TUTORIAL -3

Q) 1 Write a linear search pseudocode to search an element in a sorted array with minimum compari

```
int linear_search (int A[], int n, int
{
    for (int i = 0 to n-1)
    {   if (A[i] == key)
            return i;
    }
```

Q) 2 Write pseudo code for iterative and recursive insertion sort. Insertion sort is called online sorting. Why? What about other sorting algorithms that have been discussed in lectures?

ITERATIVE

```
void Insertion_sort ( int A[], int n)
{
    t = 0;
    for ( i = 1 to n )
    {
        t = a[i];
        j = i-1;
        while (j >= 0 && t < A[i])
        {
            A[j+1] = A[j]
            j--
        }
        A[j+1] = t;
    }
}
```

PAGE No
DATE : / 202

RECURSIVE

```
void insertion_sort (int A[], int n)
{
  if (n <= 1)
      return;
  insertion_sort (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 called online sort 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 such as bubble sort, heap sort etc are considered external sorting techniques as they need the data to be sorted in advance.

3. Complexity of all sorting algorithms 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 the sorting algorithms into inplace / stable / online sorting.

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

5. Write recursive / iterative pseudo code for binary search. Write the time complexity and space complexity of Linear search (Recursive and iterative)

RECURSIVE

```
int BS( int A[], int front, int end, int key)
{   if ( front < end) {
    int mid = (front + end) / 2;
      if ( A[mid] == key)
            return mid;
      else if ( A[mid] > key)
      return BS( A, front, mid-1, key);
      else if (A[mid] < key)
            return BS(A, mid+1, end, key);
      else
            return -1;
} }
```

Iterative

Linear Search

Time complexity : $O(n)$

Space complexity : $O(1)$

Binary Search

Time complexity = $O(\log n)$

Space complexity = $O(1)$

Recursive

Linear Search

Time complexity = $O(n)$

Space complexity = $O(n)$

Binary Search

Time complexity = O(log n)

Space complexity = o(log n)

6 Write recurrence relation for binary recursive search

$T(n)$

↓

$T(n/2)$

↓

$T(n/4)$

↓

⋮

$T(n/2^R)$

Recurrence Relation : $T(n/2) + o(1)$

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

```
int n ; //no. of elements in array
int A[n];
int key;
int i = 0, j = n-1;
while (i < j)  → o(n)
 {
    if ((A[i] + A[j]) == key)
```

sort(A, A+n) // sorting of array ↳ o(nlogn)

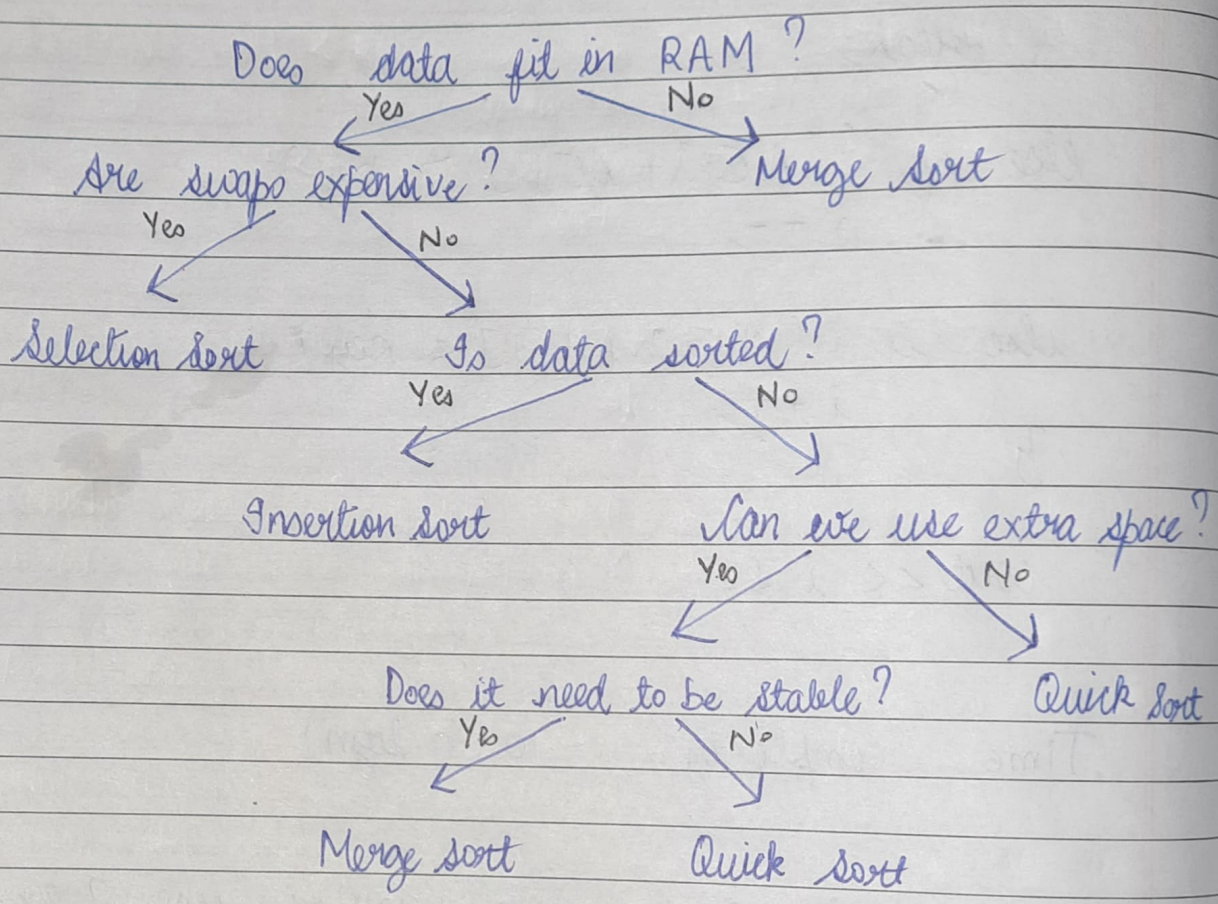
```
        break;

