NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

Practical Examination (PE)

CS2040C Data Structure and Algorithm

8 Apr 2023 Time Allowed: 2 hours

VERY IMPORTANT:

- Please make sure your code can be compiled and runnable after you submitted to Couresmology. Any crashing code will be zero mark.
    - You only have 10 chances of submission to each question. So please test out your code in MSVC first before running them on coursemology. Do not use coursemology as a debugger to test your code.
- You should stop modifying your code and start submitting 10 min before the end of the PE in order to avoid the "jam" on Coursemology. Actually, you should submit a version after every time you finished a task. It's your own responsibility to submit the code to Coursemology on time.
- You cannot add global variables. You cannot include other extra libraries such as STL.
- It's your own responsibility to make sure that you have submitted the right code on time.
- The PE consists of three parts. And in each part, you will be provided with an MSVS project file. In each project file, there will be one .h file for submission. You should modify and submit that file only by "select all" (ctrl-A) in that file and copy and paste it to the corresponding part in Coursemology. Your submission should be able to be complied and run under the assumption that all your project is the same as the original project you downloaded with the main.cpp.

INSTRUCTIONS

1. **You will be forced to log out at the time the PE ends SHARPLY**. Make sure you save and submit your work a few minutes before it ends. The network will be jammed at the last few min of the assessment because everyone is submitting at the same time.
2. ***Your program must be able to be compiled!*** Or you will receive zero mark for that part.
3. Any variables used must be declared within some functions. You are not allowed to use global variables (variables that are declared outside all the functions). Heavy penalty will be given (see below) if you use any global variable.

Advice

- Manage your time well! Do not spend excessive time on any task.
- Read all the questions and plan before you start coding.
- Please save and backup your code regularly during the PE.
- It is a bad idea to do major changes to your code at the last 10 minutes of your PE.

Part 1: Sorting Linked Lists (40 marks)

You are not allowed to include extra library or packages in this part.

In this problem, we will be implementing Bubble Sort and Merge Sort on a Single Linked List, implemented similarly with your assignment. A zipped file of the solution files for MS Visual Studio is provided which contains:

- The Linked List class.
  - `linkedlist.h`
- The main driver class containing test codes.
  - `main.cpp`

For submission, please copy and paste your whole file of "`linkedlist.h`" to Coursemology.

Part 1 Task 1: Implementing Bubble Sort

Given a single linked List, the function `bubbleSort()` sorts the linked list using bubble sort algorithm. This prints the entire linked list after each pass in the outer repetition statement (for-loop / while-loop).

```
Part Task 1: Bubble Sort
============================
Current List with 10 elements: 90 80 70 60 50 40 30 20 10 0

Sorting the list using bubbleSort:
80 70 60 50 40 30 20 10 0 90
70 60 50 40 30 20 10 0 80 90
60 50 40 30 20 10 0 70 80 90
50 40 30 20 10 0 60 70 80 90
40 30 20 10 0 50 60 70 80 90
30 20 10 0 40 50 60 70 80 90
20 10 0 30 40 50 60 70 80 90
10 0 20 30 40 50 60 70 80 90
0 10 20 30 40 50 60 70 80 90

Sorted List with 10 elements: 0 10 20 30 40 50 60 70 80 90


Current List with 5 elements: 40 10 30 20 -20

Sorting the list using bubbleSort:
10 30 20 -20 40
10 20 -20 30 40
10 -20 20 30 40
-20 10 20 30 40

Sorted List with 5 elements: -20 10 20 30 40
```

Part 1 Task 2: Implementing splitting of linked list

The function `split()` divides the linked list into two linked lists. This function takes in a List pointer `otherList`. After splitting, the first half of the initial linked list will be pointed by the list that calls the function, and `otherList` will point to the second half. You can assume `otherList` will be always empty initially. You can also assume the initial list contains at least two nodes. You have to keep the elements in the same order as the original linked list. For example, the test function calls `firstList.split(&secondList)` which results in the following output.

```
Part 1 Task 2: Split
=======================
List with even number of elements.
First List with 10 elements: 90 80 70 60 50 40 30 20 10 0

Second List with 0 elements:

Splitting the list into firstList and secondList.

First List: 90 80 70 60 50

Second List: 40 30 20 10 0
```

