are read, sorted and written out to a temporary file.

During merging, the sorted subfiles are combined into a single large file.

Internal Sorting.

Internal sorting is a type of sorting which is used when the entire collection of data is small enough to reside within RAM. Then there is no need of external memory for program execution. It is used when input is small.

Eg:- Insertion sort, quick sort, heap sort etc.