# Tutorial-3

**Ques 1.** Write a linear search pseudo code to search 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-2 to 0; i--)
          if( A[i] == t )
            { return i; }
      else
          for (i=0 to n-1; i++)
              if ( A[i] == t)
                  return i;
}
```

**Ques 2.** Write pseudo code for iterative and recursive insertion sort. Insertion sort is called online sorting. why? what about other sorting algorithms?

**Ans.** 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 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.

other sorting algorithms like bubble-sort, insertion sort, heap sort etc are considered external sorting technique as they need the data to be sorted in advance.

Ques 3. Complexity of all the sorting algorithms.

Ans:

| Sorting | 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 | $O(n \log n)$ | $O(n \log n)$ |

Ques 4. Divide all sorting algorithms into inplace/ stable/ online sorting.

Ans.

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

Ques 5. Write recursive / iterative pseudo code for binary search. What is the Time and space complexity of Linear and Binary search (Recursive and Iterative)

Ans- 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 : $O(n \log n)$
space complexity : $O(1)$

**Recursive:** Time complexity : $O(n \log n)$
space complexity : $O(\log n)$

**Ques6.** Write ~~recurrence~~ recurrence relation for binary recursive search.

**Ans.**

$T(n)$
↓
$T(n/2)$
↓
$T(n/4)$
⋮
$T(n/2^k)$

Recurrence Relation =

$T(n/2) + (0|1)$.

Ques7. find two indexes such that A[i]+A[j]=k in minimum time complexity.

Ans-

```
int n;
int A[n];
int key;
int i=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)

Ques8. Which sorting is best for practical use? Explain.

Ans. Quicksort is the fastest general-purpose sort. It is one of the most efficient sorting algorithm. In most practical situations, quicksort is the method of choice.

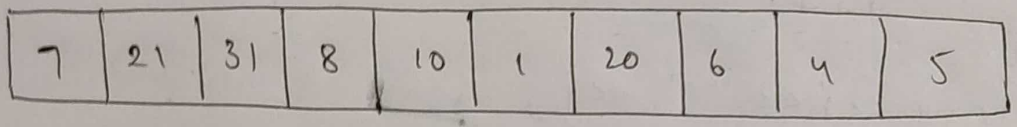
If stability is important and space is available, merge sort might be best.

Ques9. 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.

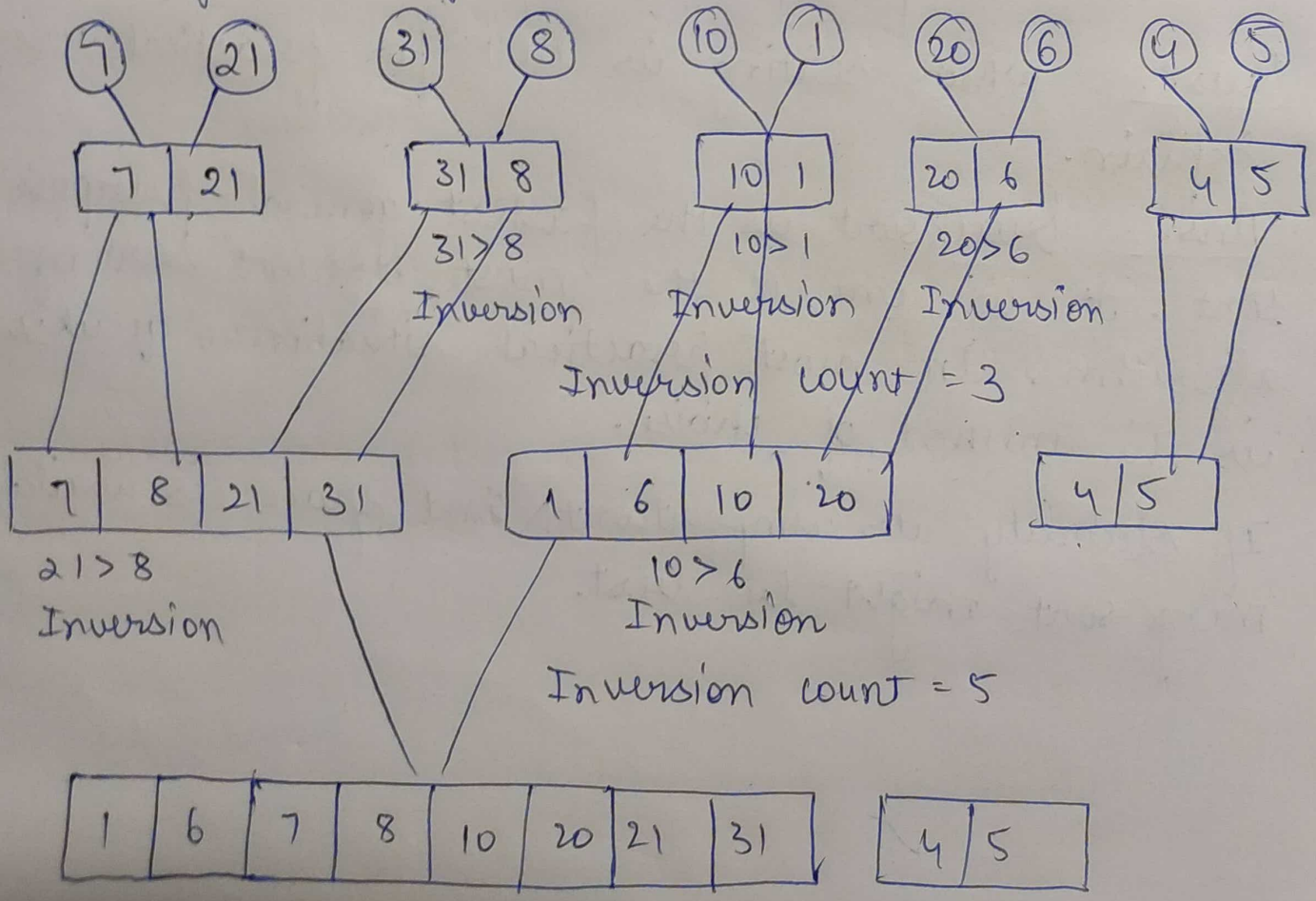
Ans. Inversion count for an array indicates - how far (or close) the array is from being sorted.
If the array is already sorted, then the inversion count is 0, but if the array is sorted in reverse order, the inversion count is the maximum.

