CS2040C Data Structure and Algorithm PE
Time Allowed: 2 hours

General

- 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 5 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.

Part 1 Circular Linked List (50 marks)

The Circular Linked List is a useful data structure for many applications. A circular linked list is a linked list that the last node has its next pointer to the first node in the list, instead of NULL.

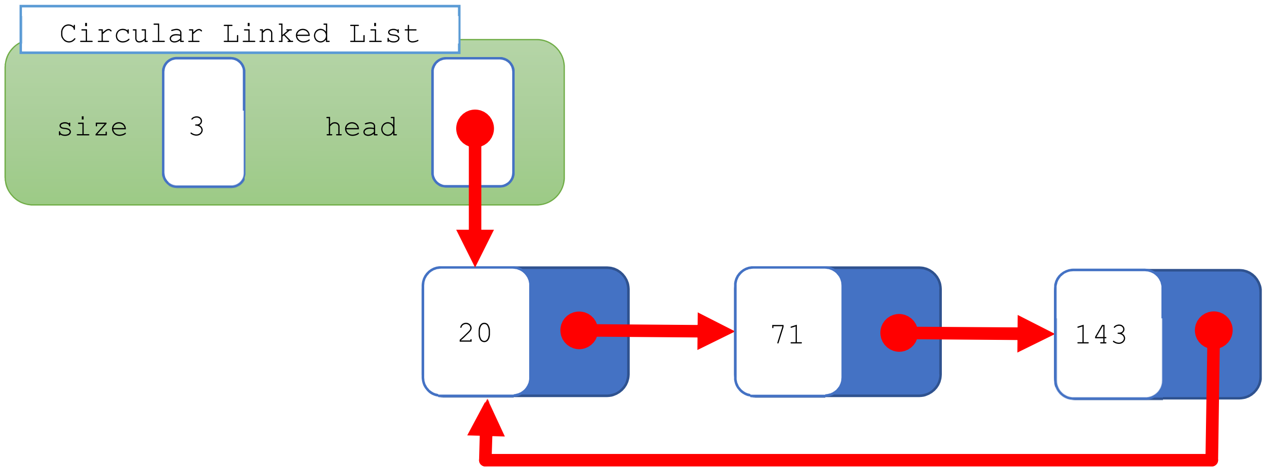


Figure 1: An example of a Circular Linked List

And the head can be "advanced", namely, move to the next node. For example, we can advance the head of the above like this:

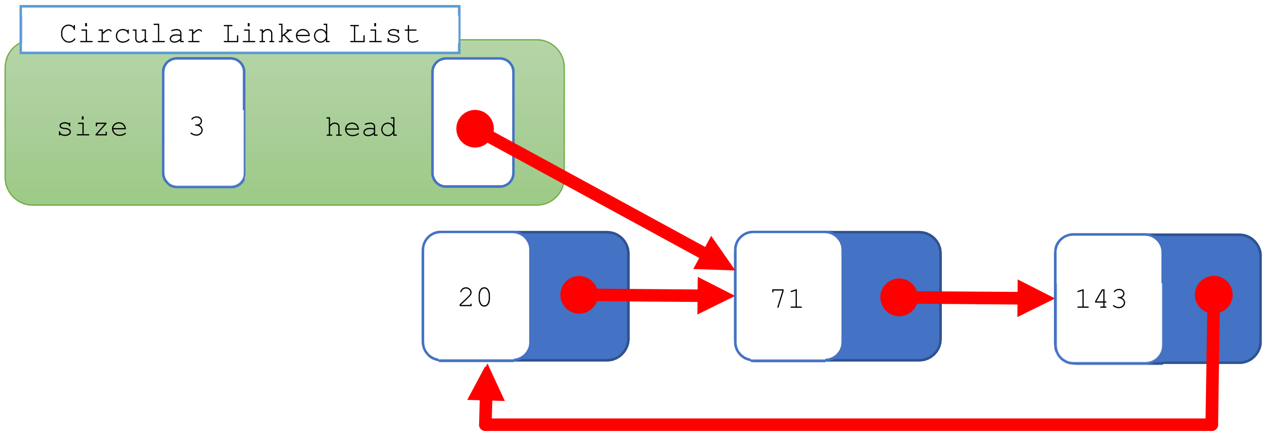


Figure 2: After 1 "Advance Head" from the list in Figure 1

And we can keep advancing forever.

One of the important applications is scheduling tasks in a multitasking operating system. All the running applications are kept in a circular linked list and the OS gives a fixed time slot to all for running. The Operating System keeps on iterating over the linked list until all the applications are completed. A new task will be added to the linked list when the user started one. And a task will be removed from the list when the task is done.

Project/File Settings

You will receive the following files in your project file:

- `circularIntLinkedList.{h,cpp}` and `main.cpp`

Your Tasks

Before you started, please bear in mind that **you will only get full marks if all of your functions have a time complexity of O(1)** except `print()` and `exist()`. (Hint: However, the *trick* to achieve O(1) without adding new attribute is not following the implementation in Figures 1 and 2 exactly.) The followings are the requirements for the functions.

