Status	Finished
Started	Sunday, 20 October 2024, 1:36 PM
Completed	Monday, 21 October 2024, 8:04 PM
Duration	1 day 6 hours
Marks	3.00/3.00
Grade	**10.00** out of 10.00 (**100**%)

Question **1**

Correct

Mark 1.00 out of 1.00

Implement method bubbleSort() in class SLinkedList to sort this list in ascending order. After each bubble, we will print out a list to check (using printList).

```
#include <iostream>
#include <sstream>
using namespace std;

template <class T>
class SLinkedList {
public:
    class Node; // Forward declaration
protected:
    Node* head;
    Node* tail;
    int count;
public:
    SLinkedList()
    {
      this->head = nullptr;
      this->tail = nullptr;
      this->count = 0;
    }
    ~SLinkedList(){};
    void add(T e)
    {
        Node *pNew = new Node(e);

        if (this->count == 0)
        {
            this->head = this->tail = pNew;
        }
        else
        {
            this->tail->next = pNew;
            this->tail = pNew;
        }

        this->count++;
    }
    int size()
    {
        return this->count;
    }
    void printList()
    {
        stringstream ss;
        ss << "[";
        Node *ptr = head;
        while (ptr != tail)
        {
            ss << ptr->data << ",";
            ptr = ptr->next;
        }

        if (count > 0)
            ss << ptr->data << "]";
        else
            ss << "]";
        cout << ss.str() << endl;
    }
public:
    class Node {
    private:
        T data;
        Node* next;
        friend class SLinkedList<T>;
    public:
        Node() {
            next = 0;
        }
        Node(T data) {
            this->data = data;
            this->next = nullptr;
        }
    };

    void bubbleSort();
};
```

For example:

Test	Result

`int arr[] = {9, 2, 8, 4, 1};`
`SLinkedList<int> list;`
`for(int i = 0; i <int(sizeof(arr))/4;i++)`
`    list.add(arr[i]);`
`list.bubbleSort();`
	`[2,8,4,1,9]`
	`[2,4,1,8,9]`
	`[2,1,4,8,9]`
	`[1,2,4,8,9]`

Answer: (penalty regime: 0 %)

Reset answer

```
template <class T>
void SLinkedList<T>::bubbleSort()
{
    Node*tailtemp=tail;
    Node*prev=nullptr;
    Node*dummy=head;
    while(dummy!=tailtemp){
        while(dummy!=tailtemp){
            if(dummy->data>dummy->next->data){
                int temp=dummy->data;
                dummy->data=dummy->next->data;
                dummy->next->data=temp;
            }
            prev=dummy;
            dummy=dummy->next;
        }
        //headtemp=headtemp->next;
        tailtemp=prev;
        dummy=head;
        printList();
    }
}
```

	Test	Expected	Got	

✓	`int arr[] = {9, 2, 8, 4, 1};`
	`SLinkedList<int> list;`
	`for(int i = 0; i <int(sizeof(arr))/4;i++)`
	`    list.add(arr[i]);`
	`list.bubbleSort();`
		[2,8,4,1,9]
		[2,4,1,8,9]
		[2,1,4,8,9]
		[1,2,4,8,9]
			[2,8,4,1,9]
			[2,4,1,8,9]
			[2,1,4,8,9]
			[1,2,4,8,9]
				✓

✓	`int arr[] = {9, 2, 8, 1, 1, 0, -2};`
	`SLinkedList<int> list;`
	`for(int i = 0; i <int(sizeof(arr))/4;i++)`
	`    list.add(arr[i]);`
	`list.bubbleSort();`
		[2,8,1,1,0,-2,9]
		[2,1,1,0,-2,8,9]
		[1,1,0,-2,2,8,9]
		[1,0,-2,1,2,8,9]
		[0,-2,1,1,2,8,9]
		[-2,0,1,1,2,8,9]
			[2,8,1,1,0,-2,9]
			[2,1,1,0,-2,8,9]
			[1,1,0,-2,2,8,9]
			[1,0,-2,1,2,8,9]
			[0,-2,1,1,2,8,9]
			[-2,0,1,1,2,8,9]
				✓

