CMPT 135: Lab Work Week 12

1. What is the output of each program?

a)
```
int f1(int x)
{
    if (x > 0)
       return x + f1(x-1);
    else
       return 1;
}
...
cout<< f1(3) <<endl;
```
3+2+1+1
7

b)
```
int f2(int x)
{
    if (x < 12)
       return x + f2(x*2);
    else
       return x;
}
...
cout<< f2(1) <<endl;
```
1+2+4+8+16
31

c)
```
int f3(int x)
{
    if (x > 2)
       return x * f3(x-2);
    else
       return 1;
}
...
cout<< f3(7) <<endl;
```
7 * 5 * 3 * 1
105

d)
```
int f4(int x)
{
    if (x > 0)
       return x + f4(x/3);
    else
       return 0;
}
...
cout<< f4(100) <<endl;
```
100+33+11+3+1+0
148

e)
```
int f5(int x)
{
    if (x > 0)
       return x * f5(x-1);
    else
       return 0;
}
...
cout<< f5(3) <<endl;
```
3*2*1*0
0

f)
```
void f6(int x)
{
    if (x > 0)
    {
        cout<< x <<" ";
        f6(x-1);
    }
}
...
f6(3);
```
3 2 1

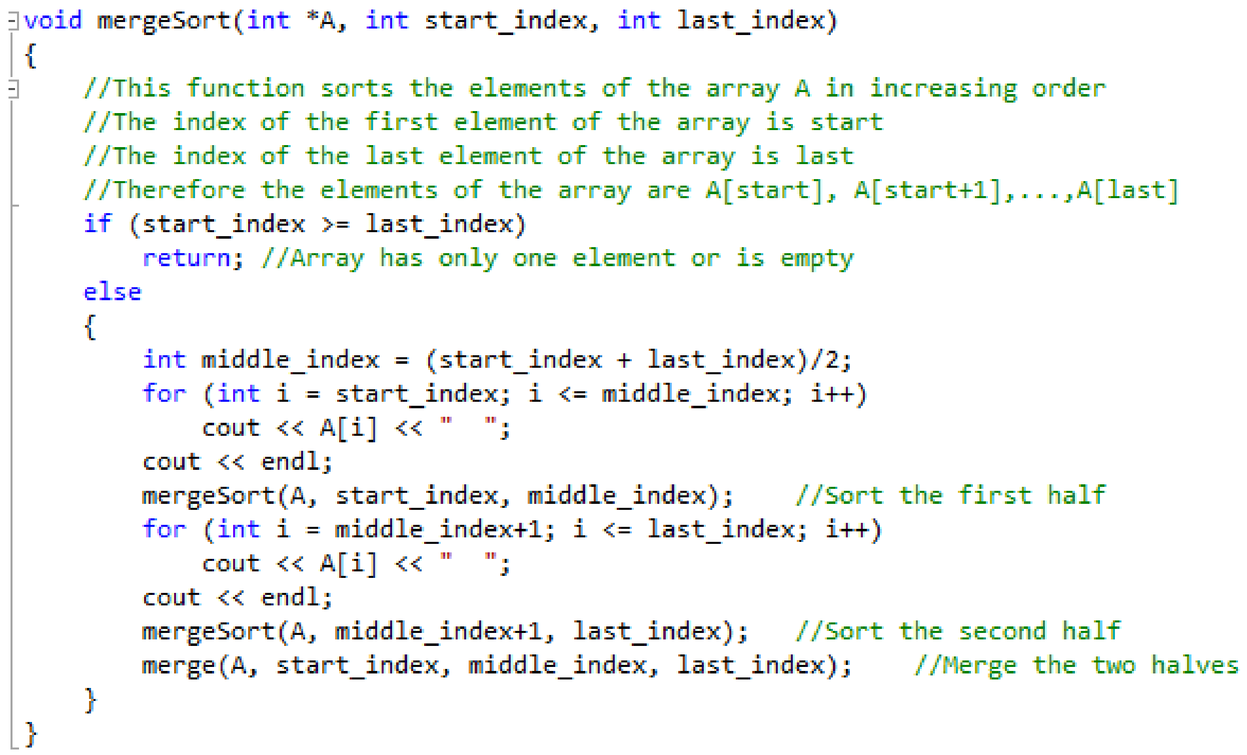
g)
```
void f7(int x)
{
    if (x > 0)
    {
         f7(x-1);
         cout<< x <<" ";
    }
}
...
f7(3);
```
1 2 3

h)
```
void f8(int x)
{
    if (x > 0)
    {
       cout<< x <<" ";
       f8(x-1);
       cout<< x <<" ";
    }
}
...
f8(3);
```
3 2 1 1 2 3

i)
```
void f9(int x)
{
    if (x > 0)
    {
        f9(x-1);
        cout<< x <<" ";
        f9(x-1);
    }
}
...
f9(3);
```
1 1 2 1 1 1 2 1
3 1 1 2 1 1 1 2
1 3

