CSCI 2270 Midterm 2

Important notes:

1) For each header file you only need to implement the member functions needed to complete the task.
2) Include ALL the source files you used (.cpp, .hpp) in your solutions.
3) We strongly encourage you to use the provided hpp files. If you do decided to make any changes to it make sure to explain this in this document.

Read these directions carefully.
On Moodle there is a quiz with **6** multiple-choice problems. Submit answers for each of those.

In addition to the multiple choice, you will be solving two of the three programming problems detailed below. *Problem 1* is **mandatory**, but you only have to do **either** *Problem 2* **or** *Problem 3*. For each problem you must submit three documents:

1. A C++ program that solves the given problem.
2. A text file that contains the output of your program when it is run.
3. A short document that contains any issues or concerns you have about the given problem, as well as any information that we need to understand, compile, or run your solution.

Your submission should be valid C++ 11 code. The solutions should use similar types and functions to those in the given starter code, but you are welcome to make modifications as you see fit. We will be running this code on other computers, so make sure to avoid **any** undefined behavior such as uninitialized variables.

Problem 1 (Mandatory)

Task:

Given a Binary Search Tree, return the sum of values of all the nodes that fall within a range [min, max] inclusive.

Requirements:

Implement a `sumRange` function that rereturns the sum of all values in a given range. If no node falls within that range or the tree is empty, return 0.

```
int sumRange(int min, int max); // example declaration
```

Examples:

- Calling `sumRange(8,12)` on the tree below should return `28`.

```
          9
        /   \
       7     16
     /  \    /
    4    8  11
```

- Calling `sumRange(8,10)` on the tree below should return `0`.

```
      4
    /   \
  2      7
```

Use the starter code in Moodle and the implementations you have prepared. Write a main function that creates a BST of ints, then calls the function defined above to demonstrate the method. Call it on multiple trees to show that it works in all cases, and print out the results.

Problem 2

Task:

Given an unsorted array, sort it using the functionality of a heap.

Requirements:

Implement an `heapSort` function to sort an array of floats. Your algorithm must not do any swaps outside of those used implicitly in the heap operations and should have complexity no worse than O(nlogn).

```
float *  heapSort(float arr[]); // example declaration
```

Examples:

- If `arr = {1.2,10.5,5,15,20.7}` then `heapSort(arr)` should return `arr = {1.2,5,10.5,15,20.7}`
- If `arr = {5}` then `heapSort(arr)` should return `arr = {5}`
- If `arr = {30,8.2,34}` then `heapSort(arr)` should return `arr = {8.2,30,34}`

Use the starter code in Moodle and the implementations you have prepared. Write a main function that creates an array of integers, then calls the function defined above to demonstrate the method. Call it on multiple arrays to show that it works in all cases, and print out the results.

Problem 3

Task:

Given a graph and a starting vertex, count the number of nodes a specified distance away.

Requirements:

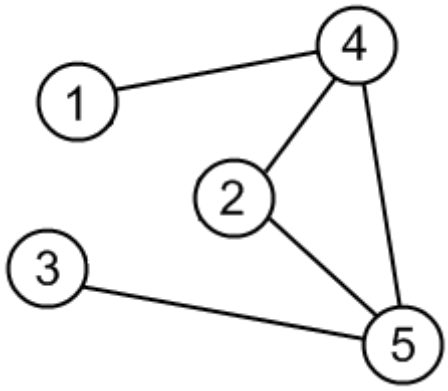
Create a graph of int's. Given a starting node with value `key` and a distance `dist`, return the number of nodes that have a path from `id` with exactly `dist` edges. If there are multiple paths, only consider the shortest one.

```
int countNodesWithDist(int id, int dist); // example declaration
```

Examples:

- For the graph below, `countNodesWithDist(2, 1)` should return `2` since there are two nodes 1 step away from 2 (i.e. 4 and 5).

- `countNodesWithDist(3, 2)` should return `2` since there are two nodes 2 steps away from 3 (i.e. 2 and 4).



Use the starter code in Moode and the implementations you have prepared. Write a main function that creates a graph and demonstrates the method above. Make sure your graph covers as many edge case scenarios as you can think of and call the method with multiple combinations of starting vertex and distance to show that it works in all cases, and print out the results.