SAGAR NEGI SEC-H ROLL NO-10

Tutorial 3

Que1: Write linear search pseudocode to search an element in a sorted array with minimum comparisons.

Sol:-
```
while (l<=h)
{
  mid = (l+h)/2;
  if (arr[mid]==key)
  return true;
  else if (arr[mid]>key)
    h=mid-1;
  else
    l=mid+1;
  }
  return false;
```

Que2: Write pseudo code for iterative and recursive insertion sort. Insertion sort is called online sort. why? what about other sorting algorithms that has been discussed in lectures.

Sol:- Iterative insertion sort

```
for (int i=1;i<=n;i++)
{ j=i-1;
  x=A[i];
while (j>1 && A[j]>n)
  {
  A[j+1]=A[j];
     j--;
  }
  A[j+1]=n;
  }
```

Recursive insertion sort

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

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

Sol :- Bubble sort → $O(n^2)$
Insertion sort → $O(n^2)$
Selection sort → $O(n^2)$
Merge sort → $O(n \log n)$
Quick sort → $O(n \log n)$
Count sort → $O(n)$
Bucket sort → $O(n)$

Que 4. Divide all the sorting algorithms into inplace / stable / online sorting.

Sol :- Online sorting → Insertion sort
Stable sorting → Merge sort, Insertion sort, Bubble sort
Inplace sorting → Bubble sort, Insertion sort, Selection sort

Que 5. Write recursive / iterative pseudo code for binary search. What is the time and space complexity of linear and binary search (Recursive and Iterative).

Sol :- Iterative Binary Search :

```
while ( l <= h)
{
int mid = ( l+h)/2;
if (arr[mid] == key)
  return true;
else if (arr[mid] > key)
  h = mid -1;
else
 l = mid +1;
}
```

$O(\log n)$

Recursive Binary search :

O(log n)

```
while ( l<=h)
{
 int mid = ( l+h)/2;
 if ( arr[mid] == key)
   return true;
else if ( arr[mid] > key)
Binary_Search ( arr, low, mid -1);
else
Binary_Search ( arr, mid+1, high);
}
return false;
```

Que 6. Write recurrence relation for binary recursive search

Sol :- $T(n) = T(\frac{n}{2}) + T(\frac{n}{2}) + c$

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

```
Sol :- map < int , int > m;
   for (int i=0; i< arr.size(); i++)
    {
    if ( m.find (target - arr[i]) = m.end())
      m[ arr[i]] = i;
    else
     {
      cout << i << " " << mp [arr[i]];
     }
    }
```

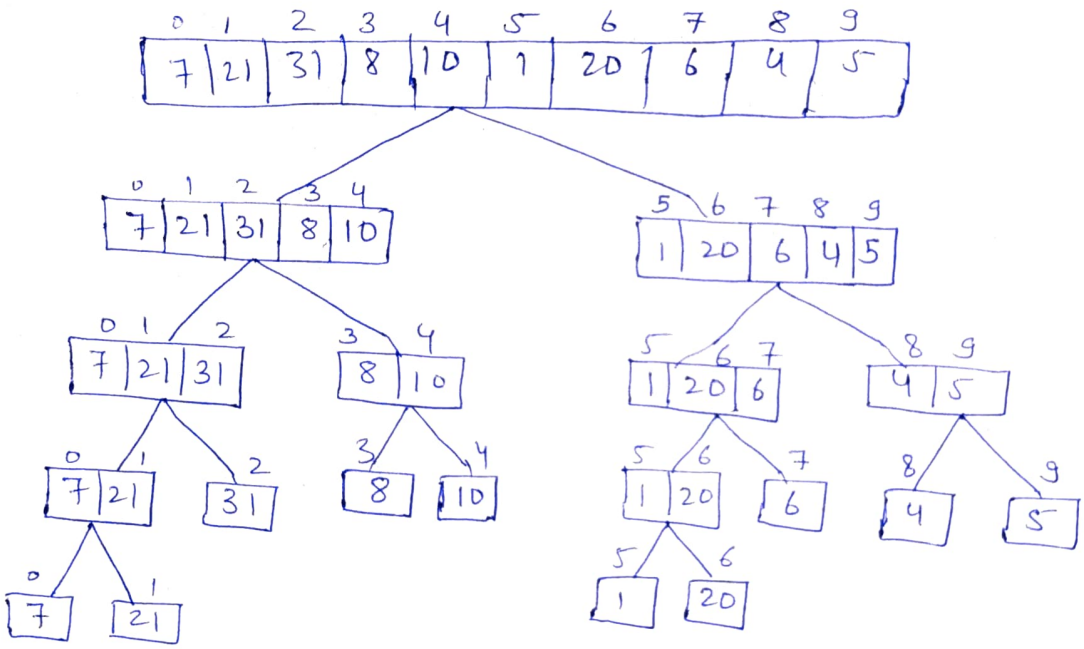
Que 8. Which sorting is best for practical uses? Explain.

Sol:- Quick sort is best for practical uses. It is the fastest general purpose sort.

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

Que 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.

Sol:- Inversion indicates how far as close the array is from being sorted.



Total no. of Inversions = 31

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

Sol:- Best case: The ~~worst~~ best case occurs when pivot element is the middle element. $TC = O(n \log n)$

Worst case: The worst case occurs when the pivot is always an extreme (smallest or largest) element.

$TC = O(n^2)$

Que 11. Write Recurrence Relation of Merge and Quick sort in best and worst case? Write similarities and differences between complexities of two algorithm and why?

Sol:- Merge sort: $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

Quick sort: $T(n) = 2T\left(\frac{n}{2}\right) + n + 1$

Similarities:

(i) Merge sort and quick sort are both divide and conquer based algorithm.

→ In Quick sort, array is divided into two parts until it is not possible to divide it fisther.

→ In Merge sort, array is split into two sub-arrays until only one array is left.

Differences:

(i) In merge sort array is parted into just two parts (halves) whereas in quick sort there is no compulsion of dividing array into two equal parts.

(ii) Merge sort can work well with any type of dataset irrespective of its size, whereas quick sort cannot work with large datasets.

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

Sol:- Selection sort can be made stable if instead of swapping, the minimum element is placed in its position without swapping i.e; by placing the number in its position by pushing every element one step forward.

Eg →

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

Que 13. 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.

Sol:- We will use merge sort because we can divide the 4GB data into 4 packets of 1GB and sort them separately and combine them latter.

Internal sorting: All the data to sort is stored in memory at all times while sorting is in progress.

External sorting: All the data is stored outside memory and only loaded into memory in small chunks.