Design and Analysis of Algorithms

Tutorial-3

Date: 7-4-22

Q.1. Write linear search pseudocode to search an element in a sorted array with minimum comparisons.

Ans.1.

```
for( i=0 to n)
{
    if ( arr[i] == value)
        // element is found.
}
```

Q.2. 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?

Ans.

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

Jatin

Date: / /

```
for(i=1 to n)        //iterative
{
    key ← A[i]
     j ← i-1
    while( j>=0 and A[j] > key)
    {
        A[j+1] = A[j]
        j ← j-1
    }
    A[j+1] ← key
}
```

Insertion sort is known as online sorting because it doesn't know the whole i/p, more i/p can be inserted while the insertion sorting runs.

Q.3. Complexity of all the sorting algorithms discussed in lectures.

Sorting Type	Best case	Worst Case	Avg. Case
1 Selection Sorting	$O(n^2)$	$O(n^2)$	$\theta(n^2)$
2 Bubble Sorting	$O(n)$	$O(n^2)$	$O(n^2)$
3 Insertion Sorting	$O(n)$	$O(n^2)$	$O(n^2)$
4 Heap Sorting	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
5 Quick Sorting	$O(n \log n)$	$O(n^2)$	$O(n \log n)$
6 Merge Sorting	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Jatin

Q.4. Ans.4.

Inplace Sorting	Stable Sorting	Online Sorting
① Bubble Sort	① Merge Sort	① Insertion Sorting
② Selection Sort	② Bubble "	
③ Insertion Sort	③ Insertion "	
④ Heap Sort	④ Count "	
⑤ Quick Sort		

Ans.5

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

```
int binary (int arr[], int l, int r, int x)
{                                            //iterative
    while (l <= r)
    {
        int m = l + (r - l) / 2;
        if (arr[m] == x)
            return m;
```

Jatin

```
else if (arr[mid] > x)
        r = m-1;
    else
        l = m+1;
}
return -1;
}
```

Time complexity of:

Binary Search - O(Log n)

Linear Search - O(n)

Ans. 6. Recurrence relation for binary recursive search:

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

T(n) = time reqd. for binary search in an array of size 'n'.

Ans. 7

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

Time Complexity = $O(n \log n) + n \cdot O(\log n)$

$= O(n \log n)$

Jatin

Ans. 8. • Quick sort is the fastest general purpose sort.
In most practical situations, quick sort is the method of choice. If stability is important and space is available, merge sort might be best

Ans. 9. A pair (a[i], a[j]) is said to be inversion if:
a[i] > a[j].
In arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, total no. of inversion are 31, 5}.
using merge sort.

A.10. The worst case time complexity of quick sort is $O(n^2)$. This case occurs when the picked pivot is always on extreme (smallest or largest) element. This happens when i/p array is sorted or reverse sorted.
The best case of quick sort is when we'll select pivot as a mean element.

A.11. Recurrence relation of:
Merge Sort → $T(n) = 2T(n/2) + n$
Quick sort → $T(n) = 2T(n/2) + n$.

Jatin

• Merge sort is more efficient and works faster than quick sort in case of larger array size or datasets.

- Worst case complexity for quick sort is $O(n^2)$, whereas $O(n \log n)$ for merge sort.

Ans. 12. 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;
    }
}
```

Ans. 13. Modified Bubble Sorting

Jatin

```
void bubbleSort(int A[], int n)
{
    for(int i=0; i<n; i++)
    {
        int swaps=0;
        for (int j=0; j<n-1-i; j++)
        {
            if (a[j] > A[j+1])
            {
                int t = a[i];
                a[j] = A[j+1];
                A[j+1] = t;
                swaps++;
            }
        }
        if (swaps==0)
            break;
    }
}
```

Jatin