Name: Nishtha Jain
Section: H
Roll No.: 39

Design And Analysis Of Algorithms (Tutorial-3)

Q1. Write linear Search pseudocode to search an element in a sorted array with minimum comparisons.

⇒
```
int linearSearch (int arr[], int n, int key)
{
    for (int i=0; i<n; i++)
    {
        if (arr[i]== key)
            return i;
    }
    return -1;
}
```

Q2. Write pseudocode for iterative and recursive insertion sort. Insertion sort is called online sorting. Why? What about other sorting algorithms that has been discussed in lectures?

⇒ Iterative Insertion sort

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

Recursive Insertion Sort

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

Insertion sort is also called online sort because it does not need to know anything about what values it will sort and the information is requested while the algorithm is running.

Q3 Complexity of all sorting algorithms that has been discussed in lectures.

⇒ i) Bubble Sort

Time complexity ⇒ Best case - $O(n^2)$
Worst case - $O(n^2)$

Space Complexity ⇒ $O(1)$

ii) Selection Sort

Time complexity ⇒ Best case - $O(n^2)$
Worst case - $O(n^2)$

Space complexity ⇒ $O(1)$

iii) Insertion Sort

Time complexity ⇒ Best case - $O(n \log n)$
Worst case - $O(n^2)$

Space complexity ⇒ $O(1)$

iv) Merge Sort

Time Complexity => Best case - O(n log n)
Worst case - O(n log n)

Space complexity => O(n)

v) Quick Sort

Time complexity => Best case - O(n log n)
Worst case - O(n²)

Space complexity => O(n)

vi) Heap Sort

Time complexity => Best case - O(n log n)
Worst case - O(n log n)

Space complexity => O(1)

Q4. Divide all sorting algorithms into inplace/stable/online sorting.

=>

Sorting	Inplace	Stable	Online
* Bubble	✓	✓	
* Selection	✓		
* Insertion	✓		✓
* Merge		✓	
* Quick	✓	✓	
* Heap	✓		

Q5. Write recursive/iterative pseudo code for binary search. What is the time and space complexity of linear and binary search. (Iterative and recursive).

=> Iterative Binary Search

```
int binarysearch (int arr, int l, int r, int key)
{
    while (l <= r)
    {   int m = (l+r)/2;
```

```
        if (arr[m] == key)
            return m;
        if (arr[m] < key)
            l = m+1;
        else
            r = m-1;
    }
    return -1
}
```

Time Complexity :) Best case - O(1)
Worst case - O(n log n)

Recursive Binary Search

```
int binarySearch (int arr[], int l, int r, int key)
{
    if (r >= l)
    {
        int m = (l+r)/2;
        if (arr[m] == key)
            return m;
        else if (arr[m] > key)
            return binarysearch (arr, l, mid-1, key);
        else
            return binary search (arr, mid+1, r, key);
    }
    return -1;
}
```

Time Complexity :) Best case - O(1)
Worst case - O(log n)

Linear Search

Time Complexity :) Best Case - O(1)
Worst Case - O(n)

Q6 Write recurrence relation for binary recursive search

⇒ Recurrence relation for Binary Recursive Search

⇒ $T(n) = T(n/2) + 1$

Q7 Find two indexes such that A[i] + A[j] = K in minimum time complexity.

⇒

```
map<int, int> m;
for (int i=0; i
```

```
int findsum(int arr[], int n, int k)
{
    Sort (A, n);
    for (i=0 to n-1)
    {
        x = binarySearch (arr, 0, n-1, k-arr[i])
        if (x)
        return 1;
    }
    return -1;
}
```