✓	`int arr[] = {1};`
	`SLinkedList<int> list;`
	`for(int i = 0; i < int(sizeof(arr))/4;i++)`
	`    list.add(arr[i]);`
	`list.bubbleSort();`
				✓

✓	`int arr[] = {1,4,12,6,5,3,2,-5,-6,-8};`
	`SLinkedList<int> list;`
	`for(int i = 0; i < int(sizeof(arr))/4;i++)`
	`    list.add(arr[i]);`
	`list.bubbleSort();`
		[1,4,6,5,3,2,-5,-6,-8,12]
		[1,4,5,3,2,-5,-6,-8,6,12]
		[1,4,3,2,-5,-6,-8,5,6,12]
		[1,3,2,-5,-6,-8,4,5,6,12]
		[1,2,-5,-6,-8,3,4,5,6,12]
		[1,-5,-6,-8,2,3,4,5,6,12]
		[-5,-6,-8,1,2,3,4,5,6,12]
		[-6,-8,-5,1,2,3,4,5,6,12]
		[-8,-6,-5,1,2,3,4,5,6,12]
			[1,4,6,5,3,2,-5,-6,-8,12]
			[1,4,5,3,2,-5,-6,-8,6,12]
			[1,4,3,2,-5,-6,-8,5,6,12]
			[1,3,2,-5,-6,-8,4,5,6,12]
			[1,2,-5,-6,-8,3,4,5,6,12]
			[1,-5,-6,-8,2,3,4,5,6,12]
			[-5,-6,-8,1,2,3,4,5,6,12]
			[-6,-8,-5,1,2,3,4,5,6,12]
			[-8,-6,-5,1,2,3,4,5,6,12]
				✓

✓	`int arr[] = {1,1,1,2,-5,-6,-8};`
	`SLinkedList<int> list;`
	`for(int i = 0; i < int(sizeof(arr))/4;i++)`
	`    list.add(arr[i]);`
	`list.bubbleSort();`
		[1,1,1,-5,-6,-8,2]
		[1,1,-5,-6,-8,1,2]
		[1,-5,-6,-8,1,1,2]
		[-5,-6,-8,1,1,1,2]
		[-6,-8,-5,1,1,1,2]
		[-8,-6,-5,1,1,1,2]
			[1,1,1,-5,-6,-8,2]
			[1,1,-5,-6,-8,1,2]
			[1,-5,-6,-8,1,1,2]
			[-5,-6,-8,1,1,1,2]
			[-6,-8,-5,1,1,1,2]
			[-8,-6,-5,1,1,1,2]
				✓

✓	`int arr[] = {9,8,7,6,5,4};`
	`SLinkedList<int> list;`
	`for(int i = 0; i < int(sizeof(arr))/4;i++)`
	`    list.add(arr[i]);`
	`list.bubbleSort();`
		[8,7,6,5,4,9]
		[7,6,5,4,8,9]
		[6,5,4,7,8,9]
		[5,4,6,7,8,9]
		[4,5,6,7,8,9]
			[8,7,6,5,4,9]
			[7,6,5,4,8,9]
			[6,5,4,7,8,9]
			[5,4,6,7,8,9]
			[4,5,6,7,8,9]
				✓

	Test	Expected	Got	

✓	`int arr[] = {7,7,7,7,7};`
	`SLinkedList<int> list;`
	`for(int i = 0; i < int(sizeof(arr))/4;i++)`
	`    list.add(arr[i]);`
	`list.bubbleSort();`
		`[7,7,7,7,7]`
		`[7,7,7,7,7]`
		`[7,7,7,7,7]`
		`[7,7,7,7,7]`
			`[7,7,7,7,7]`
			`[7,7,7,7,7]`
			`[7,7,7,7,7]`
			`[7,7,7,7,7]`
				✓