Condition for inversion

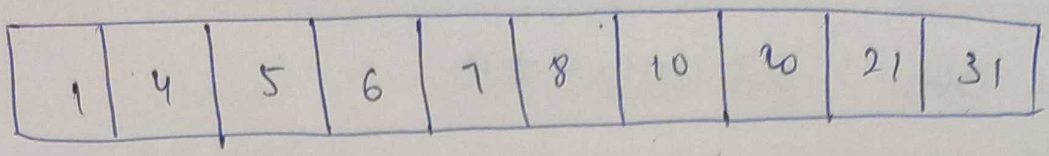
$a[i] > a[j]$ & $i < j$




Dividing the array:




Total inversions = 12




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 Inversions = 14

Inversion Count = 31.

Ques 10. In which cases Quick sort will give the best and the worst case time complexity?

Ans. Best case:

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

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 asending or descending order.

Ques 11. Write Recurrence Relation of Merge and Quick sort in best and worst case? What are the similarities and differences between complexities of two algorithms and why?

Ans. Best Case

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) Array is divided into two halves. | 1) Array is divided in any ratio. |
| 2) Worst case complexity is $O(n \log n)$. | 2) Worst case complexity is $O(n^2)$. |
| 3) Requires extra space i.e. not inplace. | 3) does not require extra space i.e. inplace. |
| 4) It is external sorting algorithm and it is stable. | 4) It is internal sorting algorithm and not stable. |
| 5) Works consistently on any size of data set. | 5) Works fast on small dataset. |

Ques 12. Selection sort is not stable by default. write a version of stable selection sort.

Ans.
```
void selection ( int A[], int n)
{
 for (int i=0; i<n-1; i++)
  {
   int num = 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;
  }
}
```

Ques 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.

Ans.
```
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;
}
```

Ques 14. Your computer has a RAM (Physical memory) of 2 GB and you are given an array of 4 GB for sorting which algorithm you are going to use for this purpose and why? Also Explain the concept of External and Internal Sorting.

Ans - When the data set is large enough to fit inside RAM, we ought to use merge sort, because it works on divide and conquer approach in which it keeps dividing the array into smaller parts until it can no longer be splitted. it then merge the array 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 amount of data. It is required when the data doesn't fit inside the RAM & instead they must reside in the slower external memory.

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

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

## Internal sorting

It 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 for external memory for program execution. It is used when input is small.

eg. Insertion sort, Quick sort, heap sort etc.