If the initial list has odd number of elements, you should split the list into two list that the first list with fewer elements.

```
List with odd number of elements.
Third List with 9 elements: 80 70 60 50 40 30 20 10 0

Fourth List with 0 elements:

Splitting the list into thirdList and fourthList.

Third List: 80 70 60 50

Fourth List: 40 30 20 10 0
```

Part 1 Task 3: Implementing merging of linked list

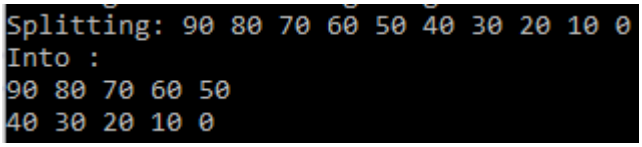
The function `merge()` takes in a List pointer `otherList`, and combines two non-empty sorted linked lists in ascending order into one linked list. After combining, the merged linked list is sorted in ascending order. You can assume both lists are non-empty initially. The `otherList` should be empty after merging. For example, the test function calls `firstList.merge(&secondList)` which results in the following output.

```
Part 1 Task 3: Merge
=======================
First List with 5 elements: 10 30 50 70 90

Second List with 5 elements: 0 20 40 60 80

Merging the two lists together.

Merged list with 10 elements: 0 10 20 30 40 50 60 70 80 90

Second list with no elements:
```

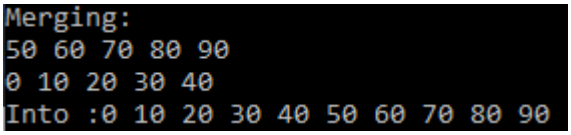
Part 1 Task 4: Implementing Merge Sort

Write the function `mergeSort()` to sort the List in ascending order using merge sort algorithm. Make use of your implementation in the previous task to implement this function. However, you need to show your steps in your `mergeSort()`.

- Whenever you split the list into two, you need to print



- Whenever you merge two sorted lists into one, you need to print



The right box shows the full output of what you should have for a complete `mergeSort()`. The wordings for printing are given to you as `SPLITSTR`, etc.

```
Splitting: 90 80 70 60 50 40 30 20 10 0
Into :
90 80 70 60 50
40 30 20 10 0
Splitting: 90 80 70 60 50
Into :
90 80
70 60 50
Splitting: 90 80
Into :
90
80
Merging:
90
80
Into :80 90
Splitting: 70 60 50
Into :
70
60 50
Splitting: 60 50
Into :
60
50
Merging:
60
50
Into :50 60
Merging:
70
50 60
Into :50 60 70
Merging:
80 90
50 60 70
Into :50 60 70 80 90
Splitting: 40 30 20 10 0
Into :
40 30
20 10 0
Splitting: 40 30
Into :
40
30
Merging:
40
30
Into :30 40
Splitting: 20 10 0
Into :
20
10 0
Splitting: 10 0
Into :
10
0
Merging:
10
0
Into :0 10
Merging:
20
0 10
Into :0 10 20
Merging:
30 40
0 10 20
Into :0 10 20 30 40
Merging:
50 60 70 80 90
0 10 20 30 40
Into :0 10 20 30 40 50 60 70 80 90
```

Part 2 Order Statistics of BST (30 marks)

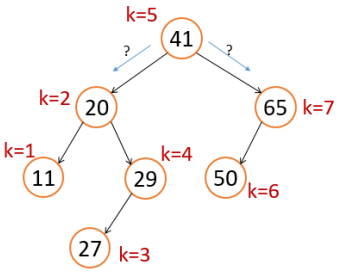
You are not allowed to include extra library or packages in this part.

