Tutorial 3 — Nikul Singh I, 20

Ques 1. Write linear search psuedocode to search an element in a sorted array with minimum comparisions.

```
Void linearSearch (int A[], int n, int key)
{ int flag = 0;
  for (int i=0; i<n; i++)
  { if (A[i] == key)
    { flag = 1;
      break;
    }
  }
  if (flag == 0)
      cout << "Not found";
  else
      cout << "Found";
}
```

Ques 2. - Write pseudo code for iterative and recursive insertion sort. Insertion sort is called Online sorting. why?

Iterative

```
for (i=1 to n-1)
{   t = A[i], j = i-1
    while (j>=0 && A[j]>t)
    {   A[j+1] = A[j]
        j = j-1;
    }
    A[j+1] = t;
}
```

Recursive

```
void insertionsort (int arr[], int n)
{ if (n<=1)
      return;
  insertionSort (arr, n-1);
  int 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 called Online Sorting because insertion sort considers one input element per iteration and produces a partial solution without considering future elements.

But other sorting algorithm requires access to the entire input, they considered as offline algorithms.

Ques3. Complexity of all sorting algorithm that has been discussed in lectures.

Algorithm	Time Complexity		
	Best	Average	Worst
① BubbleSort	$O(n)$	$O(n^2)$	$O(n^2)$
② SelectionSort	$O(n^2)$	$O(n^2)$	$O(n^2)$
③ Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
④ CountSort	$O(n+k)$	$O(n+k)$	$O(n+k)$
⑤ QuickSort	$O(n\log n)$	$O(n\log n)$	$O(n^2)$
⑥ MergeSort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
⑦ HeapSort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$

Ques4. Divide all sorting algorithms into inplace, stable, online.

Algorithm	Inplace	Stable	Online.
① BubbleSort	✓	✓	×
② SelectionSort	✓	×	×
③ InsertionSort	✓	✓	✓
④ CountSort	×	✓	×
⑤ MergeSort	×	✓	×
⑥ QuickSort	✓	×	×
⑦ HeapSort	✓	×	×

Ques → Write Recursive / iterative pseudocode for binary search. What is the time and space complexity of linear and Binary Search

Recursive ⇒

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

Iterative ⇒

```
int binarysearch(int arr[], int l, int r, int key)
{
    while(l <= r)
    {
        int m = l + (r - l)/2;
        if (arr[m] == key)
            return m;
        if (arr[m] < key)
            l = m+1;
        else
            r = m-1;
    }
    return -1;
}
```

Algorithm	T.C		S.C	
	Recursive	Iterative	Recursive	Iterative
Linear Search	O(n)	O(n)	O(1)	O(1)
Binary Search	O(logn)	O(logn)	O(logn)	O(1)

Ques 6. Write Recurrence relation for binary Search

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

Q7. Find two indices such that $A[i] + A[j] = k$ in minimum time Complexity

```
Void Sum(int A[], int k, int n)
{ Sort(A, A+n)
  int i=0; j=n-1;
  while(i<j)
  { if (A[i] + A[j] == k)
      break;
    else if (A[i] + A[j] > k)
      j--;
    else
      i++;
  }
  print(i,j);
}
```

Here sort func. has $O(n \log n)$ T.C and for while loop it is $O(n)$
∴ overall T.C = $O(n \log n)$

Q8 which sorting is best for practical uses? Explain.
for practical uses, we mostly prefer merge sort, because of its stability and it would be best for very large data. Futher, time complexity of mergesort is same in all cases, that is $O(n \log n)$.

Q10 In which case Quick Sort will be give the best and worst time complexity
when the array is already sorted in reverse order, quick sort gives the worst case time complexity i.e. $O(n^2)$, but when the array is totally unsorted, it will give the best case time complexity i.e $O(n \log n)$.

Q11 write Recurrence relation of Merge and Quick sort in best and worst case, what are the similarity and differences b/w complexities of both algo and why?

	Best	Worst
Quick Sort	$T(n) = 2T(n/2) + n$	$T(n) = T(n-1) + n$
Merge Sort	$T(n) = 2T(n/2) + n$	$T(n) = 2T(n/2) + n$

Both algorithm are based on the divide and conquer algorithm, both the algorithm have same time complexity in the best and average case.