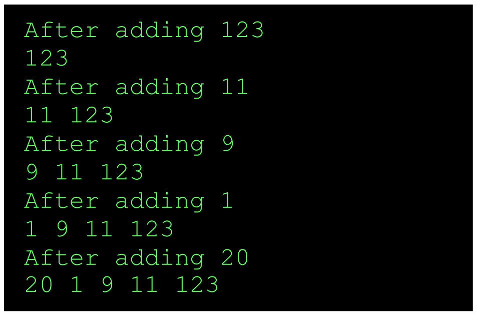
Your tasks in this part is to implement the following seven functions. You should submit all of the following 7 functions and 7 only to coursemology :

- `void CircularList::insertHead(int n)`
- `void CircularList::advanceHead()`
- `void CircularList::removeHead()`
- `void CircularList::print()`
- `bool CircularList::exist(int n)`
- `int CircularList::headItem()`
- `CircularList::~CircularList()`

You should not modify `circularIntLinkedList.h`. Namely, you cannot add/remove/change the definitions, attributes or member functions of the given classes. However, you can modify `main.cpp` for your own testing. And you should not use any extra global variables. Make it short, you should ONLY submit the above seven member functions only with their bodies implemented.

`insertHead(int n)`

Insert the integer n into the head/front of the list. Here is the sample output for the test function given on the right.



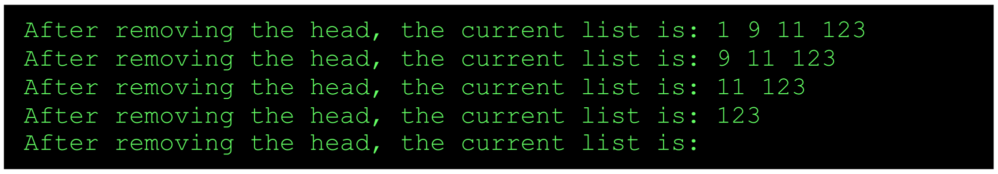
```
After adding 123
123
After adding 11
11 123
After adding 9
9 11 123
After adding 1
1 9 11 123
After adding 20
20 1 9 11 123
```

`print()`

The function should print the list in one line and end with a new line. There is no leading space before the line but there is a single space after each integer printed, including the last integer. E.g. the second printout above is "`11 123 `". And finally, print a new line at the end, e.g. endl.

removeHead();

It will remove the first item currently at the head. If the list is empty, the function will just do nothing.



```
After removing the head, the current list is: 1 9 11 123
After removing the head, the current list is: 9 11 123
After removing the head, the current list is: 11 123
After removing the head, the current list is: 123
After removing the head, the current list is:
```

exist(int n):

The function will return true if the integer n is in the list, and return false otherwise.

headItem();

This function will return the integer at the head of the list currently. You can assume that we will call this function only if the list is not empty.

advanceHead()

The function will move make the item next to the current head to be the head of the list. The previous head will be moved to the end of the list. Please see Figures 1 and 2 for an example. The function should work even if the list is empty.

```
The original list:20 1 9 11 123
After 1 "advance"
The current list is: 1 9 11 123 20
After 2 "advance"
The current list is: 9 11 123 20 1
After 3 "advance"
The current list is: 11 123 20 1 9
After 4 "advance"
The current list is: 123 20 1 9 11
After 5 "advance"
The current list is: 20 1 9 11 123
```

~CircularList()

The destructor of the class should remove all the items of the list before the list is destroyed.

Submission

Please paste the above seven functions into “Question 1: Circular Linked LIst (Part 1)” in coursemology.

Sample Output:

If you uncomment all the functions in `main()`, you should have following outputs

```
insertHeadtest()
After adding 123
123
After adding 11
11 123
After adding 9
9 11 123
After adding 1
1 9 11 123
After adding 20
20 1 9 11 123
```

```
insertHeadtest()

existTest()
The list is: 20 1 9 11 123
Does 9 exist in the list? Yes

Does 11 exist in the list? Yes

Does 99 exist in the list? No
```

```
advanceHeadtest()
The original list:20 1 9 11 123
After 1 "advance"
The current list is: 1 9 11 123 20
After 2 "advance"
The current list is: 9 11 123 20 1
After 3 "advance"
The current list is: 11 123 20 1 9
After 4 "advance"
The current list is: 123 20 1 9 11
After 5 "advance"
The current list is: 20 1 9 11 123
```

```
removeHeadtest()
After removing the head, the current list is: 1 9 11 123
Does 9 exist in the list? Yes

After removing the head, the current list is: 9 11 123
Does 9 exist in the list? Yes

After removing the head, the current list is: 11 123
Does 9 exist in the list? No

After removing the head, the current list is: 123
Does 9 exist in the list? No

After removing the head, the current list is:
Does 9 exist in the list? No

removeAndAdvanceHeadTest()
The current list is: 20 1 9 11 123
Does 9 exist in the list? Yes

Now we remove the head and advance it once
The current list is: 9 11 123 1
Does 9 exist in the list? Yes

Now we remove the head and advance it once
The current list is: 123 1 11
Does 9 exist in the list? No

Now we remove the head and advance it once
The current list is: 11 1
Does 9 exist in the list? No

Timing Test
Time taken for adding 1 items : 0s
Time taken for advancing three quarters of the list: 0s (Head item = 0)

Time taken for adding 10 items : 0s
Time taken for advancing three quarters of the list: 0s (Head item = 2)

Time taken for adding 100 items : 0s
Time taken for advancing three quarters of the list: 0s (Head item = 24)

Time taken for adding 1000 items : 0s
Time taken for advancing three quarters of the list: 0s (Head item = 249)

Time taken for adding 10000 items : 0.002s
Time taken for advancing three quarters of the list: 0s (Head item = 2499)

Time taken for adding 100000 items : 0.019s
Time taken for advancing three quarters of the list: 0.002s (Head item = 24999)

Time taken for adding 1000000 items : 0.19s
Time taken for advancing three quarters of the list: 0.02s (Head item = 249999)
```

Note that the timing for the final part may be different. However, **everything should be able to finish within 1s**.