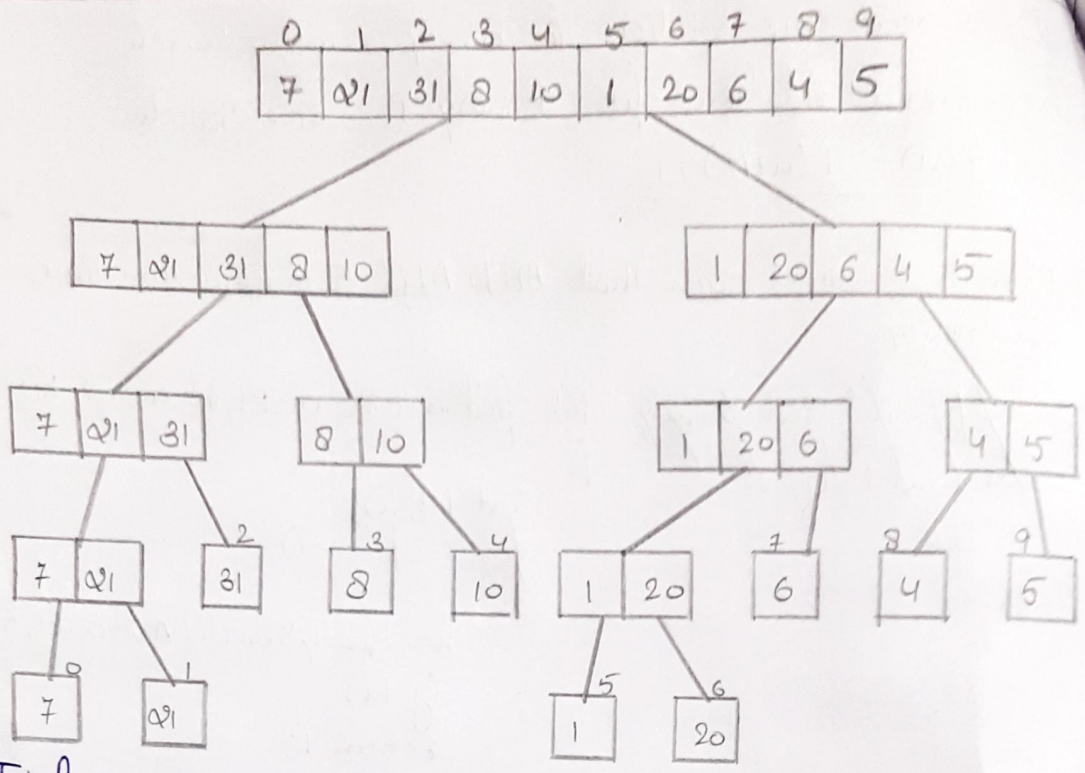
Q8 Which sorting is best for practical uses? Explain.

⇒ Quick sort is the fastest general-purpose sort. In most practical situations quicksort is the method of choice. If stability is important and space is available, merge sort might be best.

Q9 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.

⇒ Inversion count for an array indicate - 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 the reverse order, the inversion count is maximum.

arr[] = {7, 21, 31, 10, 8, 1, 20, 6, 4, 5}



Total inversion count = 31

Q10 In which cases Quick sort will give the best and worst case time complexity?

⇒ Worst time complexity of quick sort is $O(n^2)$. The worst case occurs when the picked pivot is always an extreme (smallest or largest) element. This happens when input array is sorted or reverse sorted and either first or last element is picked as pivot.

The best case complexity of quick sort is when we will select pivot as a mean element.

Q11. 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?

⇒ Recurrence Relation

Merge Sort ⇒ $T(n) = 2T(n/2) + n$

Quick Sort ⇒ Best case = $T(n) = 2T(n/2) + n-1$

Worst case = $T(n) = T(n-1) + N-1$

* Merge Sort is more efficient and works faster than quick sort in case of large array size or datasets.

* Worst case complexity for quick sort is $O(n^2)$ whose for merge sort is $O(n(\log n))$.

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

⇒ Stable Selection Sort

```
void stableselection(int arr[], int n)
{
    for(int i=0; i<n-1; i++)
    {
        int min=i;
        for(int j=i+1; j<n; j++)
        {
            if(arr[min]>arr[j])
                min=j;
        }
        int key=arr[min];
        while(min>i)
        {
            arr[min]=arr[min-1];
            min--;
        }
        arr[i]=key;
    }
}
int main()
{
    int arr[]={4,5,3,2,4,1};
    int n=sizeof(arr)/sizeof(arr[0]);
    stableselection(arr,n);
    for(int i=0; i<n; i++)
    cout<<arr[i]<<" ";
    cout<<endl;
    return 0;
}
```