Programming Assignment #3

Lecturer: Prof. Seung-Hwan Baek

Teaching Assistants: Yujin Jeon, Suhyun Shin, Seokjun Choi, Eunsue Choi

***** PLEASE READ THIS GRAY BOX CAREFULLY BEFORE STARTING THE ASSIGNMENT *****

Due date: 11:59 PM Dec 04, 2023

Evaluation policy:

- Late submission penalty.
  - 11:59 PM Dec 04 ~ 11:59 PM Dec 05.
    - Late submission penalty (30%) will be applied to the total score.
  - After 11:59 PM Dec 05.
    - 100% penalty is applied for that submission.
- Your code will be automatically tested using an evaluation program.
  - Each problem has the maximum score.
  - A score will be assigned based on the behavior of the program.
- We won't accept any submission via email - it will be ignored.

***** PLEASE READ THIS GRAY BOX CAREFULLY BEFORE STARTING THE ASSIGNMENT *****

Coding:

- Please do not use the containers in C++ standard template library (STL).
  - Such as <queue>, <vector>, and <stack>.
  - Any submission using the above headers will be disregarded.

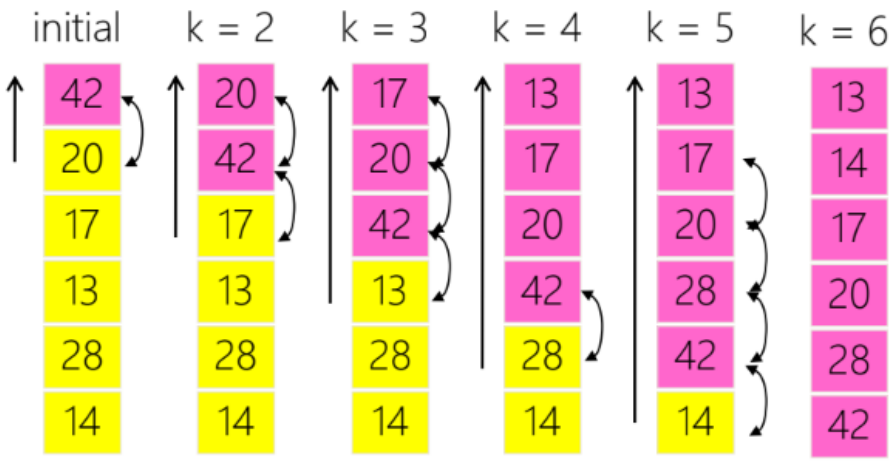
Submission:

- Before submitting your work, compile and test your code using C++11 compiler in repl.it.
  - Please refer to the attached file named “DataStructure_PA_instructions.pdf”.
  - There might be a penalty if the submission would not work in the “repl.it + C++11” environment.
- Files you need to submit. (Do not change the filename.)
  - pa3.cpp
  - sort.cpp and sort.h
  - tree.cpp and tree.h
  - bst.cpp and bst.h
  - avl.cpp and avl.h
  - open_hash_function.cpp and open_hash_function.h
  - open_hash_table.cpp and open_hash_table.h
  - closed_hash_function.cpp and closed_hash_function.h
  - closed_hash_table.cpp and closed_hash_table.h

Any questions?

- Please use PLMS - Q&A board.

1. Insertion Sort (3 pts)



a. Implement a function that sorts a given array using the **Insertion Sort** algorithm. (k: iteration, A[.]: array) On path k, A[k] is inserted at the correct position within an already sorted list [A[1], A[2], ..., A[k-1]]. You can modify `sort.cpp` and `sort.h` files for this problem.

b. Input & Output

Input: A sequence of commands

- `('insertion',integer)`: insert `integer` into the array (there will be no duplicated integers.)
- `('insertionSort',NULL)`: sort the array using the Insertion Sort algorithm

Output:

- Every value in the array **whenever insertion happens** including the initial step, string separated with the white space (please use built-in function to print the array).
- We won't test array size over 20 or array size of 0.

