Programming Assignment #3

Lecturer: Prof. Jaesik Park
Teaching Assistants: Hyeonjun Lee, Hyunsoo Ahn, Sangwoo Ryu

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

Due date: 11:59 PM May 11, 2022

Evaluation policy:

- Late submission penalty.
  - 11:59 PM May 11 ~ 11:59 PM May 12.
    - Late submission penalty (30%) will be applied to the total score.
  - After 11:59 PM May 12.
    - 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.
  - Due to the many requests, <cstring> and <string> are fine to use.

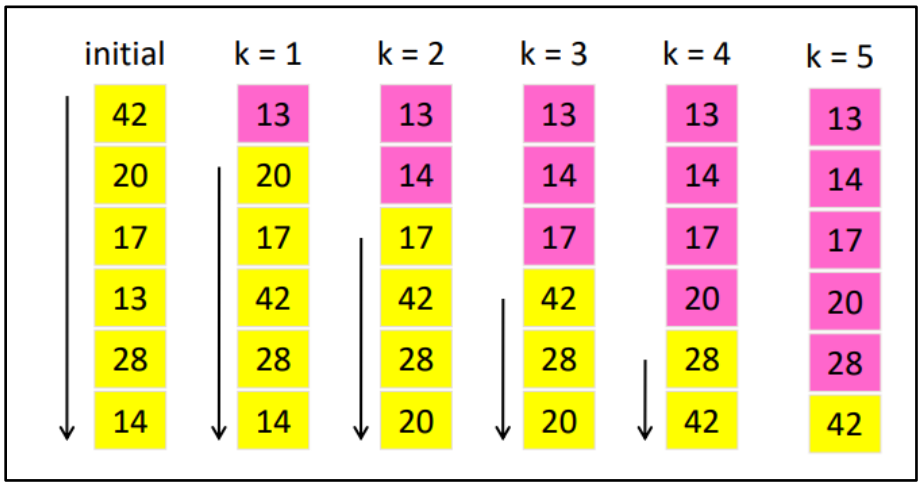
Submission:

- Before submitting your work, compile and test your code using C++11 compiler in repl.it.
  - Please refer to the attached file named "PA_instructions_updated.pdf".
  - There might be a penalty if the submission would not work in the "repl.it + C++11" environment.
- What you need to submit.
  - A zip file named "pa3.zip" that contains
    - pa3.cpp
    - sort.cpp and sort.h
    - bst.cpp and bst.h
    - avl.cpp and avl.h
    - hash_function.cpp and hash_function.h
    - shift_register.cpp and shift_register.h
    - hash_table.cpp and hash_table.h
    - tree.cpp and tree.h

Any questions?

- Please use PLMS - Q&A board.

1. Selection Sort (3 pts)



a. Implement a function that sorts a given array using the **selection sort** algorithm in ascending order. 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
- `('selectionSort',NULL)`: sort the array using the selection 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).
- We won't test array size over 20 or array size of 0.