✓	`int arr[] = {7,-7,1,-7,7};`
	`SLinkedList<int> list;`
	`for(int i = 0; i < int(sizeof(arr))/4;i++)`
	`    list.add(arr[i]);`
	`list.bubbleSort();`
		`[-7,1,-7,7,7]`
		`[-7,-7,1,7,7]`
		`[-7,-7,1,7,7]`
		`[-7,-7,1,7,7]`
			`[-7,1,-7,7,7]`
			`[-7,-7,1,7,7]`
			`[-7,-7,1,7,7]`
			`[-7,-7,1,7,7]`
				✓

✓	`int arr[] = {1,2,3,4,5};`
	`SLinkedList<int> list;`
	`for(int i = 0; i < int(sizeof(arr))/4;i++)`
	`    list.add(arr[i]);`
	`list.bubbleSort();`
		`[1,2,3,4,5]`
		`[1,2,3,4,5]`
		`[1,2,3,4,5]`
		`[1,2,3,4,5]`
			`[1,2,3,4,5]`
			`[1,2,3,4,5]`
			`[1,2,3,4,5]`
			`[1,2,3,4,5]`
				✓

✓	`int arr[] = {1,2,6,-9};`
	`SLinkedList<int> list;`
	`for(int i = 0; i < int(sizeof(arr))/4;i++)`
	`    list.add(arr[i]);`
	`list.bubbleSort();`
		`[1,2,-9,6]`
		`[1,-9,2,6]`
		`[-9,1,2,6]`
			`[1,2,-9,6]`
			`[1,-9,2,6]`
			`[-9,1,2,6]`
				✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **2**

Correct

Mark 1.00 out of 1.00

Implement static methods **sortSegment** and **ShellSort** in class **Sorting** to sort an array in ascending order.

```
#ifndef SORTING_H
#define SORTING_H

#include <sstream>
#include <iostream>
#include <type_traits>
using namespace std;

template <class T>
class Sorting {
private:
    static void printArray(T* start, T* end)
    {
        int size = end - start;
        for (int i = 0; i < size; i++)
            cout << start[i] << " ";
        cout << endl;
    }

public:
```

```
    // TODO: Write your code here
    static void sortSegment(T* start, T* end, int segment_idx, int cur_segment_total);
    static void ShellSort(T* start, T* end, int* num_segment_list, int num_phases);
};
```

```
#endif /* SORTING_H */
```

For example:

Test	Result

`int num_segment_list[] = {1, 3, 5};`
`int num_phases = 3;`
`int array[] = { 10, 9, 8 , 7 , 6, 5, 4, 3, 2, 1 };`
 	
`Sorting<int>::ShellSort(&array[0], &array[10], &num_segment_list[0], num_phases);`
	`5 segments: 5 4 3 2 1 10 9 8 7 6`
	`3 segments: 2 1 3 5 4 7 6 8 10 9`
	`1 segments: 1 2 3 4 5 6 7 8 9 10`

Answer: (penalty regime: 0 %)

Reset answer

```
static void sortSegment(T* start, T* end, int segment_idx, int cur_segment_total) {
    // TODO
    int size = end - start;
    for (int curr = segment_idx + cur_segment_total; curr < size; curr += cur_segment_total) {
        int tmp = start[curr];
        int i;
        for (i = curr - cur_segment_total; i >= 0 && start[i] > tmp;
            i -= cur_segment_total) {
                start[i + cur_segment_total] = start[i];
            }
        start[i + cur_segment_total] = tmp;
    }
}

static void ShellSort(T* start, T* end, int* num_segment_list, int num_phases) {
    // TODO
    // Note: You must print out the array after sorting segments to check whether your algorithm is true.
    for (int phase = num_phases - 1; phase >= 0; phase--) {
        int step = num_segment_list[phase];
        for (int segment = 0; segment < step; segment++) {
            sortSegment(start, end, segment, step);
        }
        cout << step << " segments: ";
        printArray(start, end);
    }
}
```

