Manas Mittal
Sec: CE, Roll No. : 10

Date / /

DAA Assignment - 2

Q-1)
```
int n, k;
int i = 0;
while(i < n){

    if (arr[i] == k){
        Print("Found %d ", i);
        break;
    }
    else if (arr[i] > k){
        Print("Not found ");
        break;
    }
    else{
        i++;
    }
}
```

Q-2)
```
int n;
for(int i=1 ; i<n ; i++){
    int t = i;
    for(int j = i-1 ; j >= 0 ; j--){
        if (arr[j] > arr[t]){

            swap(arr[j], arr[t]);
            t = t-1;
        }
    }
}
```

Date / /

Recursive :

```
void insertionsort(int arr[], int n){

        if (n <= 1){
            return;
        }
        insertionsort(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;
    }
```

Insertion sort considers one input element per iteration and produces a partial solution without considering future elements. Thus, insertion sort is an example of online sorting.

(Q-3) Next Page

Date / /

Algorithm	Complexity		
	Best	Avg.	Worst
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Merge	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
Heap sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
Quick sort	$O(n\log n)$	$O(n\log n)$	$O(n^2)$
Count sort	$O(n+k)$	$O(n+k)$	$O(n+k)$

Q-4)

Algorithm	inplace	stable	online
Bubble	1	1	0
insertion	1	1	1
Selection	1	0	0
Merge	0	1	0
Heap	1	0	0
Quick Sort	1	0	0
Count sort	0	1	0

Q-5) Iterative:

```
int n, k;
int s=0, e=n-1;
int mid = (s+e)/2;
```

Date / /

```
while (s<=e){
    if(arr[mid]==k){
        Print("Found %d",
                        mid);
        break;
    }
    else if (arr[mid]>k)
    {
        e=mid-1;
    }
    else{
        s=mid+1;
    }
    mid=(s+e)/2;
}
```

Algorithm	Time Comp.	Space Comp.
Linear Search	O(n)	O(1)
Binary Search	O(log n)	O(1)

6) $T(n) = T(n/2)+k$, where k is constant

Date / /

7)
```
int n; sort(A);
int s = 0, e = n-1;
while (s < e){
      if (A[s] + A[e] == k)
      {
            print("%d, %d", s, e);
            break;
      }
      else if (A[s]+A[e] > k){
            e = e-1;
      }
      else {
            s = s+1;
      }
}
```

8) Quick sort is one of the most efficient sorting algorithms. It works on dividing the array into smaller ones and swapping the smaller ones depending on the pivot element picked.

It is prefferred most because sorting n objects will take nlogn time. It is an inplace sorting algo which means it do not takes extra space. Its inner loop is relatively short.

Date / /

9) The no. of inversions in an array indicates how close or far the array is, from being completely sorted. If the array is already sorted then inversion count is 0, if array is reversly sorted then inversion count is max.

10) In case of Best case complexity, the best case occurs when the pivot element is the middle element or near to the middle element. The best case time complexity of quicksort is $O(n \log n)$.

In quick sort, worst case occurs when the pivot element is either greatest or smallest element. Suppose, if the pivot element is always the last element of the array, the worst case would occur when the given array is already in ascending or descending order. The worst-case time complexity of quicksort is $O(n^2)$.

11) Best Case | Next Page

Merge Sort:

Date / /

Merge Sort recurrence relation:

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

Quick Sort recurrence relation:

$T(n) = T(n-1) + T(0) + O(n)$

12)
```
# include <stdio.h>
int main()
{
        int a[100];
        int n;
        scanf("%d", &n);
        for(int i=0; i<n; i++){
                scanf("%d", &a[i]);
        }
        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];
                for(int k=min; k>i; k--){
                        a[k] = a[k-1];
                }
                a[i] = key;
        }
}
```

Date / /

13)
```
# include <stdio.h>
int main() {

        int a[100];
        int n;
        scanf("%d", &n);
        for(int i=0; i<n; i++){
             Scanf("%d", &n);
        }
        for(int i=1; i<n; i++){

             bool swapped = false;
             for(int j=0; j<n-i; j++){
                  if(arr[j]>arr[j+1]){
                       int temp = arr[j];
                       arr[j] = arr[j+1];
                       arr[j+1] = temp;
                       swapped = true;
                  }
             }
             if(swapped == false){
                   break;
             }
        }
   }
```

Date / /

14) We will use external sorting for this. We will divide our source file into temporary files of size equal to the size of RAM and first sort these files.

1. Divide the source in 2 temporary files each of size 2GB (i.e. equal to size of RAM).

2. Sort these temporary files one by one using the ram individually.

Internal Sorting: If the input data is such that it can be adjusted in the main memory at once, it is called internal sorting.

External sorting: If the input data is such that it cannot be adjusted in the memory entirely at once, it needs to be stored in a hard disk, floppy disk, or any other storage device.