Mr. Giansante

C++ Programming
Sorting

August 2018

Sorting Algorithms

A Sorting Algorithm is used to rearrange a given array elements according to a comparison operator on the elements.

There are hundreds of sorting algorithms. Some of the more common ones include:

- Bubble Sort
- Recursive Bubble Sort
- Insertion Sort
- Recursive Insertion Sort
- Selection Sort
- Merge Sort
- Iterative Merge Sort
- Quick Sort

The Bubble Sort

Elements stored in arrays can easily be sorted using a technique known as the **Bubble Sort**. The Bubble Sort is one of the most straight-forward of the numerous sort methods available, but it is also one of the least efficient. Nevertheless, if you want to sort a few hundred items, this method is one of the fastest available.

The bubble sort algorithm works by comparing adjactent array elements and interchanging (swapping) the ones that are out of order.

Example:

First Pass:

(5 1 4 2 8) –> (1 5 4 2 8), Swap since 5 > 1.
(1 5 4 2 8) –> (1 4 5 2 8), Swap since 5 > 4
(1 4 5 2 8) –> (1 4 2 5 8), Swap since 5 > 2
(1 4 2 5 8) –> (1 4 2 5 8), No swap since 8 > 5

Second Pass:

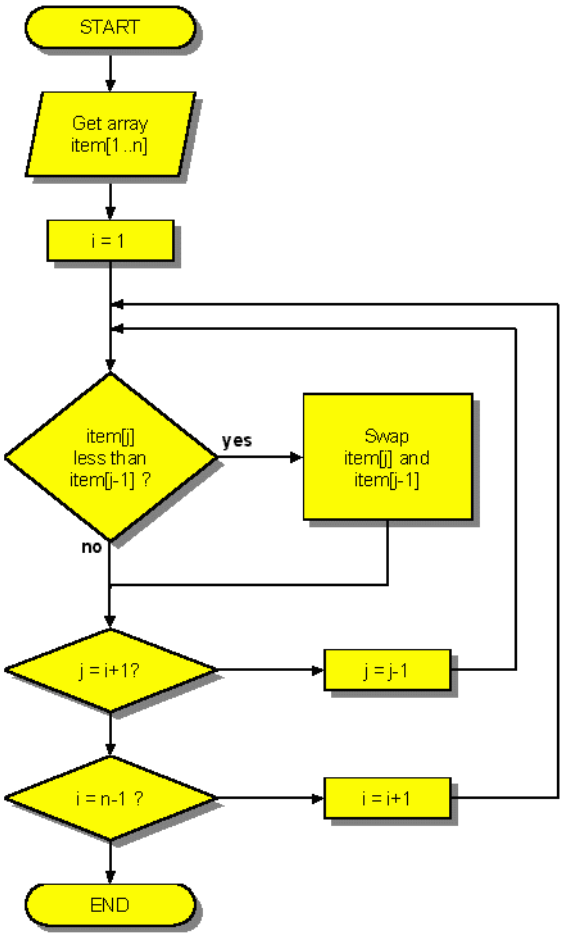
(1 4 2 5 8) –> (1 4 2 5 8)
(1 4 2 5 8) –> (1 2 4 5 8), Swap since 4 > 2
(1 2 4 5 8) –> (1 2 4 5 8)
(1 2 4 5 8) –> (1 2 4 5 8)

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one whole pass without any swap to know it is sorted.

Third Pass:

(1 2 4 5 8) –> (1 2 4 5 8)
(1 2 4 5 8) –> (1 2 4 5 8)
(1 2 4 5 8) –> (1 2 4 5 8)
(1 2 4 5 8) –> (1 2 4 5 8)

Flowchart for the Bubble Sort



Note: The flowchart below varies slightly from the algorithm discussed to the left.

