DHA Suffa University
Department of Computer Science
CS 2001L – Data Structures and Algorithms Lab
Spring 2021

Lab 03 – Sorting Algorithms

Objective:

To learn about

- Bubble Sort
- Selection Sort
- Insertion Sort

Bubble Sort

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order.

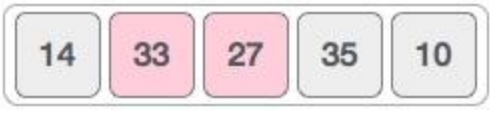
How Bubble Sort Works

We take an unsorted array for our example. Bubble sort takes $O(n^2)$ time.

Bubble sort starts with very first two elements, comparing them to check which one is greater.

In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.

We find that 27 is smaller than 33 and these two values must be swapped.



The new array should look like this −

Next we compare 33 and 35. We find that both are in already sorted positions.

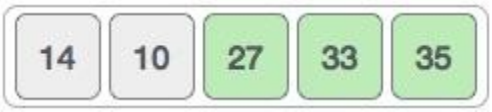
Then we move to the next two values, 35 and 10.

We know then that 10 is smaller 35. Hence they are not sorted.

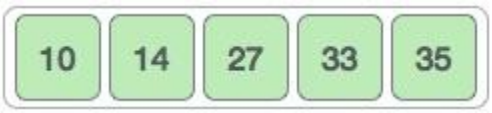
We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this −

To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this −

Notice that after each iteration, at least one value moves at the end.



And when there's no swap required, bubble sorts learns that an array is completely sorted.



Now we should look into some practical aspects of bubble sort.

Implementation:

```
#include <iostream>
//sort array of size n using Bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j,temp;
    for (i = 0; i < n-1; i++)
    {
        // Last i elements are already in place
        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;
            }
        }
    }
}

int main()
{
    int n, i;
    std::cout<<"Enter number of elements: ";
    std::cin>>n;
    int arr[n];
    for(i=0;i<n;i++)
    {
        std::cout<<"Enter Element "<<i+1<<": ";
        std::cin>>arr[i];
    }
    bubbleSort(arr, n);
    std::cout<<"\nSorted Data: \n";
    for(i=0; i<n; i++)
        std::cout<<arr[i]<<"\n";
    return 0;
}
```

Selection Sort:

Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.

The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.

How Selection Sort Works?

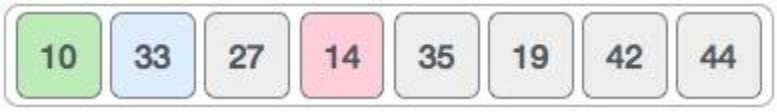
Consider the following depicted array as an example.

For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value.

So we replace 14 with 10. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of the sorted list.

For the second position, where 33 is residing, we start scanning the rest of the list in a linear manner.

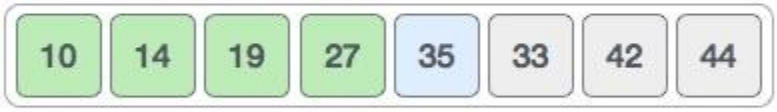
We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.

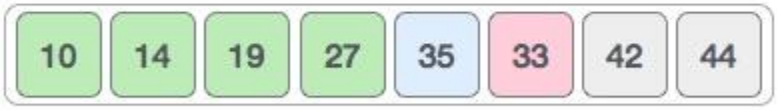


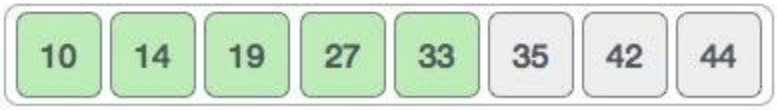
After two iterations, two least values are positioned at the beginning in a sorted manner.

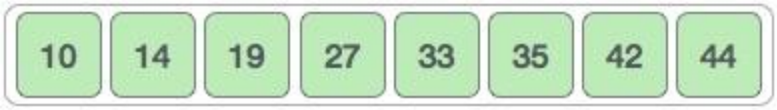
The same process is applied to the rest of the items in the array.

