Programming Assignment #3

Lecturer: Prof. Seung-Hwan Baek
Teaching Assistants: Chunghyun Park, Kanghee Lee, Seungjoo Shin, Dongmin You

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

Due date: 11:59 PM Dec 05, 2022

Evaluation policy:

- Late submission penalty.
  - 11:59 PM Dec 05 ~ 11:59 PM Dec 06.
    - Late submission penalty (30%) will be applied to the total score.
  - After 11:59 PM Dec 06.
    - 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.

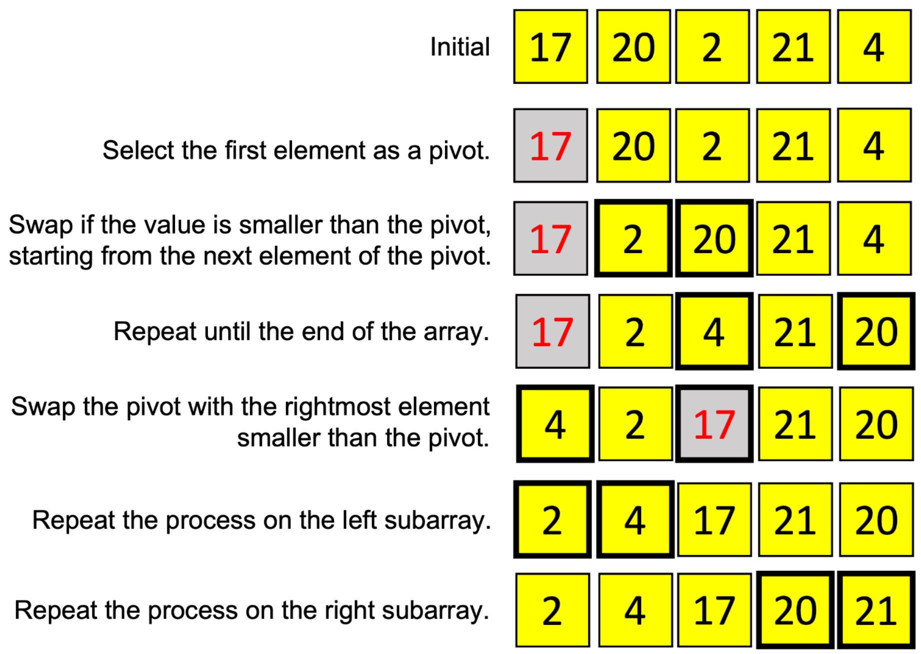
Submission:

- 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. Quick Sort (3 pts)


Initial
17 20 2 21 4
Select the first element as a pivot.
17 20 2 21 4
Swap if the value is smaller than the pivot, starting from the next element of the pivot.
17 2 20 21 4
Repeat until the end of the array.
17 2 4 21 20
Swap the pivot with the rightmost element smaller than the pivot.
4 2 17 21 20
Repeat the process on the left subarray.
2 4 17 21 20
Repeat the process on the right subarray.
2 4 17 20 21

a. Implement a function that sorts a given array using the **Quick Sort** algorithm with **in-place partitioning** in ascending order. Use **the first element** at the pivot as shown in **the above figure**. 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.)
   - `(‘quickSort’,NULL)`: sort the array using the Quick Sort algorithm

   Output:
   - Every value in the array **whenever swapping 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

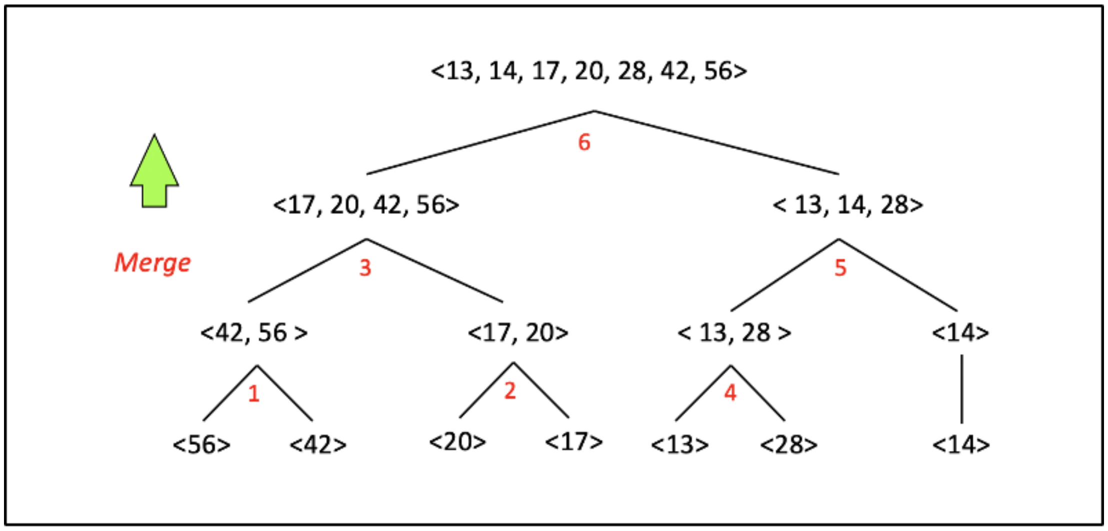
`[('insertion',17), ('insertion',20), ('insertion',2), ('insertion',21), ('insertion',4), ('quickSort',NULL)]`	17 20 2 21 4
	17 2 20 21 4
	17 2 4 21 20
	4 2 17 21 20
	2 4 17 21 20
	2 4 17 20 21

