Design and Analysis of Algorithms

Assignment-1

Solutions (Hints)

REMARK: The codes given below are just a guide, this is not the only way to write programs for these algorithms. What is important is that your program should implement the algorithm in the correct manner and as efficiently as possible.

1. Bubble sort

```
include<iostream>
using namespace std;
void swap(int *a,int *b)
{//have to use pointers and call by reference if using function
        int temp=*a;
        *a=*b;
        *b=temp;
}
void bubblesort(int arr[],int n)
{
        int i,j;
        bool swapped;// variable declared to optimize bubble sort, to prevent from running if  after a
particular iteration,it has already been sorted//
        for(i=0;i<n-1;i++)
        {
                swapped=false;
                for(j=0;j<n-i-1;j++)
                {
                        if(arr[j]>arr[j+1])
                        {
                        swap(&arr[j],&arr[j+1]);
                        swapped=true;
                        }
                }
                if(swapped==false)//if after a whole iteration, no swapping has occurred(bool swapped
is false),it means array is already sorted, no more swapping required.//
                break;
        }
}
```

2. Selection sort

```
#include<iostream>
using namespace std;
void swap(int *a,int *b)
{
        int temp=*a;
        *a=*b;
        *b=temp;
}
void selectionsort(int arr[],int n)
{
        int i,j,min,k;
        for(i=0;i<n-1;i++)
        {
                min=i;
                for(j=i+1;j<n;j++)
                {
                        if(arr[j]<arr[min])
                        min=j;
                        swap(&arr[min],&arr[i]);
                }
                for(k=0;k<n;k++)
                {
                        cout<<arr[k]<<" ";
                }
                cout<<"\n";


        }
}
```

3. Insertion Sort

```
#include<iostream>
using namespace std;
void insertionsort(int arr[],int n)
{
        int i,j,key,shift=0,comp=0;//shift and comp declared to count number of shifts and comparisons 
respectively//
        for(i=1;i<n;i++)
        {
                key=arr[i];
                j=i-1;
                while(j>=0 && arr[j]>key)
                {
                        comp++;
                        arr[j+1]=arr[j];
                        shift++;
                        j=j-1;
                }
                arr[j+1]=key;
                if(key!=arr[i])
                shift++;
                if(j!=-1)
                comp++;


        }
        cout<<"The shifts are "<<shift<<"\n";
        cout<<"The comparisons are "<<comp<<"\n";
}
```

Here shifts are counted whenever arr[j] is shifted to arr[j+1] and when key is shifted to its correct position. Shifting of key need not be counted when no shifting has been done inside the while loop. Hence outside while loop it is if(key!=arr[i]) shift++

Comparisons are done even if it does not enter while loop ,so an extra comp ++ outside while loop. But if value of j goes to -1 and while loop is exited, then there is no need for an extra comp++ outside while loop, so if(j!=-1) comp++

4. Recursive Insertion Sort

```
#include<iostream>
using namespace std;
void recursiveinsertionsort(int arr[],int n)
{
        int i,j,last;
        if(n<=1)
        return; //This is the base condition for recursion
        recursiveinsertionsort(arr,n-1); //recursive call
        last=arr[n-1];
        j=n-2;
        while(j>=0 && arr[j]>last)
        {
                arr[j+1]=arr[j];
                j--;
        }
        arr[j+1]=last;
}
```

**Initially function is called in main and then inside the function itself, it is recursively called till n=1.
The recursive calls are saved in the stack.
So call for n is saved first, then n-1, then n-2….till n=1.**

**When the program runs after n=1, initially last is 0 and j is -1, so while loop does not run.
In the next n=2, last is 1 and j is 0, now while loop runs and sorts the first two elements.**

**In the next level n=3, last =2, j=1, the first three elements are sorted
This continues till all elements are sorted.**

5. Segregate 0’s and 1’s

```
#include<iostream>
using namespace std;
void segregate(int arr[],int n)
{
        int left=0;
        int right=n-1;
        while(left<right)
        {
                while(arr[left]==0 && left<right)
                left++;
                while(arr[right]==1 && left<right)
                right--;
                if(left<right)
                {
                        arr[left]=0;
                        arr[right]=1;
                        left++;
                        right--;
                }
        }
}
```

Should be obvious from the coding itself. Use two variable to count zeros and ones whiles traversing array only once.