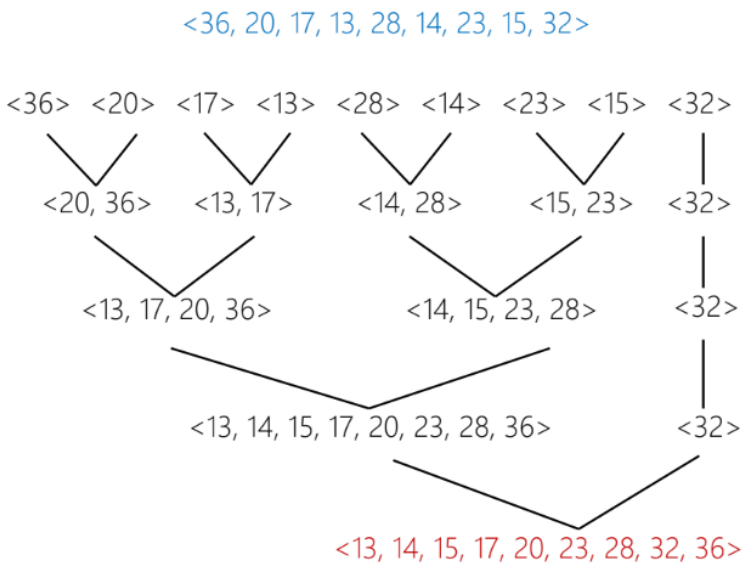
c. Example Input & Output

Input	Output
[('insertion',17), ('insertion',20), ('insertion',2), ('insertion',21), ('insertion',4), ('insertionSort',NULL)]	17 20 2 21 4 2 17 20 21 4 2 4 17 20 21
[('insertion',42), ('insertion',20), ('insertion',17), ('insertion',13), ('insertion',28), ('insertion',14), ('insertionSort',NULL)]	42 20 17 13 28 14 20 42 17 13 28 14 17 20 42 13 28 14 13 17 20 42 28 14 13 17 20 28 42 14 13 14 17 20 28 42
[('insertion',5), ('insertion',6), ('insertion',4), ('insertion',3), ('insertion',2), ('insertion',1), ('insertionSort',NULL)]	5 6 4 3 2 1 4 5 6 3 2 1 3 4 5 6 2 1 2 3 4 5 6 1 1 2 3 4 5 6

d. Example execution

```
>> ./pa3.exe 1 "[('insertion',17), ('insertion',20),
('insertion',2), ('insertion',21), ('insertion',4),
('insertionSort',NULL)]"
[Task 1]
17 20 2 21 4
2 17 20 21 4
2 4 17 20 21
```

2. Non-recursive Merge Sort (3 pts)



a. Implement a function that sorts a given array using the **Merge Sort** algorithm in ascending order using **non-recursive** merge sort. Start with the sorted segments of size 1 and do pairwise merging of these sorted segments as in the upward pass. You can modify `sort.cpp` and `sort.h` files for this problem.

b. Input & Output

Input: A sequence of commands

- `('insertion',integer)`: insert `integer` into the array.
- `('mergeSort',NULL)`: sort the array using the Merge Sort algorithm.

Output:

- Every value in the array for each iteration step including the initial step, string separated with the white space (please use built-in function to print the array).
- You don't need to consider exceptional cases such as overflow or an empty array. We will not test such cases.

c. Example Input & Output

Input	Output
[('insertion',56), ('insertion',42), ('insertion',20), ('insertion',17), ('insertion',13), ('insertion',28), ('insertion',14), ('mergeSort',NULL)]	56 42 20 17 13 28 14 42 56 17 20 13 28 14 17 20 42 56 13 14 28 13 14 17 20 28 42 56
[('insertion',6), ('insertion',5), ('insertion',4), ('insertion',3), ('insertion',2), ('insertion',1), ('mergeSort',NULL)]	6 5 4 3 2 1 5 6 3 4 1 2 3 4 5 6 1 2 1 2 3 4 5 6
[('insertion',36), ('insertion',20), ('insertion',17), ('insertion',13), ('insertion',28), ('insertion',14), ('insertion',23), ('insertion',15), ('insertion',32), ('mergeSort',NULL)]	36 20 17 13 28 14 23 15 32 20 36 13 17 14 28 15 23 32 13 17 20 36 14 15 23 28 32 13 14 15 17 20 23 28 36 32 13 14 15 17 20 23 28 32 36

d. Example execution

```
>> ./pa3.exe 2 "[('insertion',56), ('insertion',42),
('insertion',20), ('insertion',17), ('insertion',13),
('insertion',28), ('insertion',14), ('mergeSort',NULL)]"
[Task 2]
56 42 20 17 13 28 14
42 56 17 20 13 28 14
17 20 42 56 13 14 28
13 14 17 20 28 42 56
```

3. BST Insertion / Deletion / Find Nth Minimum (4 pts)

a. Implement functions that **inserts** and **deletes** an element into a binary search tree (BST). Also you should find the N^{th} minimum value. You can modify `bst.cpp` and `bst.h` files for this problem.

