Tutorial → 3

(1)

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

(2) iterative insertion sort

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

Recursive insertion sort

```
void insertion (int arr[], int i, int n)
{   int temp = arr[i];
      int j = i;
   while (j>0  && arr[j-1] > value )
    {   arr[j] = arr[j-1];
        j--;
    }
      arr[j] = value;
   if (i+1 <= n)
      insertion( arr, i+1, n);
  }
}
```

In insertion sort, we give i/p one by one and place each one at right order with comparison from already traces element. We need not the whole array simultaneous to operate algorithm, so it is online algorithm.

Let A[] = {23, 1, 4, 2, 7}

step:

1: A[] = {23, 1, 4, 2, 7} (only take 23 in consider)

2: A[] = {1, 23, 4, 2, 7} (only take 23, 1 in consider)

... and so on

Insertion is online sorting rest of the sorting are offline sorting.

Q3, Q4

	T.C	S.C	Stable	Inplace	online
i) Bubble sort	n^2	1	✓	✓	×
ii) Selection sort	n^2	1	×	✓	×
iii) Insertion sort	n	1	✓	✓	✓
iv) Merge sort	$n \log n$	n	✓	×	×
v) Quick sort	$n \log n$	n	×	×	×
vi) Heap sort	$n \log n$	1	×	✓	×

Q5

Iterative Binary search

```
int binarySearch(int A[], int x)
{ int low=0, high= A.length-1;
  while (low<= high)
{ int mid = (low + high) /2;
  if (x == A[mid]) {
        return mid;
  } else if (x < A[mid]) {
        high = mid-1;
  } else { low = mid+1;
  }
```

Recursive Binary Search

```
bool binarySearch(int arr[], int l, int r, int key)
{ if (l > r) return false;
  int mid = (l+r)/2;
  if (arr[mid] == key) return true;          ---> 1
  else if (arr[mid] < key)                    T(n/2)
      return binarySearch(arr, mid+1, r, key);
  else
      return binarySearch(arr, l, mid-1, key);
                                              T(n/2)
}
```

	Linear	Binary
Recursive time complexity	$O(n)$	$O(\log n)$
Iterative time complexity	$O(n)$	$O(\log n)$
Iterative space complexity	$O(1)$	$O(1)$
Recursive space complexity	$O(\log n)$	$O(\log n)$

Q6

$$T(n) = 1\,T(n/2) + 1$$

$a = 1 \quad b = 2 \quad k = 0$

$\log_b a = k$

$\therefore O(\log n)$

Q7

```
bool checkPair(int A[], int n, int K)
{
  Take Hash Table H of size O(n)
  for(i=0 to n-1)
  { int x = K - A[i]
    if(H.search(x) is true)
        return 1
    H.insert(A[i])
  }
  return -1
}
```

T.C = $O(n)$

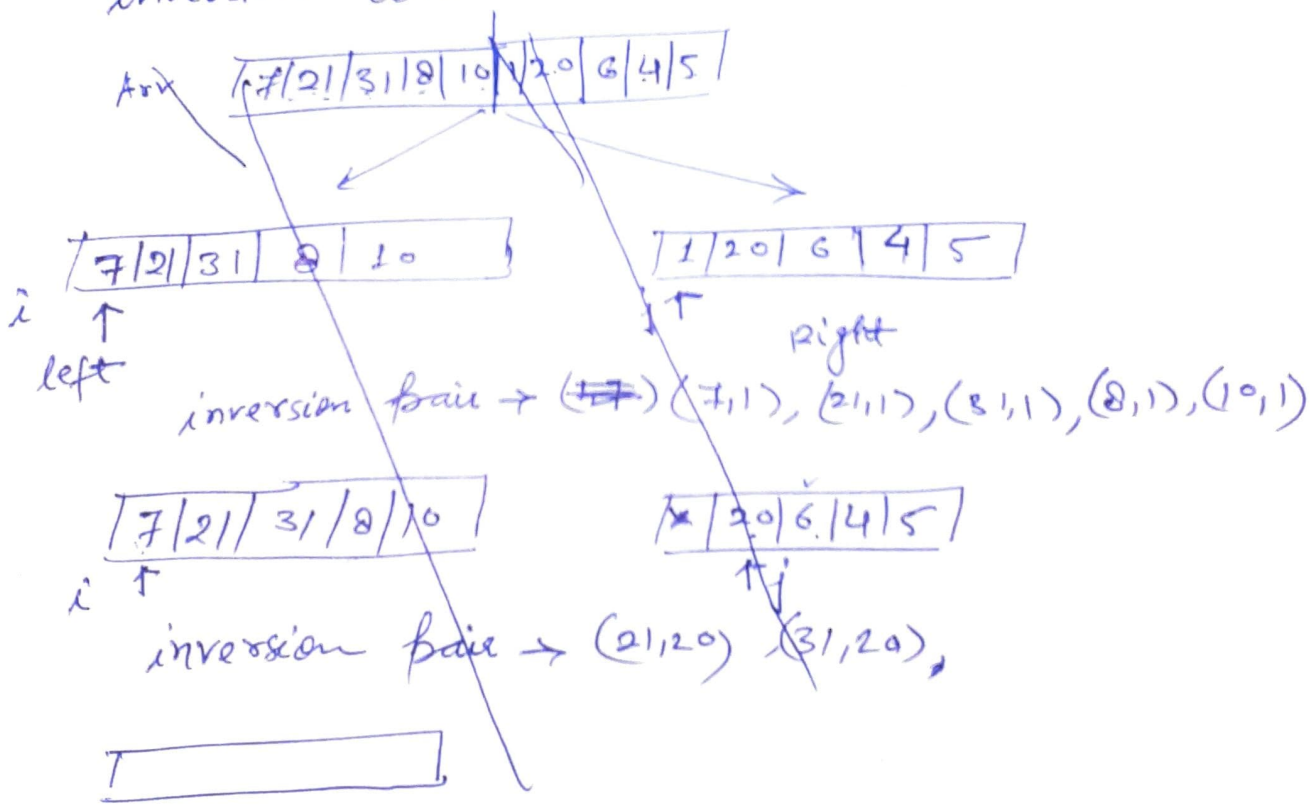
Q8. Quick sort is best for practical uses because:-