	Test	Expected	Got	

✓	`int num_segment_list[] = {1, 3, 5};`
	`int num_phases = 3;`
	`int array[] = { 10, 9, 8 , 7 , 6, 5, 4, 3, 2, 1 };`
	
	`Sorting<int>::ShellSort(&array[0], &array[10], &num_segment_list[0], num_phases);`
		5 segments: 5 4 3 2 1 10 9 8 7 6
		3 segments: 2 1 3 5 4 7 6 8 10 9
		1 segments: 1 2 3 4 5 6 7 8 9 10
			5 segments: 5 4 3 2 1 10 9 8 7 6
			3 segments: 2 1 3 5 4 7 6 8 10 9
			1 segments: 1 2 3 4 5 6 7 8 9 10
				✓

✓	`int num_segment_list[] = { 1, 2, 6 };`
	`int num_phases = 3;`
	`int array[] = { 10, 9, 8 , 7 , 6, 5, 4, 3, 2, 1 };`
	
	`Sorting<int>::ShellSort(&array[0], &array[10], &num_segment_list[0], num_phases);`
		6 segments: 4 3 2 1 6 5 10 9 8 7
		2 segments: 2 1 4 3 6 5 8 7 10 9
		1 segments: 1 2 3 4 5 6 7 8 9 10
			6 segments: 4 3 2 1 6 5 10 9 8 7
			2 segments: 2 1 4 3 6 5 8 7 10 9
			1 segments: 1 2 3 4 5 6 7 8 9 10
				✓

✓	`int num_segment_list[] = { 1, 2, 5 };`
	`int num_phases = 3;`
	`int array[] = { 10, 9, 8 , 7 , 6, 5, 4, 3, 2, 1 };`
	
	`Sorting<int>::ShellSort(&array[0], &array[10], &num_segment_list[0], num_phases);`
		5 segments: 5 4 3 2 1 10 9 8 7 6
		2 segments: 1 2 3 4 5 6 7 8 9 10
		1 segments: 1 2 3 4 5 6 7 8 9 10
			5 segments: 5 4 3 2 1 10 9 8 7 6
			2 segments: 1 2 3 4 5 6 7 8 9 10
			1 segments: 1 2 3 4 5 6 7 8 9 10
				✓

✓	`int num_segment_list[] = { 1, 2, 3 };`
	`int num_phases = 3;`
	`int array[] = { 10, 9, 8 , 7 , 6, 5, 4, 3, 2, 1 };`
	
	`Sorting<int>::ShellSort(&array[0], &array[10], &num_segment_list[0], num_phases);`
		3 segments: 1 3 2 4 6 5 7 9 8 10
		2 segments: 1 3 2 4 6 5 7 9 8 10
		1 segments: 1 2 3 4 5 6 7 8 9 10
			3 segments: 1 3 2 4 6 5 7 9 8 10
			2 segments: 1 3 2 4 6 5 7 9 8 10
			1 segments: 1 2 3 4 5 6 7 8 9 10
				✓

✓	`int num_segment_list[] = { 1, 5, 8, 10 };`
	`int num_phases = 4;`
	`int array[] = { 3, 5, 7, 10 ,12, 14, 15, 13, 1, 2, 9, 6, 4, 8, 11 };`
	
	`Sorting<int>::ShellSort(&array[0], &array[15], &num_segment_list[0], num_phases);`
		10 segments: 3 5 4 8 11 14 15 13 1 2 9 6 7 10 12
		8 segments: 1 2 4 6 7 10 12 13 3 5 9 8 11 14 15
		5 segments: 1 2 4 3 5 9 8 11 6 7 10 12 13 14 15
		1 segments: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
			10 segments: 3 5 4 8 11 14 15 13 1 2 9 6 7 10 12
			8 segments: 1 2 4 6 7 10 12 13 3 5 9 8 11 14 15
			5 segments: 1 2 4 3 5 9 8 11 6 7 10 12 13 14 15
			1 segments: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
				✓