1213121

2. Write a recursive boolean function that tests if the elements of an array of integers are in increasing order. You must decide the function name and the function parameters.
3. Write a recursive boolean function that tests if all the elements of an array of integers are even numbers. You must decide the function name and the function parameters.
4. Write a recursive function that counts how many times a given integer value is found in an array of integers. You must decide the function name and the function parameters.
5. Write a recursive function returns the minimum element of an array of integers. You must decide the function name and the function parameters.
6. Write a recursive boolean function that tests if all the elements of an array of integers are distinct. You must decide the function name and the function parameters.
7. Write a recursive boolean function that tests if a given C++ string is a substring of a given another C++ string. You must decide the function name and the function parameters.
8. Write a recursive boolean function that tests if a given C++ string is a palindrome string. You must decide the function name and the function parameters.
9. Write a recursive function that returns the reverse of a C++ string argument. You must decide the function name and the function parameters.
10. Write a recursive function that returns the reverse of a given non-negative integer argument. You must decide the function name and the function parameters.
11. Write a recursive Boolean function that tests if two given C++ strings are equal. You must decide the function name and the function parameters.
12. Analyze the following code and determine its output. Assume the array parameter has the following elements [2, 7, 4, 0, 6, 1, 3] and that the merge function is given as described in the lecture.



```
void mergeSort(int *A, int start_index, int last_index)
{
    //This function sorts the elements of the array A in increasing order
    //The index of the first element of the array is start
    //The index of the last element of the array is last
    //Therefore the elements of the array are A[start], A[start+1],...,A[last]
    if (start_index >= last_index)
        return; //Array has only one element or is empty
    else
    {
        int middle_index = (start_index + last_index)/2;
        for (int i = start_index; i <= middle_index; i++)
            cout << A[i] << "  ";
        cout << endl;
        mergeSort(A, start_index, middle_index);    //Sort the first half
        for (int i = middle_index+1; i <= last_index; i++)
            cout << A[i] << "  ";
        cout << endl;
        mergeSort(A, middle_index+1, last_index);   //Sort the second half
        merge(A, start_index, middle_index, last_index);    //Merge the two halves
    }
}
```

13. Analyze the following code and determine its output. Assume the array parameter has the following elements [2, 7, 4, 0, 6, 1, 3] and that the merge function is given as described in the lecture.

```
void mergeSort(int *A, int start_index, int last_index)
{
    //This function sorts the elements of the array A in increasing order
    //The index of the first element of the array is start
    //The index of the last element of the array is last
    //Therefore the elements of the array are A[start], A[start+1],...,A[last]
    if (start_index >= last_index)
        return; //Array has only one element or is empty
    else
    {
        int middle_index = (start_index + last_index)/2;
        mergeSort(A, start_index, middle_index);    //Sort the first half
        for (int i = start_index; i <= middle_index; i++)
            cout << A[i] << "  ";
        cout << endl;
        mergeSort(A, middle_index+1, last_index);   //Sort the second half
        for (int i = middle_index+1; i <= last_index; i++)
            cout << A[i] << "  ";
        cout << endl;
        merge(A, start_index, middle_index, last_index);    //Merge the two halves
    }
}
```

14. Analyze the following code and determine its output. Assume the array parameter has the following elements [2, 7, 4, 0, 6, 1, 3] and that the merge function is given as described in the lecture.

```
void mergeSort(int *A, int start_index, int last_index)
{
    //This function sorts the elements of the array A in increasing order
    //The index of the first element of the array is start
    //The index of the last element of the array is last
    //Therefore the elements of the array are A[start], A[start+1],...,A[last]
    if (start_index >= last_index)
        return; //Array has only one element or is empty
    else
    {
        int middle_index = (start_index + last_index)/2;
        mergeSort(A, start_index, middle_index);    //Sort the first half
        for (int i = start_index; i <= middle_index; i++)
            cout << A[i] << "  ";
        cout << endl;
        mergeSort(A, middle_index+1, last_index);   //Sort the second half
        for (int i = middle_index+1; i <= last_index; i++)
            cout << A[i] << "  ";
        cout << endl;
        merge(A, start_index, middle_index, last_index);    //Merge the two halves
        for (int i = start_index; i <= last_index; i++)
            cout << A[i] << "  ";
        cout << endl;
    }
}
```

15. Write a recursive C++ function that takes an array of double data type, its start index, and its last index and that returns the largest element of the array.

16. Write a recursive C++ function that takes an array of double data type, its start index, and its last index and that returns the smallest element of the array.