    else if ((A[i]+A[j])> key)
            j--;

    else if ((A[i]+A[j])< key)
            i++;
    }

    cout << i << " " j;
```

Time complexity = O(n logn)

8) Which sorting is best for practical uses? explain.
Factors affecting or deciding a good sorting algorithm

1. Running Time
2. Space
3. Stable
4. Swaps
5. Is data already sorted?
6. Will the data fit in RAM?

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



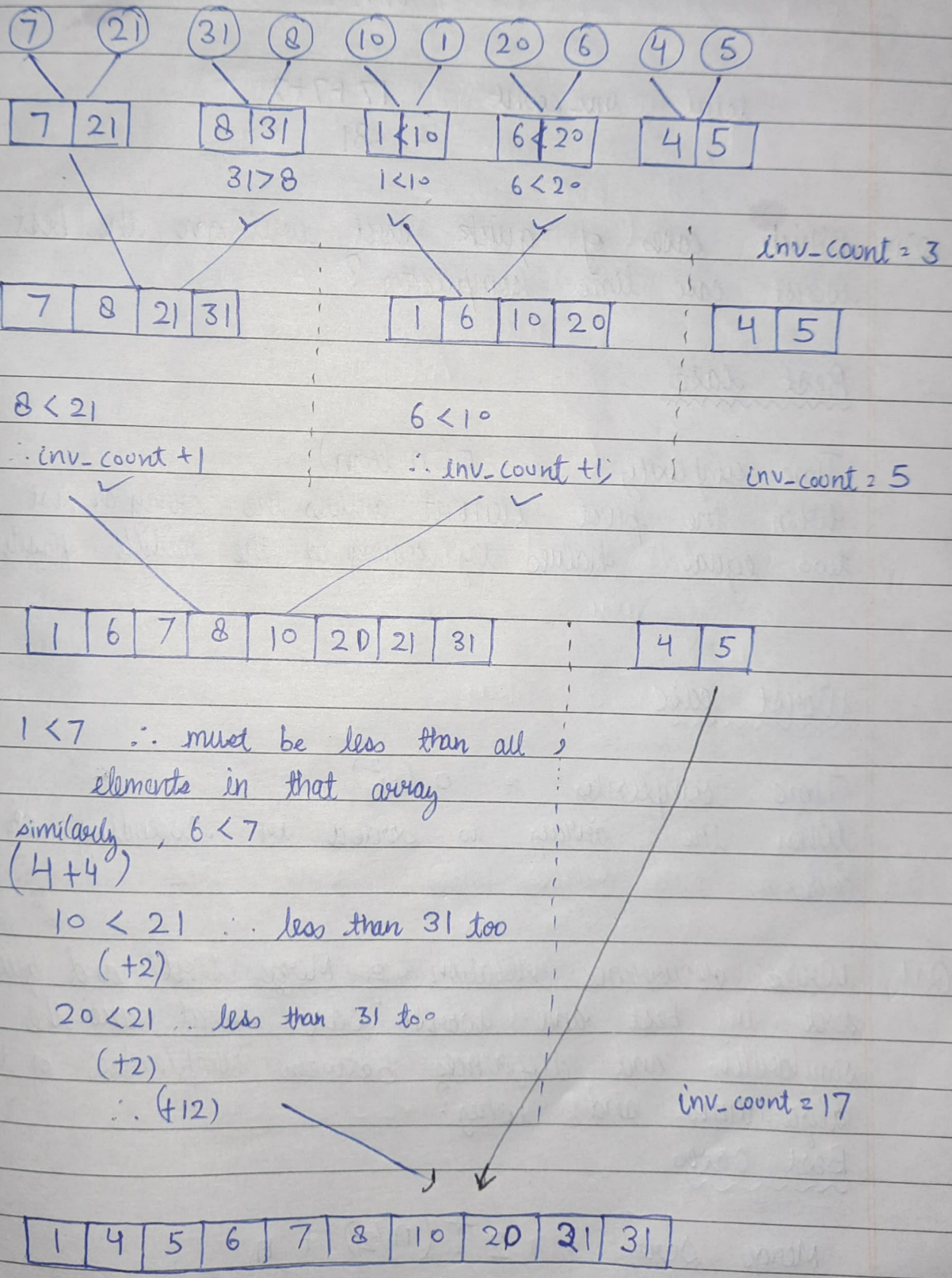
9 What do you mean by number of inversions in an array? Count the number of inversions in array arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5} using merge sort.

Inversions in an array indicate how far is an array from being sorted.

Two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j

7	21	31	8	10	1	20	6	4	5

dividing the array



4 < 6 ∴ less than 7 more elements in that array ∴ (+7)

5 < 6 ∴ less than 7 other elements ∴ (+7)

total inv_count = 17 + 7 + 7
= 31

Q)10 Which cases quick sort will give the best and worst case time complexity?

Best case

Time complexity = O(n log n)
When the pivot element divides the array or list in two equal halves by coming in the middle position.

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?

Best cases

Merge sort = $2T(n/2) + n$

Quick Sort = $2T(n/2) + n$

Worst case

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

similarities

They both rely on divide and conquer algorithm.
Both have best case of $o(n \log n)$

Difference

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

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

```
void stable_SS (int A[], int n)
{
    for( int i=0; i<n-1; i++)
    {
        int min = i;
        for ( int j = i+1; j<n; j++)
            if (A [min] > A [j])
                min = j;