`[('insertion',5), ('insertion',6), ('insertion',4), ('insertion',3), ('insertion',2), ('insertion',1), ('quickSort',NULL)]`	5 6 4 3 2 1
	5 4 6 3 2 1
	5 4 3 6 2 1
	5 4 3 2 6 1
	5 4 3 2 1 6
	1 4 3 2 5 6
	1 2 3 4 5 6

d. Example execution

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

2. Recursive Merge Sort (3 pts)



a. Implement a function that sorts a given array using the **Merge Sort** algorithm in ascending order using **recursive** merge sort. Split a list of elements into two sublists with the first sublist bigger than the second sublist for the case when the input array has an odd number of elements. 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 sorting 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 20 17 13 28 14`
	`42 56 17 20 13 28 14`
	`17 20 42 56 13 28 14`
	`17 20 42 56 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 4 3 2 1`
	`4 5 6 3 2 1`
	`4 5 6 2 3 1`
	`4 5 6 1 2 3`
	`1 2 3 4 5 6`

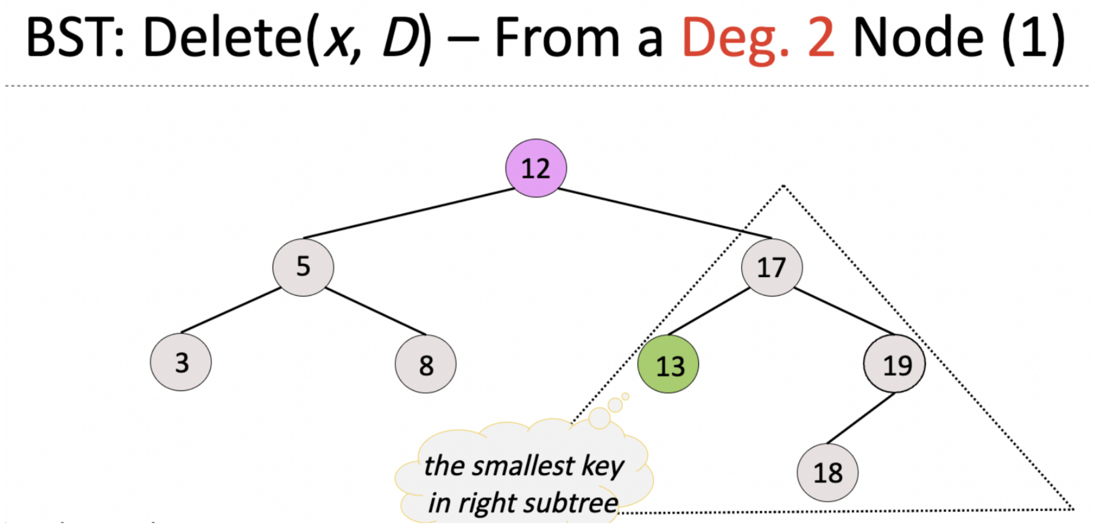
d. Example execution

```
>> ./pa3.exe 2 "[('insertion',56), ('insertion',42),
('insertion',20), ('insertion',17), ('insertion',13),
('insertion',28), ('insertion',14), ('mergeSort',NULL)]"
[Task 4]
56 42 20 17 13 28 14
42 56 20 17 13 28 14
42 56 17 20 13 28 14
17 20 42 56 13 28 14
17 20 42 56 13 28 14
17 20 42 56 13 14 28
13 14 17 20 28 42 56
```

3. BST Insertion / Deletion (4 pts)

a. Implement functions that **inserts** and **deletes** an element into a binary search tree (BST). 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)


BST: Delete(x, D) – From a Deg. 2 Node (1)
12
5
17
3
8
13
19
18
the smallest key in right subtree

Example of deleting the node with degree 2 in Binary Search Tree

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. `task_3` prints
   i. the return for each insertion/deletion and
   ii. the results of preorder and inorder traversal of the constructed tree.