17. Write a recursive C++ function that takes an array of integer data type, its start index, its last index, and an integer search value and that counts and returns the number of elements in the array whose value is equal to the search value argument.

18. Given the following test main program, provide the recursive **bubble sort** function so that the program works correctly. Your bubble sort algorithm must be recursive. Thus you need to first come up with the algorithm, identify the base case and finally identify the recursive step.

```
int main()
{
	int size = 6;
	int *A = new int [size];
	for (int i = 0; i < size; i++)
		A[i] = rand() % 25;
	cout << "Original Array\n\t";
	for (int i = 0; i < size; i++)
		cout << A[i] << "  ";
	cout << endl;
	bubbleSort(A, 0, size-1);
	cout << "Sorted Array Array\n\t";
	for (int i = 0; i < size; i++)
		cout << A[i] << "  ";
	cout << endl;
	system("Pause");
	return 0;
}
```

19. Given the following test main program, provide the recursive **selection sort** function so that the program works correctly. Your selection sort algorithm must be recursive. Thus you need to first come up with the algorithm, identify the base case and finally identify the recursive step.

```
int main()
{
	int size = 6;
	int *A = new int [size];
	for (int i = 0; i < size; i++)
		A[i] = rand() % 25;
	cout << "Original Array\n\t";
	for (int i = 0; i < size; i++)
		cout << A[i] << "  ";
	cout << endl;
	selectionSort(A, 0, size-1);
	cout << "Sorted Array Array\n\t";
	for (int i = 0; i < size; i++)
		cout << A[i] << "  ";
	cout << endl;
	system("Pause");
	return 0;
}
```

20. Given the following test main program, provide the recursive **insertion sort** function so that the program works correctly. Your insertion sort algorithm must be recursive. Thus you need to first come up with the algorithm, identify the base case and finally identify the recursive step.

```
int main()
{
	int size = 6;
	int *A = new int [size];
	for (int i = 0; i < size; i++)
		A[i] = rand() % 25;
	cout << "Original Array\n\t";
	for (int i = 0; i < size; i++)
		cout << A[i] << "  ";
	cout << endl;
	insertionSort(A, 0, size-1);
	cout << "Sorted Array Array\n\t";
	for (int i = 0; i < size; i++)
		cout << A[i] << "  ";
	cout << endl;
	system("Pause");
	return 0;
}
```

21. Given the following test main program, implement the Tower's of Hanoi algorithm and so that the program works correctly. In your function, you must have cout statements describing the moves required.

```
int main()
{
	int n;
	do
	{
		cout << "Enter the number of discs (positive number please): ";
		cin >> n;
	} while (n <= 0);
	char sourcePeg = 'A';
	char temporaryPeg = 'B';
	char destinationPeg = 'C';
	towersOfHanoi(n, sourcePeg, temporaryPeg, destinationPeg);

	system("Pause");
	return 0;
}
```

22. Implement the **power(x, y)** function that computes **x^y** recursively. Assume both x and y are integers, x is never equal to zero and that y is greater or equal to zero. Write a test main program to test your function.

23. [Challenge] Given an array of characters that contains distinct elements; write a function named **printPermutations** that prints all the permutations (i.e. all different arrangements) of the elements of the array. This is beyond the scope of the course and you don't have to do it. If you want to attempt it, first read on the concept of Dynamic Programming and Back Tracking Algorithms.

24. [Challenge] Consider the following test main program

```
int main()
{
	int n;
	do
	{
		cout << "Enter the number of elements (positive number please): ";
		cin >> n;
	} while (n <= 0);

	cout << "All the subsets of the set {1,2,3,...,n} are" << endl;
	allSubsets(n);

	system("Pause");
	return 0;
}
```

Write the function **allSubsets** that takes a positive integer argument and prints all the subsets of the set given by {1, 2, 3, …, n}. This is beyond the scope of the course and you don't have to do it. If you want to attempt it, first read on the concept of Dynamic Programming and Back Tracking Algorithms.

25. [Challenge] Consider the following test main program

```
int main()
{
	const int n = 9;
	int **sudoku = new int*[n];
	for (int i = 0; i < n; i++)
		sudoku[i] = new int[n];

	fillSudokuBoard(Sudoku, n);

	cout << "An example an nxn sudoku board such that" << endl;
	cout << "\tEach row contains the integers 1,2,3,...,n" << endl;
	cout << "\tEach column contains the integers 1,2,3,...,n" << endl;
	cout << "is the following" << endl;

	for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < n; j++)
			cout << sudoku[i][j] << "    ";
		cout << endl;
	}

	system("Pause");
	return 0;
}
```

Write the function **fillSudokuBoard** that will fill the board as described. This is beyond the scope of the course and you don't have to do it. If you want to attempt it, first read on the concept of Dynamic Programming and Back Tracking Algorithms.