b. Input & output of `BinarySearchTree::insertion`
Input: Key of the element to be inserted. The key has a positive integer value.
Output: Return the -1 if the key already exists in the tree, `0` otherwise.
(If the key already exists, do not insert the element)

c. Input & output of `BinarySearchTree::deletion`
Input: Key of the element to be deleted.
Output: Return -1 if the key does not exist in the tree, `0` otherwise. If the key does not exist, do not delete any element
Note that replace the smallest key in right subtree when delete the node with degree 2

d. Input & output of `BinarySearchTree::findNthMinimum`
Input: Key of the N^{th} minimum value. The key has a positive integer value.
Output: Return the -1 if there is no such N^{th} minimum value, N^{th} minimum value otherwise

e. `task_3` prints
   i. the return for each insertion/deletion and
   ii. the return for N^{th} minimum value
   iii. the results of preorder and inorder traversal of the constructed tree.
   (If empty tree, don't return preorder or inorder traveral results)

Example Input & Output

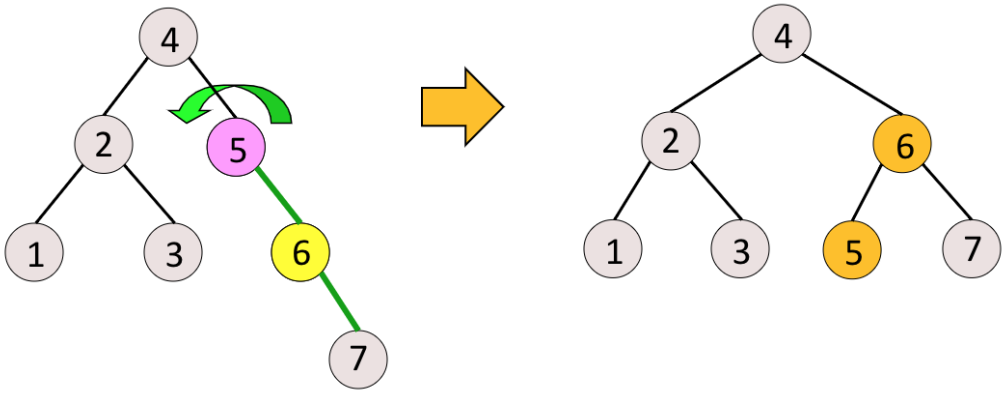
Input	Output
[('insertion',4), ('insertion',6), ('insertion',6), ('insertion',7), ('deletion',7)]	0 0 -1 0 0 4 6 4 6
[('insertion',4), ('insertion',2), ('findNthMinimum', 1), ('insertion',10), ('insertion',9), ('insertion',15), ('insertion',1), ('deletion',1), ('deletion',4), ('deletion',10), ('findNthMinimum', 5)]	0 0 2 0 0 0 0 0 0 0 -1 9 2 15 2 9 15
[('deletion', 3),('insertion', 10),('deletion', 10),('findNthMinimum',2)]	-1 0 0 -1

f. Example execution

```
>> ./pa3.exe 3 "[('insertion',4), ('insertion',6),
('insertion',6), ('insertion',7),
('deletion',7),('findNthMinimum', 2)]"
[Task 3]
0
0
-1
0
0
6
4 6
4 6
```

4. AVL Tree Insertion / Deletion (10 pts)

- Insert 7
  - Violation at node 2
  - Left (RR) rotation



Example of left rotation to resolve RR imbalance

AVLTree class is a subclass of BinarySearchTree class implemented in task3. If you use the insert and erase functions of BinarySearchTree, you can implement it more simply.

a. Implement a function that inserts and deletes an element into an AVL tree. The insertion and deletion might cause the AVL tree to violate its properties (***Imbalance***). Your code should be able to resolve the imbalances (LL, LR, RL, RR) of the AVL tree. You can modify `avl.cpp` and `avl.h` files for this problem. Also, You can add public member at Node class implemented in `tree.h` if needed.

b. Input & Output of AVL`Tree::insertion` (insert function for AVL tree)
Input: key of element to be inserted. (keys are given only positive value)
Output:
   - **Corresponding imbalance type (RR, LR, RL, LL)** , if the insertion is successful and the imbalance occurred.
   - **Fail**, if the key already exists in the tree.
   - **OK**, if the insertion is successful and the imbalance did not occur

c. Input & Output of `AVLTree::deletion` (delete function for AVL tree)
Input: key of element to be deleted. (keys are given only positive value)
Output:
- **OK,** if the deletion is successful.
- **Fail**, if the key does not exist in the tree.
- *Note that replace the smallest key in right subtree when delete the node with degree 2 in BST deletion stage.*

d. `task_4` prints
i. The return value for each insertion and deletion
ii. The results of preorder and inorder traversal of the constructed tree.