Part 2 BST with Statistics (40 marks)

Basically similar to Assignment 3. And the content of each tree node will contain integers (or any extended integers like `long long int`) only. You are given some skeleton code, but there are a few differences from Assignment 3:

- the BST class is all in one single file, BST.h, including all the definitions and bodies of the member functions
- If you initialize the class with "`BinarySearchTree<int> bsti(true);`" the tree should be auto-balanced by the AVL tree we taught according to our lectures. However, if the parameter is false, the tree will not do any rotation/balancing after insertion.
- You can assume all the item inserted into the trees are unique from here to the end of this PE. Namely, there will be no duplicates inserted into the tree
- You do not need to implement deletions
- There is a new attribute in the `TreeNode`, namely "`T _sst`". There is the gist of this question. Make it short, `_sst` should store the sum of all items in its subtree (abv. SST) , including itself.
- The `printTree()` function is given and similar to Assignment 3, except that it will print out the value of `_sst` of each node also

It is possible to finish some of the tasks from scratch, namely, without balancing. However, you are allowed to reference to your submitted Assignment 3 for the balancing and tree rotations. If you haven't finished that part in your Assignment 3, we will suggest you try to finish the tasks from scratch without balancing. For the students who have finished Assignment 3, we suggest you to finish reading questions in Parts 2 and 3 first before you proceed, and make plan on how to merge your previous code into this question.

You will find some "Pancake" code there in the skeleton. You can ignore it for Part 2 because those are supposed for Part 3. (Although it's no harm to paste the code into coursemology.)

You will only gain full marks in this question only if your code is efficient and able to handle large sets of data.

Submission

Simply copy and paste your full file of `BST.h` into "Question 2: Question 2 BST with Statistics (Part 2)" in coursemology. Namely, paste every code you need except the `main()` function.

Here are the task you need to do. Please do plan carefully. We will put all the sample output after all the tasks.

Task 1 Tree Balancing

You may skip this step if you are not confident with it. When the tree is created, there is a parameter given to the constructor to indicate if the tree should be auto-balanced by the AVL tree algorithm in our lecture slides as mentioned above.

Task 2 Sum of Subtree in a Node (SST)

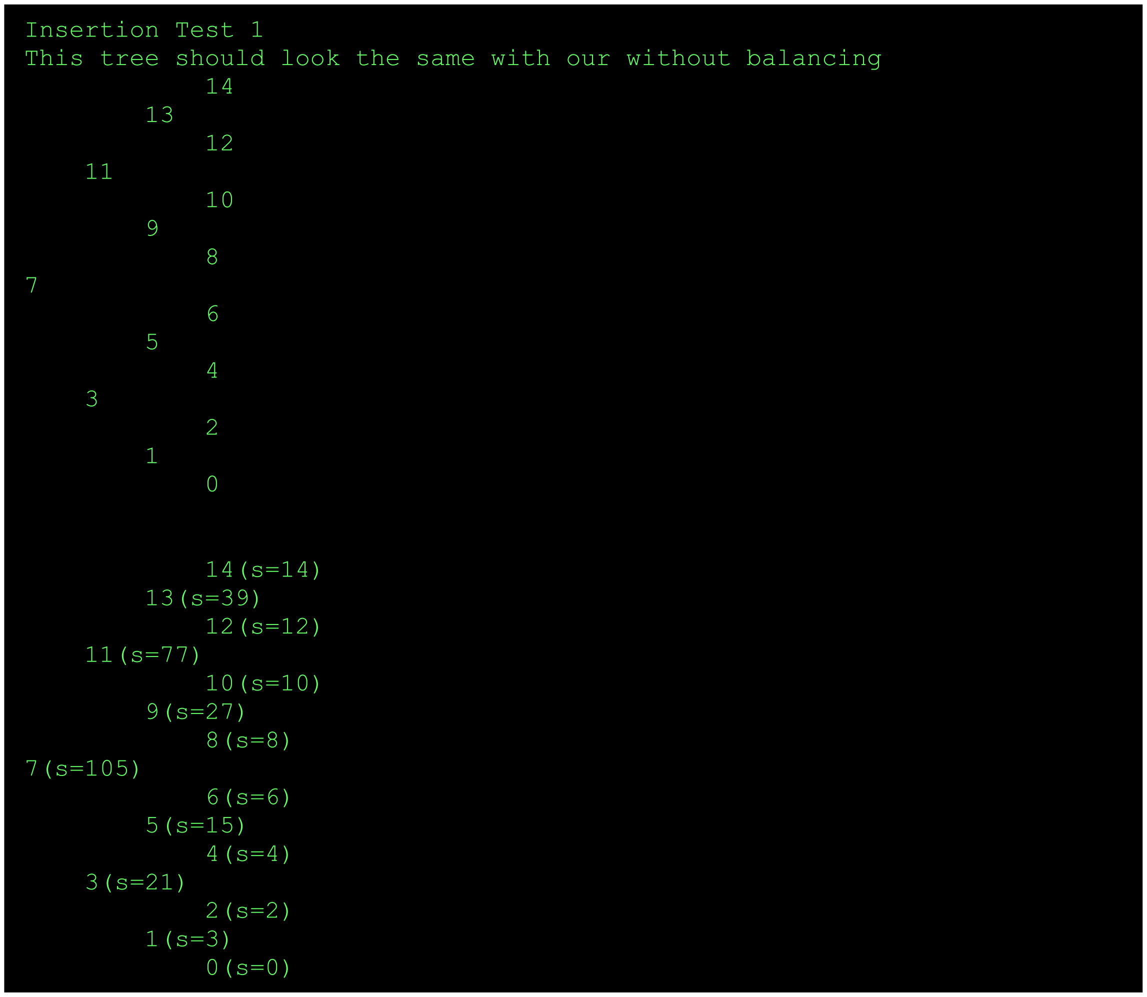
In each node, there is a new member called `_sst`. It is supposed to store the sum of all the nodes in its subtree including itself. Your job is to maintain this `_sst` after each insertion. Here is the correct sum for the test code in the given skeleton file.

