```
/** sortingalgorithms.cpp by Robert Szkutak **/

#include <stdlib.h>//Included for STL rand()
#include <iostream>//Included for STL cout
#include <time.h>//Included for STL time()

#define MAX_ARRAY_ROWS 10//Defines the maximum size of an array

void PrintData(int dataArray[]);//Outputs the content of an array
void ScrambleData(int dataArray[]);//Fills the array with random values in a random order

void SelectionSort(int dataArray[]);//Performs a selection sort on an array
void InsertionSort(int dataArray[]);//Performs an insertion sort on an array
void BubbleSort(int dataArray[]);//Performs a bubble sort on an array

int main()
{
	int dataArray[MAX_ARRAY_ROWS];//The array we will be sorting

	srand(time(0));//Seed the random number generator

	ScrambleData(dataArray);//Fill the array with random data and output contents
	std::cout << "Data before sorting:\n";
	PrintData(dataArray);

	InsertionSort(dataArray);//Sort the array and output contents
	std::cout << "Data after Insertion Sort:\n";
	PrintData(dataArray);

	ScrambleData(dataArray);//Fill the array with random data and output contents
	std::cout << "Data before sorting:\n";
	PrintData(dataArray);

	SelectionSort(dataArray);//Sort the array and output contents
	std::cout << "Data after Selection Sort:\n";
	PrintData(dataArray);

	ScrambleData(dataArray);//Fill the array with random data and output contents
	std::cout << "Data before sorting:\n";
	PrintData(dataArray);

	BubbleSort(dataArray);//Sort the array and output contents
	std::cout << "Data after Bubble Sort:\n";
	PrintData(dataArray);

	return 0;

}

/**
  Fills an array with random data
```

```
  @param the array to be filled with data
 */
void ScrambleData(int dataArray[])
{
	for(int i  = 0; i < MAX_ARRAY_ROWS; i++)
	{
		dataArray[i] = rand() % 20 + 1;//Data
	}
}

/**
  Outputs the contents of the array via STL
  @param the array whose contents are output
*/
void PrintData(int dataArray[])
{
	for(int i = 0; i < MAX_ARRAY_ROWS; i++)
		std::cout << dataArray[i] << "\n";
}

/**
  Performs a selection sort on the contents of a one dimensional array
  @param the array to be sorted
*/
void SelectionSort(int dataArray[])
{
	int current, smallest, holdData, walker;

	for(current = 0; current < MAX_ARRAY_ROWS; current++)
	{
		smallest = current;
		for(walker = current + 1; walker <= MAX_ARRAY_ROWS; walker++)
			if(dataArray[walker] < dataArray[smallest])
				smallest = walker;
		holdData = dataArray[current];
		dataArray[current] = dataArray[smallest];
		dataArray[smallest] = holdData;
	}
}

/**
  Performs an insertion sort on the contents of a one dimensional array
  @param the array to be sorted
*/
void InsertionSort(int dataArray[])
{
	int current, hold, walker;

	for(current = 1; current <= MAX_ARRAY_ROWS; current++)
	{
```

```
                hold = dataArray[current];
                for(walker = current -1; walker >= 0 && hold < dataArray[walker]; walker--)
                        dataArray[walker+1] = dataArray[walker];
                dataArray[walker+1] = hold;
        }
}

/**
  Performs a bubble sort on the contents of a one dimensional array
  @param the array to be sorted
*/
void BubbleSort(int dataArray[])
{
        int current, walker, temp;
        bool sorted = false;

        for(current = 0, sorted = false; current <= MAX_ARRAY_ROWS && !sorted; current++)
                for(walker = MAX_ARRAY_ROWS, sorted = true; walker > current; walker--)
                        if(dataArray[walker] < dataArray[walker - 1])
                        {
                                sorted = false;
                                temp = dataArray[walker];
                                dataArray[walker] = dataArray[walker - 1];
                                dataArray[walker - 1] = temp;
                        }
}
```