e. Example Input & Output

Input	Output

`[('insertion',4), ('insertion',6), ('insertion',0), ('deletion',7)]`	`OK`
	`OK`
	`OK`
	`Fail`
	`4 0 6`
	`0 4 6`

`[('insertion',4), ('insertion',2), ('insertion',10), ('insertion',9), ('insertion',15), ('insertion',5), ('insertion',0), ('deletion',4), ('insertion',10)]`	`OK`
	`OK`
	`OK`
	`OK`
	`OK`
	`RL`
	`OK`
	`OK`
	`Fail`
	`9 2 0 5 10 15`
	`0 2 5 9 10 15`

f. Example execution

```
>> ./pa3.exe 4 "[('insertion',4), ('insertion',2),
('insertion',10), ('insertion',9), ('insertion',15),
('insertion',5), ('insertion',0), ('deletion',4),
('insertion',10)]"

[Task 4]
OK
OK
OK
OK
OK
RL
OK
OK
Fail
9 2 0 5 10 15
0 2 5 9 10 15
```

5. Open hash table (2 pts)

a. Implement an **open hash table** with **division-method**. This hash table is used with integer keys and hashing into a table of size M.
Every component of the key is calculates as modulo as described in our Lecture Note.
This hash table uses a singly **linked list** as a collision handling method. You don't need to consider deletion, the maximum length of the linked list and multiple insertions of the same key.
You can modify `open_hash_function.cpp`, `open_hash_table.cpp`, `open_hash_function.h` and `open_hash_table.h` files for this problem.

b. Input & output
Input: A sequence of commands
- `('M',integer)`: the size of a hash table.
(The first command is always `'M'`)
- `('insertion',int)`: insert `integer` into the hash table.

Output: For each slot of the hash table, print out
- the linked list, if the state of the slot is occupied.
- the state, if the state of the slot is empty.

c. Example Input & Output

Input	Output

`[('M',4), ('insertion',32615), ('insertion',315), ('insertion',6468), ('insertion',94833)]`	`0: 6468`
	`1: 94833`
	`2: empty`
	`3: 32615, 315`

`[('M',7), ('insertion',32615), ('insertion',315), ('insertion',6468), ('insertion',94833), ('insertion',22)]`	`0: 315, 6468`
	`1: 22`
	`2: 32615`
	`3: empty`
	`4: 94833`
	`5: empty`
	`6: empty`

d. Example execution

```
>> ./pa3.exe 5 "[('M',4), ('insertion',32615),
('insertion',315), ('insertion',6468), ('insertion',94833)]"
[Task 5]
0: 6468
1: 94833
2: empty
3: 32615, 315
```

6. Closed hash table (5 pts)

a. Implement insertion of **a closed hash table** with **rehashing** implementation. This hash table is used with integer keys and hashing into a table of size M.
This hash table uses **double hashing** as a collision handling method. The index of the key k after i-th collision, $h_i(k)$, is:

$$h_i(k) = (h_1(k) + ih_2(k)) \; mod \; M$$

Where $h_1(k)$ is the **digital-folding method**. Every component of the key is folded with a size of **1(digit)** and calculates their sum as described in our Lecture Note. $h_2(k)$ is:

$$h_2(k) = 1 + (k \; mod \; (M-1))$$

Hash table consists of 0 ~ (M-1) indices and you need to implement additional list named 'F' to contain integers that could not get into the hash table. You don't need to consider deletion or multiple insertions of the same key.
You can modify `closed_hash_function.cpp`, `closed_hash_table.cpp`, `closed_hash_function.h` and `closed_hash_table.h` files for this problem.

b. Input & Output

Input: A sequence of commands

`('M',integer)`: the size of a hash table.
(The first command is always `'M'`)

- `('insertion',integer)`: insert `integer` into the hash table. If the insertion fails, insert the integer into `'F'` list.

Output: For each slot of the hash table, print out

- the value, if the state of the slot is occupied.
- the state if the state of the slot is empty.
- `'F: '` and its elements. If 'F' is empty, print empty.

c. Example Input & Output

Input	Output

`[('M',4), ('insertion',15), ('insertion',2), ('insertion',3)]`	`0: empty`
	`1: 2`
	`2: 15`

	3: 3 F: empty
[('M',4), ('insertion',15), ('insertion',46), ('insertion',81), ('insertion',21), ('insertion',35), ('insertion',7)	0: 46 1: 81 2: 15 3: 21 F: 35, 7

d. Example execution

```
>> ./pa3.exe 6 "[('M',4), ('insertion',15), ('insertion',2),
('insertion',3)]"
[Task 6]
0: empty
1: 2
2: 15
3: 3
F: empty
```