A zipped file of the solution files for MS Visual Studio is provided which contains:

- The BST class similar to our assignment.
  - `BST.h`
- The main driver class containing test codes.
  - `main.cpp`

For submission, please copy and paste your whole file of "`BST.h`" to Coursemology. Do NOT modify the `print()` function or you will suffer from different output comparison in Coursemology.

In this task, we will be augmenting the BST with the rank function and implementing it to cater for integers. You will only get full marks for each task if your BST is height balanced (AVL).



11	20	27	29	41	50	65

A node in a Binary Search Tree has rank k if it's the kth element in the ascending order of all the elements. For example, this is the example we showed in class on the right.

Task 1 Maintaining the Weights.

The weight of each node is the total number of nodes in that subtree. You can see the `print()` function also print out the weight now. Here is a sample output after you maintained the weights (left below).

```
            14(h=0,w=1)
        13(h=1,w=3)
            12(h=0,w=1)
    11(h=2,w=7)
            10(h=0,w=1)
        9(h=1,w=3)
            8(h=0,w=1)
7(h=3,w=15)
            6(h=0,w=1)
        5(h=1,w=3)
            4(h=0,w=1)
    3(h=2,w=7)
            2(h=0,w=1)
        1(h=1,w=3)
            0(h=0,w=1)
```

```
The item with rank 1 is 0
The item with rank 2 is 2
The item with rank 3 is 4
The item with rank 4 is 6
The item with rank 5 is 8
The item with rank 6 is 10
The item with rank 7 is 12
The item with rank 8 is 14
The item with rank 9 is 16
The item with rank 10 is 18
The item with rank 11 is 20
The item with rank 12 is 22
The item with rank 13 is 24
The item with rank 14 is 26
The item with rank 15 is 28
```

Task 2 Select()

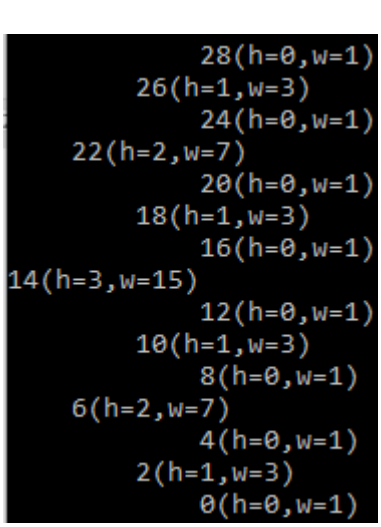
Implement the function `T select(int rank)` to return the kth element in the tree. You will only get full marks if your tree is balanced. If there are n elements in the tree, you can assume your input rank is between 1 and n. For the give tree above, the output are shown in the right above.

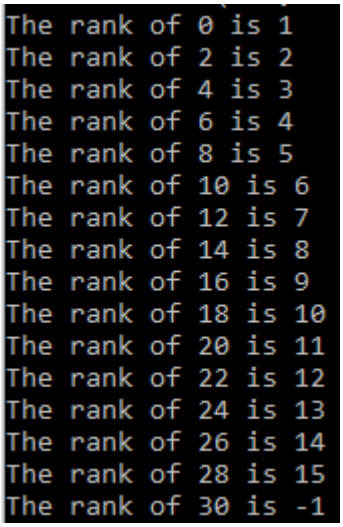
Task 3 Rank()

Implement the `int rank(T item)` function that will return:

- The rank of the node containing item, if such a node exists.
- -1, if such a node does not exist.

Here are the example output of sample code:





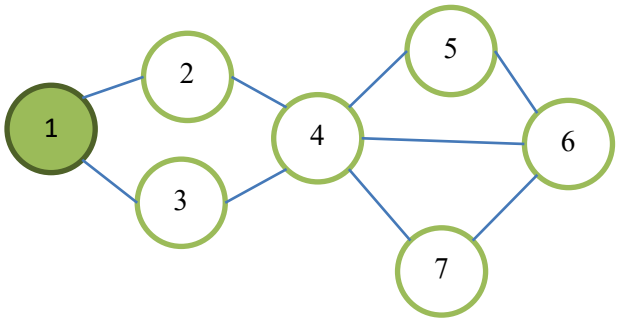
You may assume that all the nodes in the binary search tree have unique values.