	Test	Expected	Got	

✓	`int num_segment_list[] = { 1, 5, 7, 10 };`
	`int num_phases = 4;`
	`int array[] = { 3, 5, 7, 10 ,12, 14, 15, 13, 1, 2, 9, 6, 4, 8, 11 };`
	
	`Sorting<int>::ShellSort(&array[0], &array[15], &num_segment_list[0], num_phases);`
		10 segments: 3 5 4 8 11 14 15 13 1 2 9 6 7 10 12
		7 segments: 3 1 2 8 6 7 10 12 5 4 9 11 14 15 13
		5 segments: 3 1 2 5 4 7 10 12 8 6 9 11 14 15 13
		1 segments: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
			10 segments: 3 5 4 8 11 14 15 13 1 2 9 6 7 10 12
			7 segments: 3 1 2 8 6 7 10 12 5 4 9 11 14 15 13
			5 segments: 3 1 2 5 4 7 10 12 8 6 9 11 14 15 13
			1 segments: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
				✓

✓	`int num_segment_list[] = { 1, 3, 5, 10 };`
	`int num_phases = 4;`
	`int array[] = { 3, 5, 7, 10 ,12, 14, 15, 13, 1, 2, 9, 6, 4, 8, 11 };`
	
	`Sorting<int>::ShellSort(&array[0], &array[15], &num_segment_list[0], num_phases);`
		10 segments: 3 5 4 8 11 14 15 13 1 2 9 6 7 10 12
		5 segments: 3 5 4 1 2 9 6 7 8 11 14 15 13 10 12
		3 segments: 1 2 4 3 5 8 6 7 9 11 10 12 13 14 15
		1 segments: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
			10 segments: 3 5 4 8 11 14 15 13 1 2 9 6 7 10 12
			5 segments: 3 5 4 1 2 9 6 7 8 11 14 15 13 10 12
			3 segments: 1 2 4 3 5 8 6 7 9 11 10 12 13 14 15
			1 segments: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
				✓

✓	`int num_segment_list[] = { 1, 3, 5, 10, 15 };`
	`int num_phases = 5;`
	`int array[] = { 3, 5, 7, 10 ,12, 14, 15, 13, 1, 2, 9, 6, 4, 8, 11, 16, 17, 18, 20, 19 };`
	
	`Sorting<int>::ShellSort(&array[0], &array[20], &num_segment_list[0], num_phases);`
		15 segments: 3 5 7 10 12 14 15 13 1 2 9 6 4 8 11 16 17 18 20 19
		10 segments: 3 5 4 8 11 14 15 13 1 2 9 6 7 10 12 16 17 18 20 19
		5 segments: 3 5 4 1 2 9 6 7 8 11 14 15 13 10 12 16 17 18 20 19
		3 segments: 1 2 4 3 5 8 6 7 9 11 10 12 13 14 15 16 17 18 20 19
		1 segments: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
			15 segments: 3 5 7 10 12 14 15 13 1 2 9 6 4 8 11 16 17 18 20 19
			10 segments: 3 5 4 8 11 14 15 13 1 2 9 6 7 10 12 16 17 18 20 19
			5 segments: 3 5 4 1 2 9 6 7 8 11 14 15 13 10 12 16 17 18 20 19
			3 segments: 1 2 4 3 5 8 6 7 9 11 10 12 13 14 15 16 17 18 20 19
			1 segments: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
				✓

✓	`int num_segment_list[] = { 1, 3, 5, 7, 12 };`
	`int num_phases = 5;`
	`int array[] = { 3, 5, 7, 10 ,12, 14, 15, 13, 1, 2, 9, 6, 4, 8, 11, 16, 17, 18, 20, 19 };`
	