Task 3 Traversal with SST

As in Assignment 3, you are given the pre-order traversal and please finish the post- and in-order traversals. Moreover, like the given code, you are required to print the `_sst` as well in a pair of parentheses immediate after the node (no space between them).

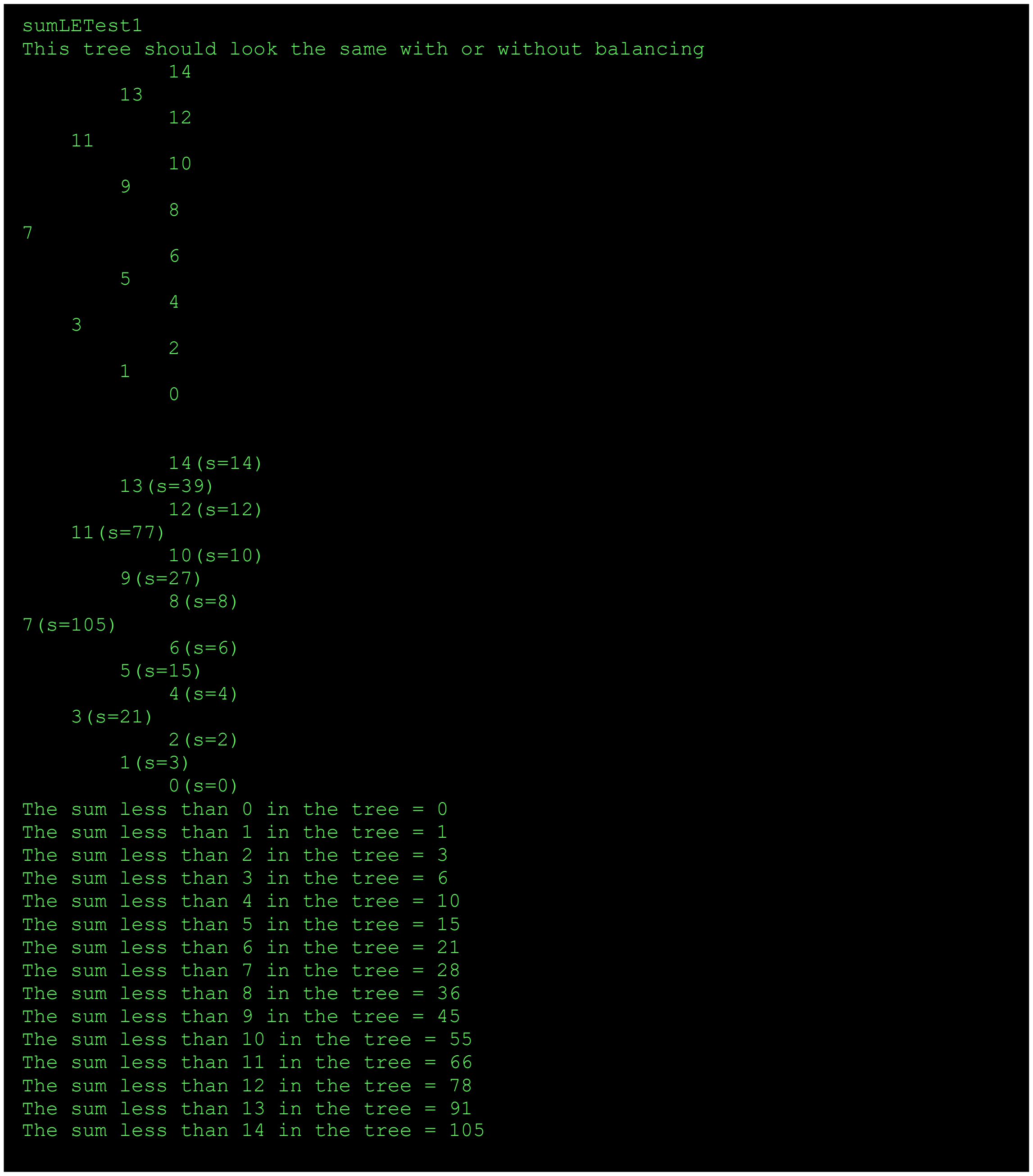
Task 4 Compute the Sum of all the Numbers Less than or Equals to n

Compute the function `sumLE(int n)` such that it will return the sum of all the numbers in the tree that is less than or equals to n.