e. Example Input & Output

Input	Output

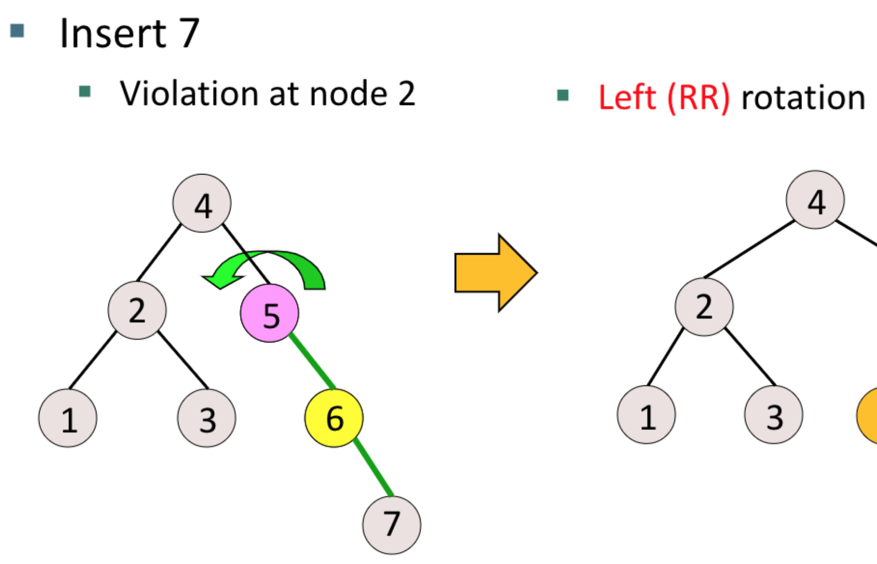
`[('insertion',4), ('insertion',6), ('insertion',6), ('insertion',7), ('deletion',7)]`	0
	0
	-1
	0
	0
	4 6
	4 6

`[('insertion',4), ('insertion',2), ('insertion',10), ('insertion',9), ('insertion',15), ('insertion',1), ('deletion',1), ('deletion',4), ('deletion',10)]`	0
	0
	0
	0
	0
	0
	0
	0
	0
	9 2 15
	2 9 15

f. Example execution

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

4. AVL Tree Insertion / Deletion (10 pts)



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 a 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:
   - 0, if the insertion is successful.
   - -1, if the key already exists in the tree.

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:
   - 0, if the deletion is successful.
   - -1, 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)]`	`0`
	`0`
	`0`
	`-1`
	`4 0 6`
	`0 4 6`

`[('insertion',4), ('insertion',2), ('insertion',10), ('insertion',9), ('insertion',15), ('insertion',5), ('insertion',0), ('deletion',4), ('insertion',10)]`	`0`
	`0`
	`0`
	`0`
	`0`
	`0`
	`0`
	`0`
	`-1`
	`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]
0
0
0
0
0
0
0
0
-1
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 **digit-folding-method**. This hash table is used with integer keys and hashing into a table of size M.
Every component of the key is folded with a size of **1(digit)** and calculates their sum 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 the maximum length of the linked list, deletion of a key which does not exist or 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'`)
- `('insert',int)`: insert `integer` into the hash table.
- `('delete',int)`: delete `integer` from 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), ('insert',32615), ('insert',315), ('insert',6468), ('insert',94833)]`	`0: 6468`
	`1: 32615, 315`
	`2: empty`
	`3: 94833`

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

`[('M',4), ('insert',32615), ('insert',315), ('insert',6468), ('insert',94833),`	`0: 22`
	`1: 315`

('delete',32615), ('delete',6468), ('insert',22)]	2: empty 3: 94833

d. Example execution

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

6. Closed hash table (5 pts)

a. Implement **a closed hash table** with **rehashing** implementation. This hash table is used with n-bit integer keys and hashing into a table of size 2^r.
This hash table uses **quadratic probing** as a collision handling method. The index of the key k after i-th collision, $h_i(k)$, is:

$$h_i(k) = (h(k) + p(i)) \, mod \, 2^r$$

Where $h(k)$ is the **binary mid-square** hash function. This function maps an n-bit integer key to an index of a 2^r-sized table. As a key is n bits, your code should treat the square of the key as 2n bits. You can assume that r is even. $p(i)$, is:

$$p(i) = i^2 + i + 1$$

You don't need to consider an insertion when the table is full or a deletion of a key which does not exist or multiple insertions of the same key. And also you don't need to consider the case when the hash function cannot find an available slot. Assume you cannot insert a new key into the deleted slot.
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
- `('n',integer)`: the size of a key.
  (The first command is always `'n'`)
- `('r',integer)`: the size of an index.
  (The second command is always `'r'`)
- `('insert',integer)`: insert `integer` into the hash table.
- `('delete',integer)`: delete `integer` from the hash table.

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 or deleted.

c. Example Input & Output

Input	Output
`[('n',4), ('r',2), ('insert',15), ('insert',2), ('insert',3)]`	0: 15 1: 3 2: empty 3: 2
`[('n',4), ('r',2), ('insert',15), ('insert',2), ('insert',3), ('delete',2), ('delete',3)]`	0: 15 1: deleted 2: empty 3: deleted
`[('n',4), ('r',2), ('insert',15), ('insert',2), ('insert',3), ('delete',2), ('delete',3), ('insert',4)]`	0: 15 1: deleted 2: 4 3: deleted

d. Example execution

```
>> ./pa3.exe 6 "[('n',4), ('r',2), ('insert',15), ('insert',2),
('insert',3)]"
[Task 6]
0: 15
1: 3
2: empty
3: 2
```