	`Sorting<int>::ShellSort(&array[0], &array[20], &num_segment_list[0], num_phases);`
		12 segments: 3 5 7 10 12 14 15 13 1 2 9 6 4 8 11 16 17 18 20 19
		7 segments: 3 1 2 9 6 4 8 11 5 7 10 12 14 15 13 16 17 18 20 19
		5 segments: 3 1 2 5 6 4 8 11 9 7 10 12 14 15 13 16 17 18 20 19
		3 segments: 3 1 2 5 6 4 7 10 9 8 11 12 14 15 13 16 17 18 20 19
		1 segments: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
			12 segments: 3 5 7 10 12 14 15 13 1 2 9 6 4 8 11 16 17 18 20 19
			7 segments: 3 1 2 9 6 4 8 11 5 7 10 12 14 15 13 16 17 18 20 19
			5 segments: 3 1 2 5 6 4 8 11 9 7 10 12 14 15 13 16 17 18 20 19
			3 segments: 3 1 2 5 6 4 7 10 9 8 11 12 14 15 13 16 17 18 20 19
			1 segments: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
				✓

✓	`int num_segment_list[] = { 1, 2, 5, 8, 13 };`
	`int num_phases = 5;`
	`int array[] = { 3, 5, 7, 10 ,12, 14, 15, 13, 1, 2, 9, 6, 4, 8, 11, 16, 17, 18, 20, 19 };`
	
	`Sorting<int>::ShellSort(&array[0], &array[20], &num_segment_list[0], num_phases);`
		13 segments: 3 5 7 10 12 14 15 13 1 2 9 6 4 8 11 16 17 18 20 19
		8 segments: 1 2 7 6 4 8 11 13 3 5 9 10 12 14 15 16 17 18 20 19
		5 segments: 1 2 7 3 4 8 10 12 6 5 9 11 13 14 15 16 17 18 20 19
		2 segments: 1 2 4 3 6 5 7 8 9 11 10 12 13 14 15 16 17 18 20 19
		1 segments: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
			13 segments: 3 5 7 10 12 14 15 13 1 2 9 6 4 8 11 16 17 18 20 19
			8 segments: 1 2 7 6 4 8 11 13 3 5 9 10 12 14 15 16 17 18 20 19
			5 segments: 1 2 7 3 4 8 10 12 6 5 9 11 13 14 15 16 17 18 20 19
			2 segments: 1 2 4 3 6 5 7 8 9 11 10 12 13 14 15 16 17 18 20 19
			1 segments: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
				✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **3**

Correct

Mark 1.00 out of 1.00

Implement static method selectionSort in class **Sorting** to sort an array in ascending order. After each selection, we will print out a list to check (using printArray).

```
#include <iostream>
using namespace std;

template <class T>
class Sorting
{
public:
    /* Function to print an array */
    static void printArray(T *start, T *end)
    {
        int size = end - start;
        for (int i = 0; i < size - 1; i++)
            cout << start[i] << ", ";
        cout << start[size - 1];
        cout << endl;
    }

    static void selectionSort(T *start, T *end);
};
```

For example:

Test	Result

`int arr[] = {9, 2, 8, 1, 0, -2};`
`Sorting<int>::selectionSort(&arr[0], &arr[6]);`
	`-2, 2, 8, 1, 0, 9`
	`-2, 0, 8, 1, 2, 9`
	`-2, 0, 1, 8, 2, 9`
	`-2, 0, 1, 2, 8, 9`
	`-2, 0, 1, 2, 8, 9`

Answer: (penalty regime: 0 %)

Reset answer

```
template <class T>
void Sorting<T>::selectionSort(T *start, T *end)
{
    for(int i=0;i<end-start-1;i++){
        int min=i;
        for(int j=i+1;j<end-start;j++){
            if(start[j]<start[min]){
                min=j;
            }
        }
        swap(start[i],start[min]);
        printArray(start, end);
    }
}
```