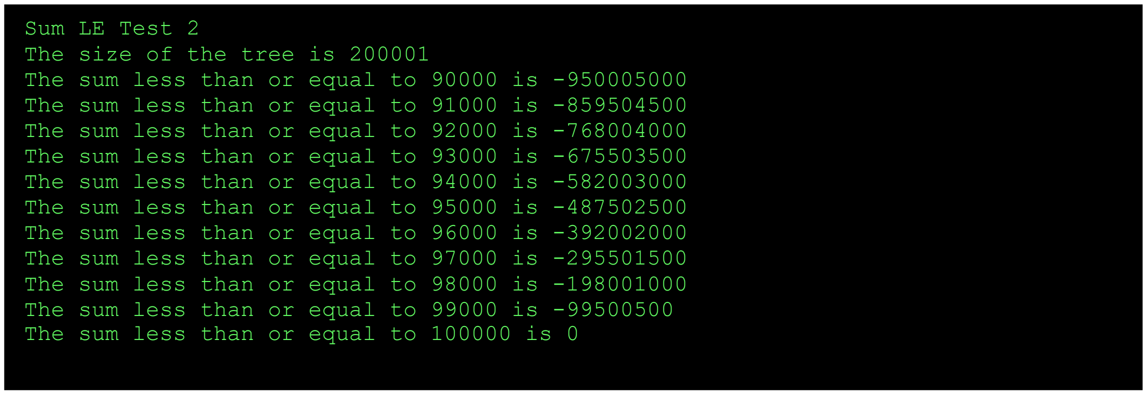
```
Insertion Test 1
This tree should look the same with our without balancing
            14
        13
            12
    11
            10
        9
            8
7
            6
        5
            4
    3
            2
        1
            0


            14(s=14)
        13(s=39)
            12(s=12)
    11(s=77)
            10(s=10)
        9(s=27)
            8(s=8)
7(s=105)
            6(s=6)
        5(s=15)
            4(s=4)
    3(s=21)
            2(s=2)
        1(s=3)
            0(s=0)
```



```
sumLETest1
This tree should look the same with or without balancing
            14
        13
            12
    11
            10
        9
            8
7
            6
        5
            4
    3
            2
        1
            0


            14(s=14)
        13(s=39)
            12(s=12)
    11(s=77)
            10(s=10)
        9(s=27)
            8(s=8)
7(s=105)
            6(s=6)
        5(s=15)
            4(s=4)
    3(s=21)
            2(s=2)
        1(s=3)
            0(s=0)
The sum less than 0 in the tree = 0
The sum less than 1 in the tree = 1
The sum less than 2 in the tree = 3
The sum less than 3 in the tree = 6
The sum less than 4 in the tree = 10
The sum less than 5 in the tree = 15
The sum less than 6 in the tree = 21
The sum less than 7 in the tree = 28
The sum less than 8 in the tree = 36
The sum less than 9 in the tree = 45
The sum less than 10 in the tree = 55
The sum less than 11 in the tree = 66
The sum less than 12 in the tree = 78
The sum less than 13 in the tree = 91
The sum less than 14 in the tree = 105
```



```
Sum LE Test 2
The size of the tree is 200001
The sum less than or equal to 90000 is -950005000
The sum less than or equal to 91000 is -859504500
The sum less than or equal to 92000 is -768004000
The sum less than or equal to 93000 is -675503500
The sum less than or equal to 94000 is -582003000
The sum less than or equal to 95000 is -487502500
The sum less than or equal to 96000 is -392002000
The sum less than or equal to 97000 is -295501500
The sum less than or equal to 98000 is -198001000
The sum less than or equal to 99000 is -99500500
The sum less than or equal to 100000 is 0
```

Part 3 Gym of the Magical Pancake Kingdom (10 marks)

Before you started, you will get very low mark for this problem if your code is not efficient and not using any of the cool techniques in our module or previous parts of this PE (*hint* *hint*). You will find no new code is given to you because all the test code and skeleton is in the zip file in Part 2.

The Story Setting

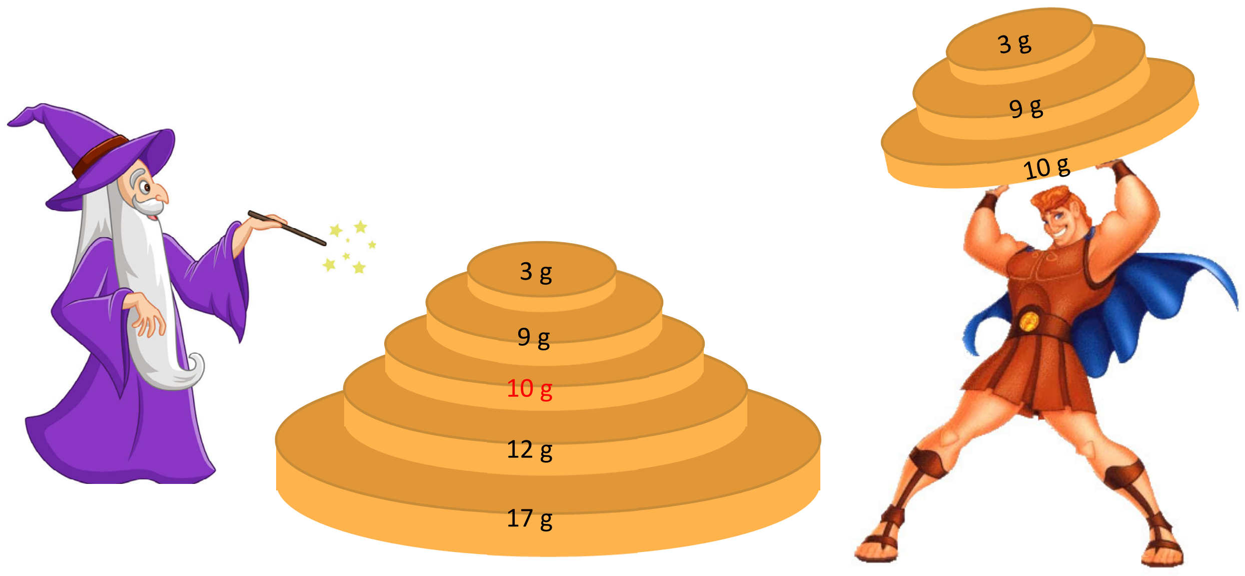
In the magical kingdom of *Pancakia*. The King *Stalan* order the wizards to help the warriors to train, with ***pancakes***! He order one wizard to pair with one warrior. And the wizard will create many pancakes with magic with the following constraints

- After the wizard create a pancake, it will be inserted into a stack of pancake sorting in descending order of weights from the bottom to the top.
- The wizard will only create pancakes with weight n grams such that n is an integer > 0, and no two pancakes will have equal weights.

