Experiment 1
Implementation Of Bubble Sort Algorithm

Date: 26-08-2020

Aim: To implement the bubble sort and it’s algorithm

Data Structure Used: Arrays

Operation Used: Comparisons and Swapping

Algorithm:

Input: Unsorted array of length n
Output: Sorted array of length n

```
Step 1  : Start
Step 2  : Receive the size of the array in a variable n
Step 3  : i←0                                          //Receive the elements in the array
Step 4  : Repeat Step 5 to 6 until i=n
Step 5  : Receive an element and store it in a[i]
Step 6  : i←i+1
Step 7  : i←0                                          //Beginning the bubble sort
Step 8  : Repeat steps 9 to Step 16 until I = n-1
Step 9  : j←0
Step 10 : Repeat steps 11 to step 15 until j=n-i-1
Step 11 : if arr[j]>arr[j+1] then do Step 12 to Step 14 else skip to Step 15
Step 12 : temp = arr[j]
Step 13 : arr[j] = arr[j+1]
Step 14 : arr[j+1] = temp
Step 15 : j=j+1
Step 16 : i+=i+1                                       //Bubble sort ends here
Step 17 : i←0                                          //Print the sorted array
Step 18 : Repeat Step 19 to 20 until i=n
Step 19 : Print the value of arr[i]
Step 20: i=i+1
Step 21 : Stop
```

Details of the Algorithm:
The Bubble sort algorithm takes a given array and keeps swapping the elements in such a way that the largest number (in case of ascending order) is guaranteed to come at the end of the loop on the first iteration of the i loop. This then divides the array into two parts, sorted and the unsorted, the sorted part is the one from n-i-1 to n-1 and the unsorted part is from 0 to (n-i-2)th element. In the second iteration of the I loop the largest element of the unsorted part gets moved to the n-i-1 th position (the previous (n-i-1)th position since the value of i is incremented by one). This continues until the value of i=n-1 in which case the value of the last element of the unsorted sub array becomes n-n+1-2 = -1 and the first position of the sorted sub-array becomes n-n+1-1=0 this proves that the whole array is sorted. The time complexity is $O(n^2)$, since the total number of comparison is n(n-1)/2

Result: the Program is successfully compiled and the desired output is obtained.

Program/ Source Code:

```
#include<stdio.h>

void main(){
    int comp=0,swaps=0,i,j;
    int arr[100];
    int n,temp;
    printf("Enter the number of elements in the arrat: ");
    scanf("%d",&n);
    printf("Enter the elements in the array : ");
    for(i=0;i<n;i++)
        scanf("%d%*c",arr+i);

    for(i=0;i<n-1;i++){
        for(j=0;j<n-i-1;j++){
            if(arr[j]>arr[j+1]){
                temp = arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
                swaps++;
            }
            comp++;
        }
    }
    printf("Sorted Array: ");
    for(i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
    printf("\nNumber of comparisons = %d\n",comp);
    printf("Number of swaps= %d\n",swaps);

}
```

Sample Input/Output

```
Sample Input 1:
5
23 43 56 78 91

Sample Output 1:
Sorted Array: 23 43 56 78 91
Number of comparisons = 10
Number of swaps= 0


Sample Input 2:
5
91 78 56 43 23

Sample Output 2:
Sorted Array: 23 43 56 78 91
Number of comparisons = 10
Number of swaps= 10
```

Sample Input 3:

```
5
78 43 56 91 23
```

Sample Output 3:

```
Sorted Array: 23 43 56 78 91
Number of comparisons = 10
Number of swaps= 6
```

Experiment 2
Implementation of Selection Sort Algorithm

Date: 26-08-2020

Aim: To implement the Selection sort and it's algorithm

Data Structure Used: Arrays

Operation Used: Comparisons and Swapping

Algorithm:

Input: An unsorted array of length n
Output: Sorted Array of length n

```
Step 1  : Start
Step 2  : Receive the size of the array in a variable n
Step 3  : i←0                                   //Receive the elements in the array
Step 4  : Repeat Step 5 and 6 until i=n
Step 5  : Receive an element and store it in a[i]
Step 6  : i←i+1
Step 7  : i←0                                   // Beginning of Sorting process
Step 8  : Repeat steps 9 to  20 until i=n
Step 9  : pos←i
Step 10:  smallest←arr[i]
Step 11: j←i
Step 12: Repeat Steps 13 to  16 until j=n
Step 13: if arr[j] < smallest then do Steps 14 to 15
Step 14: smallest ← arr[j]
Step 15:  pos← j
Step 16: j← j+1
Step 17: if pos != I then do steps 18 to 20
Step 18: temp ← arr[i]
Step 19: arr[i] ← arr[pos]
Step 20: arr[pos] ← temp
Step 21:  i ← i++                               //Selection Sort ends here
Step 22: i←0
Step 23: Repeat Step 22 to 23 until i=n         //Print the sorted array
Step 24: Print the value of arr[i]
Step 25: i=i+1
Step 26: Stop
```

Description of the Algorithm:

The selection sort as the name implies selects the smallest element (in case of ascending order) from the unsorted sub array, initially the unsorted sub array is from 0 to n-1 where n is the number of element in the array, and swaps it with the first element in the said sub-array making the array from 0 to the i-1 sorted and the remaining () unsorted. This process goes on until the value of i becomes n at which the starting and ending indices of the sorted array becomes 0 and i-1 which is equal to n-1. Hence we can say that the array is sorted. The time complexity is $O(n^2)$, since the total number of comparison is n(n-1)/2

Result: the Program is successfully compiled and the desired output is obtained.

Program/ Source Code:

```
#include<stdio.h>

void main(){
    int comp=0,swaps=0,i,j;
    int arr[100];
    int n,temp,smallest,pos;
    printf("Enter the number of elements in the array: ");
    scanf("%d",&n);
    printf("Enter the elements in the array: ");
    for(i=0;i<n;i++)
        scanf("%d%*c",arr+i);

    for(i=0;i<n;i++){
        smallest =arr[i];
        pos =i;
        for(j=i;j<n;j++){
            if(arr[j]<smallest){
                smallest= arr[j];
                pos =j;
            }
            comp++;
        }
        if(pos!=i){
            temp = arr[pos];
            arr[pos] = arr[i];
            arr[i]=temp;
            swaps++;
        }

    }

    printf("Sorted Array: ");
    for(i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
    printf("\nNumber of comparisons = %d\n",comp);
    printf("Number of swaps= %d\n",swaps);

}
```

Sample Input/Output

```
Sample Input1 :
5
23 43 56 78 91

Sample Output1 :
23 43 56 78 91
Number of comparisons : 15
Number of swaps : 0
```

Sample input 2:

```
5
91 78 56 43 23
```

Sample Output 2:

```
Sorted Array: 23 43 56 78 91
Number of comparisons = 15
Number of swaps= 2
```

Sample input 3:

```
5
43 78 56 91 23
```

Sample output 3:

```
Sorted Array: 23 43 56 78 91
Number of comparisons = 15
Number of swaps= 3
```