Part 3 Party or no party? That is the question (30 marks)

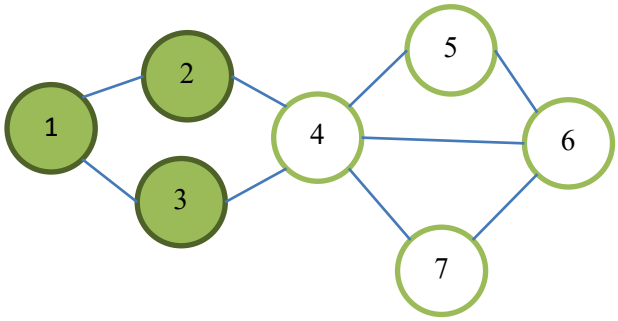
In a large class (with the number of students = A), not every classmate knows each other. And the professor invited the whole class to his party! When the professor asked in the lecture personally, only one guy (Student D) raised up his hand and agreed to go! The rest of the students are too shy and indecisive on the spot, but here is the catch:

> **If any student know that more than or equal to half of his/her known friends are going, he/she will decide to go!**

For example, if there are 7 students, ranging from Student 1 to Student 7 (Student numbers starts from 1). And their friendship are connected in the following graph

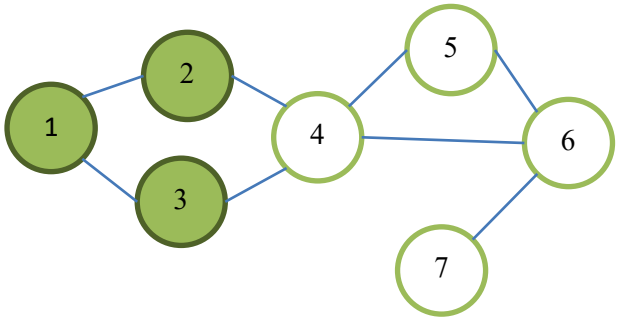


Let's say, Student 1 is the first one who raised his hand in the lecture. Then for Students 2 and 3 they are influenced to go to the party because each of them has two friends and "half" of their friends wanted to go (thanks to Student 1). However, now Students 4 doesn't want to go because he has 5 friends (Students 2, 3, 5, 7, 6) and only two of his friends (Student 2 and 3) wanted to go. Then the spread stops here.



Finally, only Students 1, 2 and 3 wants to go to the party.

However, for a different situation, if Student 4 didn't know Student 7 from the beginning.



Then Student 4 will go because he has only 4 friends and two of them wanted to go. Then the next one is Student 5 and he will be influenced by Student 4. After Students 4 and 5 wanted to go, Student 6 will go. And finally, Student 7 will go. Meaning ALL the students will go!

Task

The students' relationships will be given in a text file for you to read in. The file looks like the box on the right. The first line of the file recorded 4 integers `A, B, C, D`:

```
7 9 5 1
1 2
1 3
2 4
3 4
4 5
4 6
4 7
```

- `A`: The total number of students
- `B`: The total number of friendship
- `C`: You!
- `D`: The first student who indicated he wanted to go

And then, it followed by `B` lines of two integers `X` and `Y` and it means that Student `X` is a friend of Student `Y`.

Complete the function `partyGoing(filename)` to read in a file with `filename` and print out

- If you are Student `C`, will you go?
- All the classmates who are going to the party in the ascending order of their student number

Here is two outputs for two example files:

```
For the file input of "part3input0.txt".
5 will not go for the party
Classmate(s) who will go to the party is/are:
1 2 3
For the file input of "part3input0b.txt".
5 will go for the party
Classmate(s) who will go to the party is/are:
1 2 3 4 5 6 7
```

Please use the given strings output to match your expected answers in Coursemology.