Source: Edge Diagrammer, © PaceStar Software

The Bubble Sort

```
// C++ program for implementation of Bubble sort

#include <iostream>

using namespace std;

// Function to implement Bubble Sort

void bubbleSort(int arr[], int size)
{
     int temp;

     for (int i = 0; i < size-1; i++)
     {
          for (int j = 0; j < size-i-1; j++)
          {
               if (arr[j] > arr[j+1])
               {
                    temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j + 1] = temp;
               }
          }
     }
}

// Function to print an array

void printArray(int arr[], int size)
{
     for (int i=0; i < size; i++)
     {
          cout << arr[i] << ", ";
     }
     // erase the extra ", " at the end
     cout << "\b\b ";  // moves cursor back 2 places and then puts a space
}

int main()
{
     int myArray[] = {64, 34, 25, 12, 22, 11, 90};

     int myArraySize = sizeof(myArray)/sizeof(myArray[0]);

     cout << "Original array: \n";

     printArray(myArray, myArraySize);

     bubbleSort(myArray, myArraySize);

     cout << "\n\nSorted array: \n";

     printArray(myArray, myArraySize);

     return 0;
}
```

The Recursive Bubble Sort

Source: Suprotik Dey

Recursive Bubble Sort has no performance/implementation advantages over the regular Bubble Sort, but can be a good exercise to check one's understanding of Bubble Sort and recursion.

If we take a closer look at Bubble Sort algorithm, we notice that in first pass, we move the largest element to the end (assuming sorting in increasing order). In second pass, we move the second largest element to the second last position and so on.

Recursion Idea.

- Base Case: If array size is 1, return.
- Do One Pass of normal Bubble Sort. This pass fixes last element of current subarray.
- Recur for all elements except last of current subarray.

```
// C++ program for implementation of Recursive Bubble Sort

#include <iostream>

using namespace std;

// Function to implement Recursive Bubble Sort

void recursiveBubbleSort(int arr[], int size)
{
     int temp;

     // Base case
     if (size == 1) return;

     // One pass of bubble sort. After this pass, the largest element is moved (or bubbled) to end
     for (int i = 0; i <size-1; i++)
     {
          if (arr[i] > arr[i+1])
          {
               temp = arr[i];
               arr[i] = arr[i+1];
               arr[i + 1] = temp;
          }
     }

     // Largest element is fixed, recur for remaining array
     recursiveBubbleSort(arr, size-1);
}

/* Function to print an array */

void printArray(int arr[], int size)
{
     for (int i=0; i < size; i++)
     {
          cout << arr[i] << ", ";
     }
     // erase the extra ", " at the end
     cout << "\b\b ";  // moves cursor back 2 places and then puts a space
}

int main()
{
     int myArray[] = {64, 34, 25, 12, 22, 11, 90};

     int myArraySize = sizeof(myArray)/sizeof(myArray[0]);

     cout << "Original array: \n";

     printArray(myArray, myArraySize);

     recursiveBubbleSort(myArray, myArraySize);

     cout << "\n\nSorted array: \n";

     printArray(myArray, myArraySize);

     return 0;
}
```

The Selection Sort

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1. The subarray which is already sorted.
2. Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

Following example explains the above steps:

myArray[ ] = 64 25 12 22 11

Find the minimum element in myArray[0...4] and place it at beginning

11 25 12 22 64

Find the minimum element in myArray[1...4] and place it at beginning of myArray[1...4]

11 12 25 22 64

Find the minimum element in myArray[2...4] and place it at beginning of myArray[2...4]

11 12 22 25 64

Find the minimum element in myArray[3...4] and place it at beginning of myArray[3...4]

11 12 22 25 64

The Selection Sort

```
// C++ program for implementation of Selection Sort

#include <iostream>

using namespace std;

// Function to implement Selection Sort

void selectionSort(int arr[], int size)
{
     int min_index;
     int temp;

     // One by one move boundary of unsorted subarray
     for (int i = 0; i < size-1; i++)
     {
          // Find the minimum element in unsorted array
          min_index = i;

          for (int j = i+1; j < size; j++)
          {
               if (arr[j] < arr[min_index]){ min_index = j; }
          }
          // Swap the found minimum element with the first element
          temp = arr[min_index];
          arr[min_index] = arr[i];
          arr[i] = temp;
     }
}

/* Function to print an array */

void printArray(int arr[], int size)
{
     for (int i=0; i < size; i++)
     {
          cout << arr[i] << ", ";
     }
     // erase the extra ", " at the end
     cout << "\b\b ";  // moves cursor back 2 places and then puts a space
}

int main()
{
     int myArray[] = {64, 34, 25, 12, 22, 11, 90};

     int myArraySize = sizeof(myArray)/sizeof(myArray[0]);

     cout << "Original array: \n";

     printArray(myArray, myArraySize);

     selectionSort(myArray, myArraySize);

     cout << "\n\nSorted array: \n";

     printArray(myArray, myArraySize);

     return 0;
}
```