And the warrior will come to train himself with weight lifting with the pancake stack! Every day, the warrior wants to lift up **at least** a stack of pancakes with a minimal weight of w grams. On the other hand, the wizard knows that weight lifting may spoil the pancakes so he will only let the warrior to weight-lift as few pancakes as possible.

Your job is, to write the function `minimalBottomPancake2Lift(int w)` to return the weight of the "highest" pancake in the stack for the warrior to lift such that the weight of the stack from that pancake upwards is greater than or equals to w grams.

Here is an example, e.g. the wizard created the pancakes in the following order: 10g, 12g, 17g, 3g, 9g. These pancakes will be stacked up according to their sizes, heaviest at the bottom to the lightest on the top. Then the warriors says, "I want to lift at least 14 g." Then he should lift the stack with bottom as 10g (the stack of 3, 9 and 10g) with a total weight of 22g. Because if he lift one fewer pancake, the stack will only weight 12g (=3g+9g) that is less than 14 g of what the warrior requested. So the function `minimalBottomPancake2Lift()` will return the "base" pancake of the stack, in which, it's "10 g" in this example. The warrior can come any time for the stack and the wizard can also add more pancakes anytime he wants. If the warrior requested 0 g, then your function should return 0 because there is no need to lift any pancake. If the warrior requested more than the total weight of ALL pancakes, your function should return -1 to indicate an error because there are not enough pancakes for him to lift.



For example, after the above training, the wizard added another 2g pancake, then if the warrior still want to lift a total of 14 g of pancakes, then he should lift from the 9g pancake instead of 10g previously because 2g+3g+9g=14g.

And bear in mind that the weight can be very large and you should represent all the weights in "`long long int`" in your code instead of just "int".

Task 1

In the code, you are given the class `MagicalPancakeGym()`. Design the attributes and write the function `addPancake(w)` to add a pancake with w grams into the pancake stack. Write the member functions `totalWeightAbovePancake(long long int w)` that return the total weight of all the pancakes above and including the pancake with w grams.

