Design and Analysis of Algorithms

Sharad Gurnbeer
GEU CST
Roll no → 12

Tutorial - 3

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

Ans Pseudocode for linear search :-

```
for (i=0 to n)
    {
        if (arr[i] == value)
            {
            }
    }
```

Q.2 Write pseudo code 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 Iterative :-

```
InsertionSort ( int a[], int n)
  {
    int i, j, k;
  for (i=1; i<n; i++)
     {
        k = a[i]
        j = i-1;

  while (j >= i && a[j] > k)
     {
     a[j+1] = a[j];
```

```
            j--;
          }
        a[j+1] = k;
      }
    }
```

Recursive :-

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

=> Insertion sort is called online sorting because it considers only one input per iteration and produces a partial solution without considering future elements whereas other sorting algorithms process the whole problem data ~~algorithm~~ altogether from the beginning and is required to output an answer which solve the problem at hand.

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

Ans (a) :- Selection sort :- Best → $O(n^2)$
Worst → $O(n^2)$
Average → $O(n^2)$

(b) Bubble sort :- Best → $O(n)$
Worst → $O(n^2)$
Average → $O(n^2)$

(c) Insertion sort :- Best → $O(n)$
Worst → $O(n^2)$
Average → $O(n^2)$

(d) Heap sort :- Best → $O(n \log(n))$
Worst → $O(n \log(n))$
Average → $O(n \log(n))$

(e) Quick sort :- Best → $O(n \log(n))$
Worst → $O(n^2)$
Average → $O(n \log(n))$

(f) Merge sort :- Best → $O(n \log(n))$
Worst → $O(n \log(n))$
Average → $O(n \log(n))$

Q.4 Divide all the sorting algorithms into inplace / Stable / Online Sorting.

Sol Inplace Sorting :- Bubble, Selection, Insertion, Quick Sort, Heap sort

Stable Sorting :- Merge sort, Bubble Sort, Insertion, Count sort

Online Sorting :- Insertion

Q.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 Recursive Binary Search :-

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

```
else
return -1;
    }
```

Iterative Binary Search :-

```
int binarySearch (int arr[], int l, int r, int x)
{
    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;
      }
}
```

Time Complexity of Binary search → $O(\log n)$

Time Complexity of linear search → $O(n)$

Q.6 Write recurrence relation for binary recursive Search.

Ans

$$T(n) = T(n/2) + 1 \quad \text{---} (1)$$

Put $n = n/2$ in eq. (1)

$$T(n/2) = T(n/4) + 1$$

$$T(n) = T(n/4) + 2 \quad \text{---} (2)$$

Put $n = n/4$

$$T(n/4) = T(n/8) + 1$$

$$T(n) = T(n/8) + 2 + 1 \quad \text{---} (3)$$

$$\vdots$$

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

$$2^k = n$$

$$k = \log n$$

$$T(n) = T(n/n) + \log n$$

$$T(n) = T[1] + \log n$$

$$T(n) = O(\log n)$$

Q.7 Find two indexes such that A[i] + A[j] = K in minimum time complexity.

Sol

```
findIndex ( int a[], int n, int K)
{
  int j = 0, i = 1;
while ( j < n && i < n)
{
if ( j != i && (a[i] - a[j] == K ||
      a[j] - a[i] == K))
      printf i, j;
else if ( a[i] - a[j] < K)
      i++;
  else j++;
}
}
```

Q.8 Which sorting is best for practical users? Explain

Sol 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.

Q.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 Inversions in an array basically defines how far or close an array is from being sorted. If array is already sorted, Inversion count → 0; if array is in reverse order inversion count → maximum.

Total inversion for above example will be 31.

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

Sol Worst case in Quick sort :- The worst case time complexity of a quick sort is $O(n^2)$ if the picked pivot element is always an extreme (smallest or largest element).
Or the give array is sorted and we pick either first or last element.

Best case in Quick sort :- The best case is $O(n \log(n))$ when we will select pivot element as a mean element.

Q-11 write Recurrence Relation of Merge sort and Quick sort in best and worst case? what are the similarities and differences between complexities of two algorithms and why?

Sol Quick sort :-

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

worst : $T(n) = T(n-1) + n$

Merge Sort :-

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

* In merge sort, the array is divided into two equal halves.

$\therefore T.C = O(n \log n)$

* In quick sort, the array is divided into any ratio depending on the position of pivot element.

Time comp. ranges from $O(n^2)$ to $O(n \log n)$.

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

Sol

```
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];
```

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

Q-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.

Sol A better version of bubble sort, known as modified bubble sort, includes a flag that is set if an exchange is made after an entire pass over the array. If no exchange is made, then it should be clear that the array is already order because no two elements needed to be sorted.

```
Void bubble ( int a[], int n)
    {
   for (int i=0; i<n; i++)
       {
         int swaps = 0;
     for (int j = 0; j < n - i - 1; 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;
  }
}
```