	Test	Expected	Got	

✓	`int arr[] = {9, 2, 8, 1, 0, -2};`
	`Sorting<int>::selectionSort(&arr[0], &arr[6]);`
		`-2, 2, 8, 1, 0, 9`
		`-2, 0, 8, 1, 2, 9`
		`-2, 0, 1, 8, 2, 9`
		`-2, 0, 1, 2, 8, 9`
		`-2, 0, 1, 2, 8, 9`
			`-2, 2, 8, 1, 0, 9`
			`-2, 0, 8, 1, 2, 9`
			`-2, 0, 1, 8, 2, 9`
			`-2, 0, 1, 2, 8, 9`
			`-2, 0, 1, 2, 8, 9`
				✓

✓	`int arr[] = {9, 2, 8, 4, 1};`
	`Sorting<int>::selectionSort(&arr[0], &arr[5]);`
		`1, 2, 8, 4, 9`
		`1, 2, 8, 4, 9`
		`1, 2, 4, 8, 9`
		`1, 2, 4, 8, 9`
			`1, 2, 8, 4, 9`
			`1, 2, 8, 4, 9`
			`1, 2, 4, 8, 9`
			`1, 2, 4, 8, 9`
				✓

✓	`int arr[] = {9, 2, 1, 1, 1};`
	`Sorting<int>::selectionSort(&arr[0], &arr[5]);`
		`1, 2, 9, 1, 1`
		`1, 1, 9, 2, 1`
		`1, 1, 1, 2, 9`
		`1, 1, 1, 2, 9`
			`1, 2, 9, 1, 1`
			`1, 1, 9, 2, 1`
			`1, 1, 1, 2, 9`
			`1, 1, 1, 2, 9`
				✓

✓	`int arr[] = {9, 2, 1, -7, -9};`
	`Sorting<int>::selectionSort(&arr[0], &arr[5]);`
		`-9, 2, 1, -7, 9`
		`-9, -7, 1, 2, 9`
		`-9, -7, 1, 2, 9`
		`-9, -7, 1, 2, 9`
			`-9, 2, 1, -7, 9`
			`-9, -7, 1, 2, 9`
			`-9, -7, 1, 2, 9`
			`-9, -7, 1, 2, 9`
				✓

	Test	Expected	Got	

✓	int arr[] = {9, 2, 1, -7, -9, -9, 5, 6};
	Sorting<int>::selectionSort(&arr[0],
	&arr[8]);
		-9, 2, 1, -7, 9, -9, 5, 6
		-9, -9, 1, -7, 9, 2, 5, 6
		-9, -9, -7, 1, 9, 2, 5, 6
		-9, -9, -7, 1, 9, 2, 5, 6
		-9, -9, -7, 1, 2, 9, 5, 6
		-9, -9, -7, 1, 2, 5, 9, 6
		-9, -9, -7, 1, 2, 5, 6, 9
			-9, 2, 1, -7, 9, -9, 5, 6
			-9, -9, 1, -7, 9, 2, 5, 6
			-9, -9, -7, 1, 9, 2, 5, 6
			-9, -9, -7, 1, 9, 2, 5, 6
			-9, -9, -7, 1, 2, 9, 5, 6
			-9, -9, -7, 1, 2, 5, 9, 6
			-9, -9, -7, 1, 2, 5, 6, 9
				✓

✓	int arr[] = {9, 30, 1, -7, 7, -9, 5, 6};
	Sorting<int>::selectionSort(&arr[0],
	&arr[8]);
		-9, 30, 1, -7, 7, 9, 5, 6
		-9, -7, 1, 30, 7, 9, 5, 6
		-9, -7, 1, 30, 7, 9, 5, 6
		-9, -7, 1, 5, 7, 9, 30, 6
		-9, -7, 1, 5, 6, 9, 30, 7
		-9, -7, 1, 5, 6, 7, 30, 9
		-9, -7, 1, 5, 6, 7, 9, 30
			-9, 30, 1, -7, 7, 9, 5, 6
			-9, -7, 1, 30, 7, 9, 5, 6
			-9, -7, 1, 30, 7, 9, 5, 6
			-9, -7, 1, 5, 7, 9, 30, 6
			-9, -7, 1, 5, 6, 9, 30, 7
			-9, -7, 1, 5, 6, 7, 30, 9
			-9, -7, 1, 5, 6, 7, 9, 30
				✓