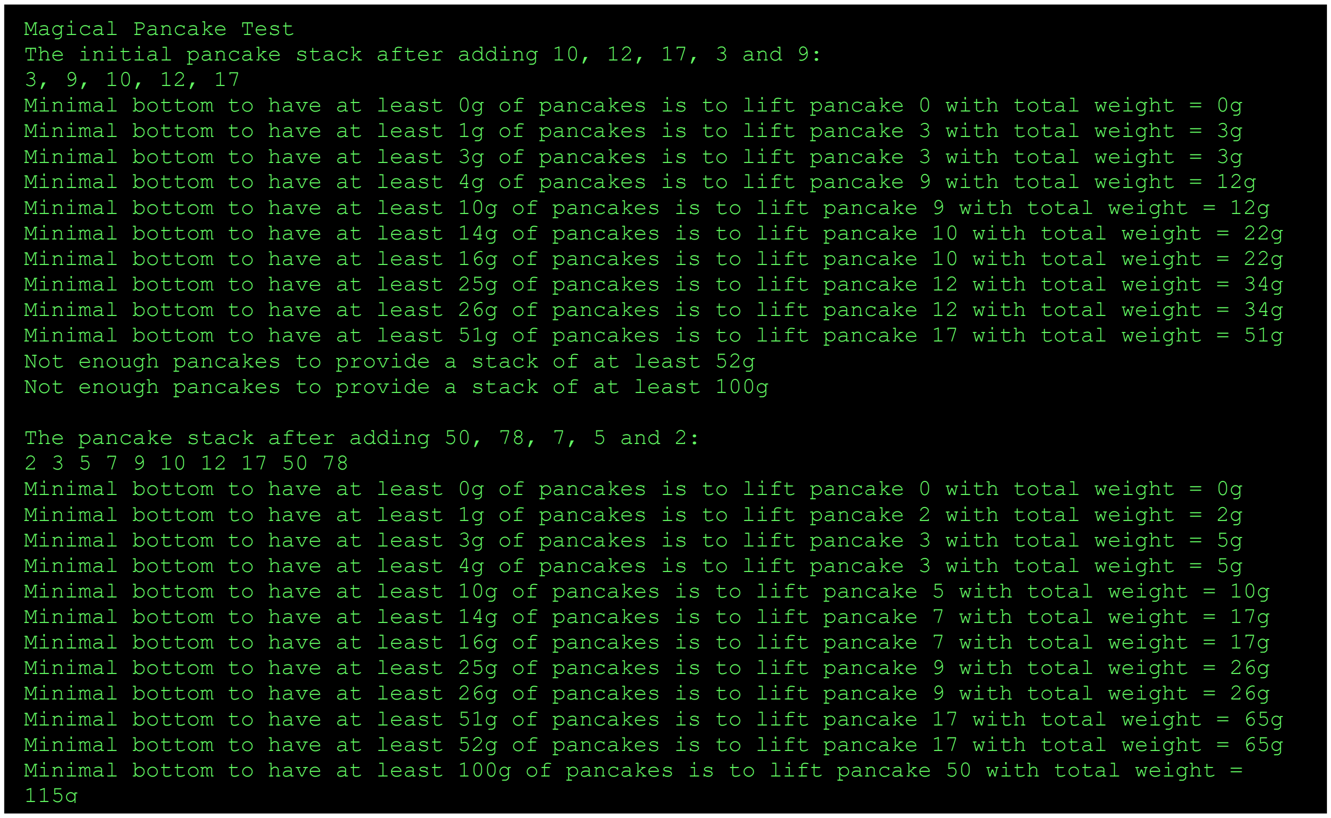
Task 2

Write the function `minimalBottomPancake2Lift(long long int w)` that returns the one of the pancake weight in the stack such that all the pancake above and including that pancake will have a weight at least w grams with the minimal number of pancakes.

(If there is any discrepancy between the function descriptions with the definitions in the skeleton file given, please refer to the skeleton code.)

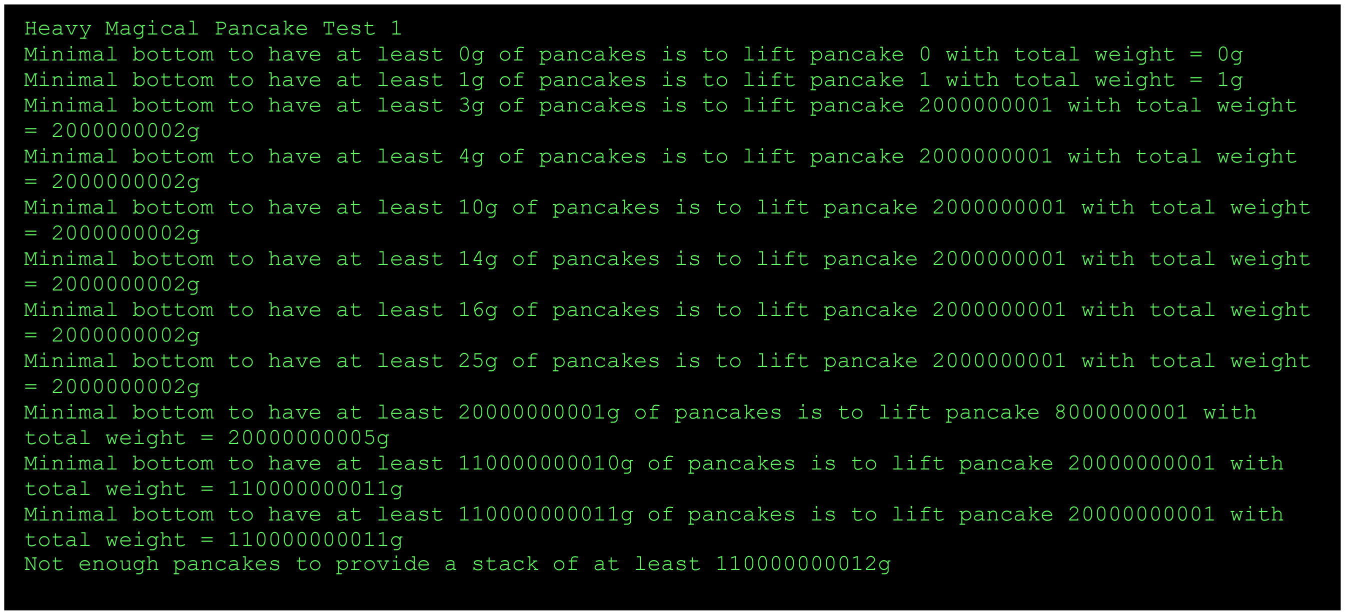
Submission

Simply copy and paste whatever code you needed into coursemology except the function `main()`.