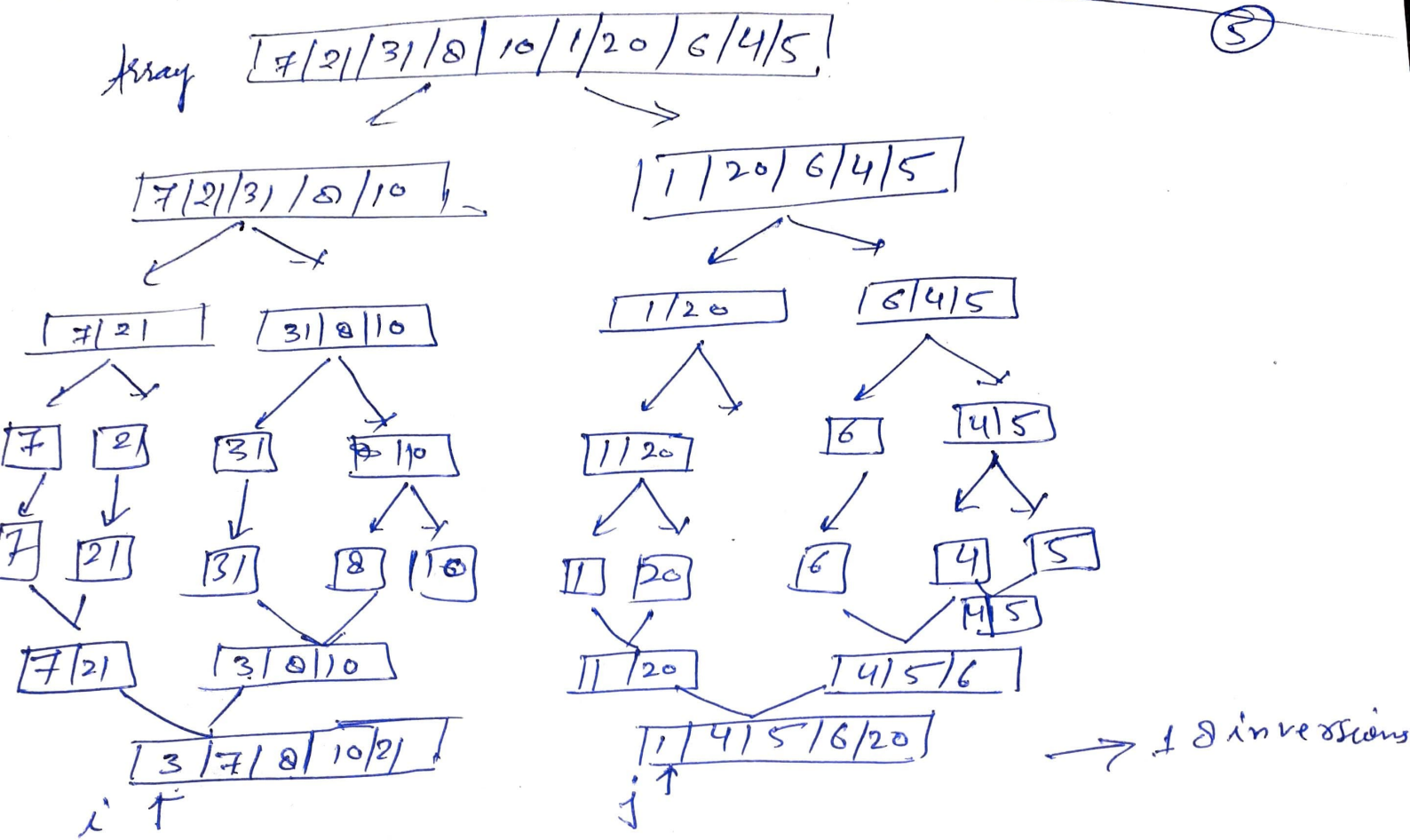
i) The sorting algorithm is used for information searching and as Quicksort is the fastest algorithm so it is widely used as a better way of searching.

ii) It is used everywhere where a stable sort is not needed.

iii) Quicksort is a cache-friendly algorithm as it has a good locality of reference when used for arrays.