✓	int arr[] = {30, 7, 20, 0, -30, -7, -20, 0};
	Sorting<int>::selectionSort(&arr[0],
	&arr[8]);
		-30, 7, 20, 0, 30, -7, -20, 0
		-30, -20, 20, 0, 30, -7, 7, 0
		-30, -20, -7, 0, 30, 20, 7, 0
		-30, -20, -7, 0, 30, 20, 7, 0
		-30, -20, -7, 0, 0, 20, 7, 30
		-30, -20, -7, 0, 0, 7, 20, 30
		-30, -20, -7, 0, 0, 7, 20, 30
			-30, 7, 20, 0, 30, -7, -20, 0
			-30, -20, 20, 0, 30, -7, 7, 0
			-30, -20, -7, 0, 30, 20, 7, 0
			-30, -20, -7, 0, 30, 20, 7, 0
			-30, -20, -7, 0, 0, 20, 7, 30
			-30, -20, -7, 0, 0, 7, 20, 30
			-30, -20, -7, 0, 0, 7, 20, 30
				✓

✓	int arr[] = {-30, -7, -20, 0, -30, -7, -20,
	0};
	Sorting<int>::selectionSort(&arr[0],
	&arr[8]);
		-30, -7, -20, 0, -30, -7,
		-20, 0
		-30, -30, -20, 0, -7, -7,
		-20, 0
		-30, -30, -20, 0, -7, -7,
		-20, 0
		-30, -30, -20, -20, -7, -7,
		0, 0
		-30, -30, -20, -20, -7, -7,
		0, 0
		-30, -30, -20, -20, -7, -7,
		0, 0
		-30, -30, -20, -20, -7, -7,
		0, 0
			-30, -7, -20, 0, -30, -7,
			-20, 0
			-30, -30, -20, 0, -7, -7,
			-20, 0
			-30, -30, -20, 0, -7, -7,
			-20, 0
			-30, -30, -20, -20, -7, -7,
			0, 0
			-30, -30, -20, -20, -7, -7,
			0, 0
			-30, -30, -20, -20, -7, -7,
			0, 0
			-30, -30, -20, -20, -7, -7,
			0, 0
				✓

✓	int arr[] = {1,2,3,4,5,6,7};
	Sorting<int>::selectionSort(&arr[0],
	&arr[7]);
		1, 2, 3, 4, 5, 6, 7
		1, 2, 3, 4, 5, 6, 7
		1, 2, 3, 4, 5, 6, 7
		1, 2, 3, 4, 5, 6, 7
		1, 2, 3, 4, 5, 6, 7
		1, 2, 3, 4, 5, 6, 7
			1, 2, 3, 4, 5, 6, 7
			1, 2, 3, 4, 5, 6, 7
			1, 2, 3, 4, 5, 6, 7
			1, 2, 3, 4, 5, 6, 7
			1, 2, 3, 4, 5, 6, 7
			1, 2, 3, 4, 5, 6, 7
				✓

✓	int arr[] = {7,6,5,4,3,2,1};
	Sorting<int>::selectionSort(&arr[0],
	&arr[7]);
		1, 6, 5, 4, 3, 2, 7
		1, 2, 5, 4, 3, 6, 7
		1, 2, 3, 4, 5, 6, 7
		1, 2, 3, 4, 5, 6, 7
		1, 2, 3, 4, 5, 6, 7
		1, 2, 3, 4, 5, 6, 7
			1, 6, 5, 4, 3, 2, 7
			1, 2, 5, 4, 3, 6, 7
			1, 2, 3, 4, 5, 6, 7
			1, 2, 3, 4, 5, 6, 7
			1, 2, 3, 4, 5, 6, 7
			1, 2, 3, 4, 5, 6, 7
				✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.