Following is a pictorial depiction of the entire sorting process −









Implementation:

```
#include <iostream>
//sort array of size n using Selection sort
void selectionSort(int arr[], int n)
{
   //pos_min is short for position of min
   int pos_min, temp;
   for (int i = 0; i < n-1; i++)
   {
       pos_min=i; //set pos_min to the current index of array
       for (int j = i+1; j < n; j++)
       {
           if (arr[j] < arr[pos_min])
           {
              pos_min=j;
              //pos_min will keep track of the index that min is in, this is needed when a swap happens
           }
       }
       //if pos_min no longer equals i, then a smaller value must have been found, so a swap must occur
       if(pos_min != i)
       {
            temp=arr[i];
            arr[i]=arr[pos_min];
            arr[pos_min]=temp;
       }
    }
}
int main()
{
     int n, i;
     std::cout<<"Enter number of element: ";
     std::cin>>n;
     int arr[n];
     for(i=0;i<n;i++)
     {
          std::cout<<"Enter Element "<<i+1<<": ";
          std::cin>>arr[i];
     }
     selectionSort(arr, n);
     std::cout<<"\nSorted Data: \n";
     for(i=0; i<n; i++)
          std::cout<<arr[i]<<"\n";
     return 0;
}
```

Insertion Sort:

This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be inserted in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, insertion sort.

The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array).

How Insertion Sort Works?

We take an unsorted array for our example.

Insertion sort compares the first two elements.

It finds that both 14 and 33 are already in ascending order. For now, 14 is in sorted sub-list.

Insertion sort moves ahead and compares 33 with 27.

And finds that 33 is not in the correct position.

It swaps 33 with 27. It also checks with all the elements of sorted sub-list. Here we see that the sorted sub-list has only one element 14, and 27 is greater than 14. Hence, the sorted sub-list remains sorted after swapping.

By now we have 14 and 27 in the sorted sub-list. Next, it compares 33 with 10.

These values are not in a sorted order.

So we swap them.

However, swapping makes 27 and 10 unsorted.

Hence, we swap them too.

Again we find 14 and 10 in an unsorted order.

We swap them again. By the end of third iteration, we have a sorted sub-list of 4 items.

This process goes on until all the unsorted values are covered in a sorted sub-list.

Implementation:

```
#include<iostream>
void insertionSort(int array[], int n)
{
    for( int i = 1 ;i < n ; i++ )
    {
        int temp = array[ i ];
        int j = i;
        while(j > 0  && temp <=array[ j -1])
        {
            array[ j ] = array[ j-1];
            j= j - 1;
        }
        array[ j ] = temp;
     }
}

int main()
{
    int n, i;
    std::cout<<"Enter number of elements: ";
    std::cin>>n;
    int arr[n];
    for(i=0;i<n;i++)
    {
        std::cout<<"Enter Element "<<i+1<<": ";
        std::cin>>arr[i];
    }
    insertionSort(arr, n);
    std::cout<<"\nSorted Data: \n";
    for(i=0; i<n; i++)
        std::cout<<arr[i]<<"\n";
    return 0;
}
```

LAB ASSIGNMENT

1. Following is the data of students including their Roll number, Name, Course Code and Course Name.

Roll Number	Name	Course Code	Course Name
CS13102	Tom	CS102	PF
CS10156	Jerry	CS306	DE
CS56782	Blossom	CS404	PIT
CS12470	Buttercup	CS102	PF
CS10111	Bubbles	CS404	PIT
CS13026	Sylvester	CS102	PF
CS13025	Bunny	CS404	PIT
CS10101	Daffy	CS101	ITC
CS16024	Tweety	CS236	CAL

Sort the column name by applying **each sorting algorithm** in such a way that their respective data in the rows is shifted with the names too while sorting.

SUBMISSION GUIDELINES

- Take a screenshot of each task (code and its output), labeled properly.
- Place all the screenshots and the code files (properly labeled) in a single folder labeled with Roll No and Lab No. e.g. **'cs191xxx_Lab01'**.
- Submit the folder at LMS
- **-100%** policies for plagiarism.