Q9 Inversion count for an array indicates - how far (or close) the array is from being sorted. If the array is already sorted, then the inversion count is 0, but if the array is sorted in reverse order, the inversion count is maximum.





inversion pair → (3,1), (7,1), (8,1), (10,1), (21,1)

3 7 8 10 21 | 1 4 5 6 20

inversion pair → (7,4), (8,4), (10,4), (21,4)

x 7 8 10 21 | x x 5 6 20

inversion pair → (7,5), (8,5), (10,5), (21,5)

x 7 8 10 21 | x x x 6 20

inversion pair → (7,6), (8,6), (10,6), (21,6)

x 7 8 10 21 | x x x 20

inversion pair → (21,20)

Q10

The best case occurs when the partition process always picks the middle element as pivot

When the array is reverse sorted or already sorted Quick sort becomes worst.

Q11

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

$a = 2 \quad , b = 2 \quad , k = 1$

$\log_2 2 = 1 = 1$

$\therefore \quad T.C \Rightarrow \Theta(n \log n)$

} Merge sort

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

$a = 2 \, , b = 2 \, , k = 1$

$\therefore \quad T.C = \Theta(n \log n)$

} Quick sort

	Quicksort	Merge sort
Similarities →		
Best case T.C →	$O(n \log n)$	$O(n \log n)$
Avg " " →	$O(n \log n)$	$O(n \log n)$
Space complexity →	$O(n)$	$O(n)$
	Inplace	Inplace
Difference		
Worst case T.C →	$O(n^2)$	$O(n \log n)$
	not stable	stable

Q12 Stable selection sort

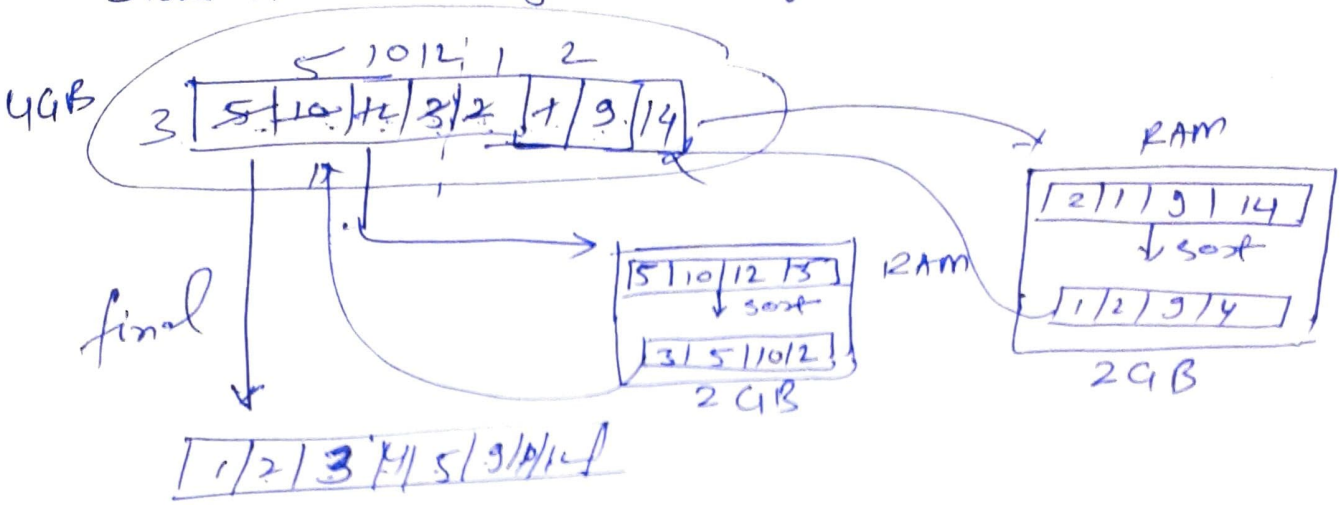
```
void stableselectionsort (int a[], int n)
{   for(i=0; i<n; i++)
    {
        int min = i;
        for(j=i+1; j<n; j++)
            if (a[min] > a[j])
                min = j;

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

Q13

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

Q14 We use merge sort, we first take half of array i.e 2 GB of array size in our RAM and performs merge sort and we put it back to the original array then we take rest of half of array on the RAM and sort other part of array and put it back to the original array. (8)



External Sorting → It handles massive amounts of data. It is used when data being sorted do not fit into the main memory.

Internal sorting → In this data sorting process that takes place entirely within the main memory of a computer.