        int key = A[min];

        while ( min > i)
        {
            A[min] = A[min-1];
            min--;
        }
        A [i] = key;
}
```

13. Bubble sort scans whole array even when array is sorted. Can you modify the bubble sort so that it doesn't scan the whole array once it is sorted.

```
void bubblesort (int A[], int n)
{
    int i, j;
    bool swapped;
```

```
for (int i = 0; i < n-1; i++)
{
    swapped = false;
    for (j = 0; j < n-i-1; j++)
    {
        if (A[j] > A[j+1])
        {
            swap (&A[j], &A[j+1])
            swapped = true;
        }
    }
    if (swapped == false)
        break;
}
}
```

14 Your computer has 2 GB 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.

Merge Sort is preferred when the data to be sorted is very large.

Merge Sort uses the divide and conquer approach in which it keeps dividing the array into smaller parts until which it can no longer be divided.

It then merges the array divided in 'n' parts.

PAGE No
DATE : / 202

External Sorting

It is used to sort massive amounts of data
It is required when the data being sorted do not fit into the main memory of a computing device (RAM) and instead they must reside in the slower external memory.
In the sorting phase, chunks of small data that can fit in main memory are read, sorted and written out to a temporary file.
In the merging phase, the sorted sub-files are combined into a single larger file.

Internal Sorting

Internal Sorting are type of sorting which is used when the entire collection of data is small enough that sorting can take place within main memory. There is no need for external ~~mat~~ memory for execution of sorting program.
It is used when input is small.
Eg. Insertion sort, Quick sort, heap sort etc.