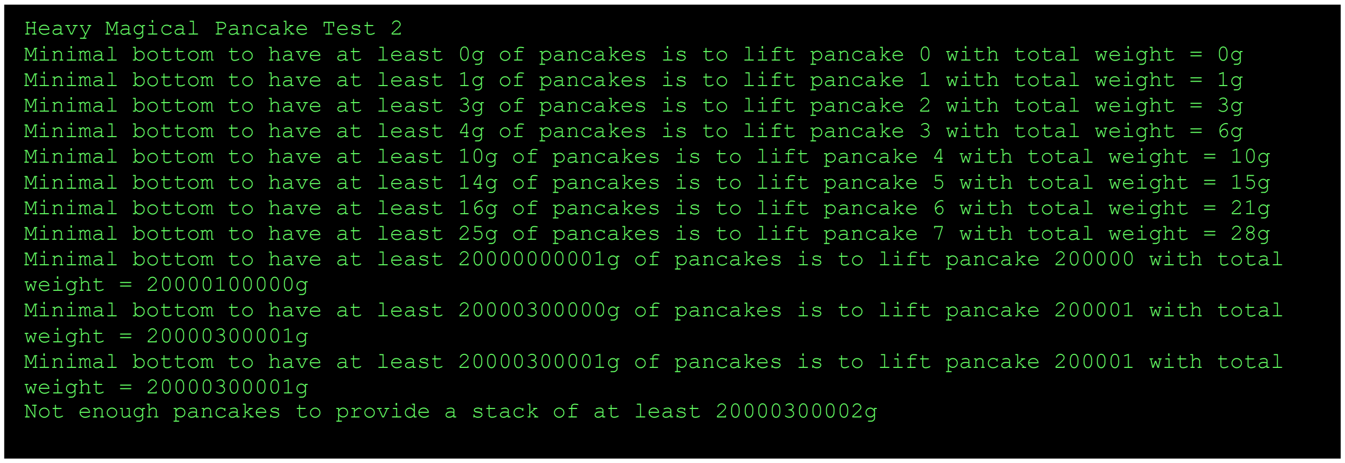


```
Magical Pancake Test
The initial pancake stack after adding 10, 12, 17, 3 and 9:
3, 9, 10, 12, 17
Minimal bottom to have at least 0g of pancakes is to lift pancake 0 with total weight = 0g
Minimal bottom to have at least 1g of pancakes is to lift pancake 3 with total weight = 3g
Minimal bottom to have at least 3g of pancakes is to lift pancake 3 with total weight = 3g
Minimal bottom to have at least 4g of pancakes is to lift pancake 9 with total weight = 12g
Minimal bottom to have at least 10g of pancakes is to lift pancake 9 with total weight = 12g
Minimal bottom to have at least 14g of pancakes is to lift pancake 10 with total weight = 22g
Minimal bottom to have at least 16g of pancakes is to lift pancake 10 with total weight = 22g
Minimal bottom to have at least 25g of pancakes is to lift pancake 12 with total weight = 34g
Minimal bottom to have at least 26g of pancakes is to lift pancake 12 with total weight = 34g
Minimal bottom to have at least 51g of pancakes is to lift pancake 17 with total weight = 51g
Not enough pancakes to provide a stack of at least 52g
Not enough pancakes to provide a stack of at least 100g

The pancake stack after adding 50, 78, 7, 5 and 2:
2 3 5 7 9 10 12 17 50 78
Minimal bottom to have at least 0g of pancakes is to lift pancake 0 with total weight = 0g
Minimal bottom to have at least 1g of pancakes is to lift pancake 2 with total weight = 2g
Minimal bottom to have at least 3g of pancakes is to lift pancake 3 with total weight = 5g
Minimal bottom to have at least 4g of pancakes is to lift pancake 3 with total weight = 5g
Minimal bottom to have at least 10g of pancakes is to lift pancake 5 with total weight = 10g
Minimal bottom to have at least 14g of pancakes is to lift pancake 7 with total weight = 17g
Minimal bottom to have at least 16g of pancakes is to lift pancake 7 with total weight = 17g
Minimal bottom to have at least 25g of pancakes is to lift pancake 9 with total weight = 26g
Minimal bottom to have at least 26g of pancakes is to lift pancake 9 with total weight = 26g
Minimal bottom to have at least 51g of pancakes is to lift pancake 17 with total weight = 65g
Minimal bottom to have at least 52g of pancakes is to lift pancake 17 with total weight = 65g
Minimal bottom to have at least 100g of pancakes is to lift pancake 50 with total weight =
115g
```



```
Heavy Magical Pancake Test 1
Minimal bottom to have at least 0g of pancakes is to lift pancake 0 with total weight = 0g
Minimal bottom to have at least 1g of pancakes is to lift pancake 1 with total weight = 1g
Minimal bottom to have at least 3g of pancakes is to lift pancake 2000000001 with total weight
= 2000000002g
Minimal bottom to have at least 4g of pancakes is to lift pancake 2000000001 with total weight
= 2000000002g
Minimal bottom to have at least 10g of pancakes is to lift pancake 2000000001 with total weight
= 2000000002g
Minimal bottom to have at least 14g of pancakes is to lift pancake 2000000001 with total weight
= 2000000002g
Minimal bottom to have at least 16g of pancakes is to lift pancake 2000000001 with total weight
= 2000000002g
Minimal bottom to have at least 25g of pancakes is to lift pancake 2000000001 with total weight
= 2000000002g
Minimal bottom to have at least 20000000001g of pancakes is to lift pancake 8000000001 with
total weight = 20000000005g
Minimal bottom to have at least 110000000010g of pancakes is to lift pancake 20000000001 with
total weight = 110000000011g
Minimal bottom to have at least 110000000011g of pancakes is to lift pancake 20000000001 with
total weight = 110000000011g
Not enough pancakes to provide a stack of at least 110000000012g
```



```
Heavy Magical Pancake Test 2
Minimal bottom to have at least 0g of pancakes is to lift pancake 0 with total weight = 0g
Minimal bottom to have at least 1g of pancakes is to lift pancake 1 with total weight = 1g
Minimal bottom to have at least 3g of pancakes is to lift pancake 2 with total weight = 3g
Minimal bottom to have at least 4g of pancakes is to lift pancake 3 with total weight = 6g
Minimal bottom to have at least 10g of pancakes is to lift pancake 4 with total weight = 10g
Minimal bottom to have at least 14g of pancakes is to lift pancake 5 with total weight = 15g
Minimal bottom to have at least 16g of pancakes is to lift pancake 6 with total weight = 21g
Minimal bottom to have at least 25g of pancakes is to lift pancake 7 with total weight = 28g
Minimal bottom to have at least 20000000001g of pancakes is to lift pancake 200000 with total
weight = 20000100000g
Minimal bottom to have at least 20000300000g of pancakes is to lift pancake 200001 with total
weight = 20000300001g
Minimal bottom to have at least 20000300001g of pancakes is to lift pancake 200001 with total
weight = 20000300001g
Not enough pancakes to provide a stack of at least 20000300002g
```

Now you know why we need `long long int`?