The Insertion Sort

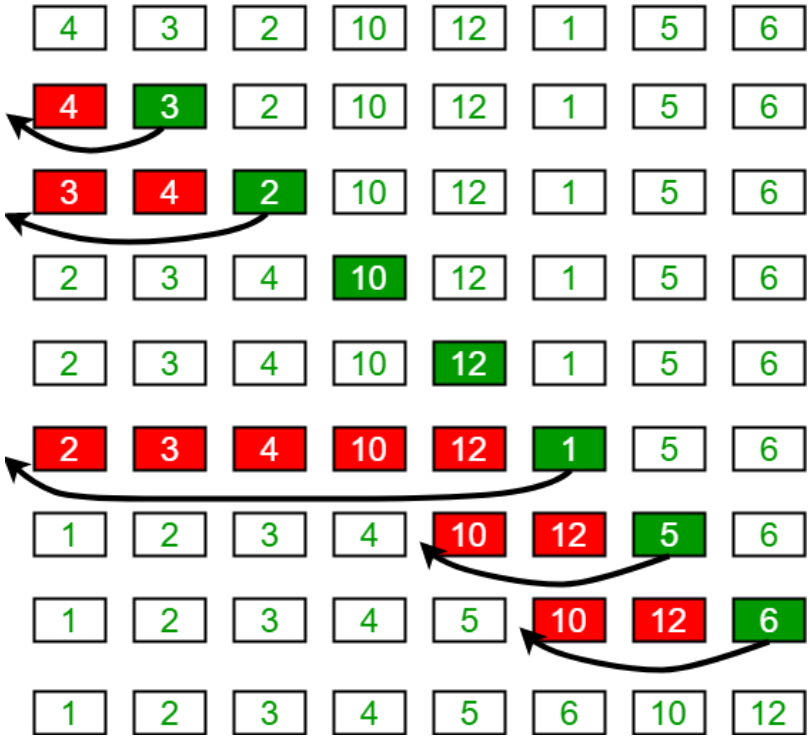
Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

Algorithm

Loop from i = 1 to size-1

Pick element myArray[i] and insert it into sorted sequence arr[0…i-1]

Example 1.



Example 2.

Sor the following array: 12, 11, 13, 5, 6

Let us loop for i = 1 (second element of the array) to 5 (size of input array)

Iteration	Action Taken	Resulting Array
i = 1	Since 11 is smaller than 12, move 12 and insert 11 before 12	11, 12, 13, 5, 6
i = 2	13 will remain at its position as all elements in A[0..i-1] are smaller than 13	11, 12, 13, 5, 6
i = 3	5 will move to the beginning and all other elements from 11 to 13 will move one position ahead of their current position	5, 11, 12, 13, 6
i = 4	6 will move to position after 5, and elements from 11 to 13 will move one position ahead of their current position	5, 6, 11, 12, 13

The Insertion Sort

```
// C++ program for implementation of Insertion Sort

#include <iostream>

using namespace std;

// Function to implement Insertion Sort

void insertionSort(int arr[], int size)
{
     int key;
     int j;

     for (int i = 1; i < size; i++)
     {
          key = arr[i];
          j = i-1;

          // Move elements of arr[0..i-1], that are greater than key,
          // to one position ahead of their current position

          while (j >= 0 && arr[j] > key)
          {
               arr[j+1] = arr[j];
               j = j-1;
          }

          arr[j+1] = key;
     }
}

/* Function to print an array */

void printArray(int arr[], int size)
{
     for (int i=0; i < size; i++)
     {
          cout << arr[i] << ", ";
     }
     // erase the extra ", " at the end
     cout << "\b\b ";  // moves cursor back 2 places and then puts a space
}

int main()
{
     int myArray[] = {64, 34, 25, 12, 22, 11, 90};

     int myArraySize = sizeof(myArray)/sizeof(myArray[0]);

     cout << "Original array: \n";

     printArray(myArray, myArraySize);

     insertionSort(myArray, myArraySize);

     cout << "\n\nSorted array: \n";

     printArray(myArray, myArraySize);

     return 0;
}
```