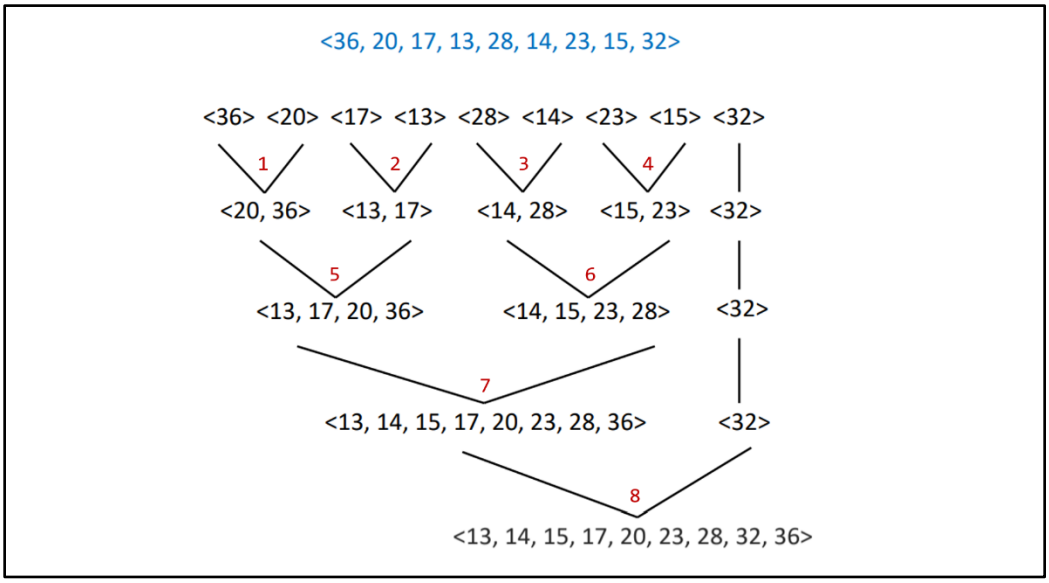
c. Example Input & Output

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

d. Example execution

```
>> ./pa3.exe 1 "[('insertion',42), ('insertion',20),
('insertion',17), ('insertion',13), ('insertion',28),
('insertion',14), ('selectionSort',NULL)]"
[Task 1]
42 20 17 13 28 14
13 20 17 42 28 14
13 14 17 42 28 20
13 14 17 42 28 20
13 14 17 20 28 42
13 14 17 20 28 42
```

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. 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 non-recursive merge sort algorithm.

Output:

- Every value in the array for each sorting step (after the pairwise merging) 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',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 17 13 28 14 23 15 32 20 36 13 17 28 14 23 15 32 20 36 13 17 14 28 23 15 32 20 36 13 17 14 28 15 23 32 13 17 20 36 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
[('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 5 6 3 4 2 1 5 6 3 4 1 2 3 4 5 6 1 2 1 2 3 4 5 6

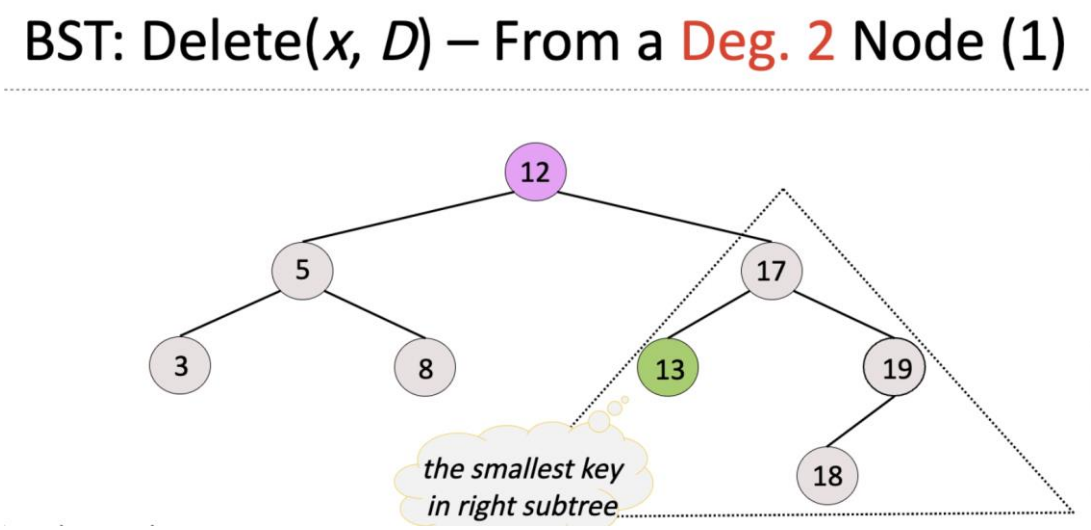
d. Example execution

```
>> ./pa3.exe 2 "[('insertion',36), ('insertion',20),
('insertion',17), ('insertion',13), ('insertion',28),
('insertion',14), ('insertion',23), ('insertion',15),
('insertion',32), ('mergeSort',NULL)]"
[Task 2]
36 20 17 13 28 14 23 15 32
20 36 17 13 28 14 23 15 32
20 36 13 17 28 14 23 15 32
20 36 13 17 14 28 23 15 32
20 36 13 17 14 28 15 23 32
13 17 20 36 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
```

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)



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 the -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

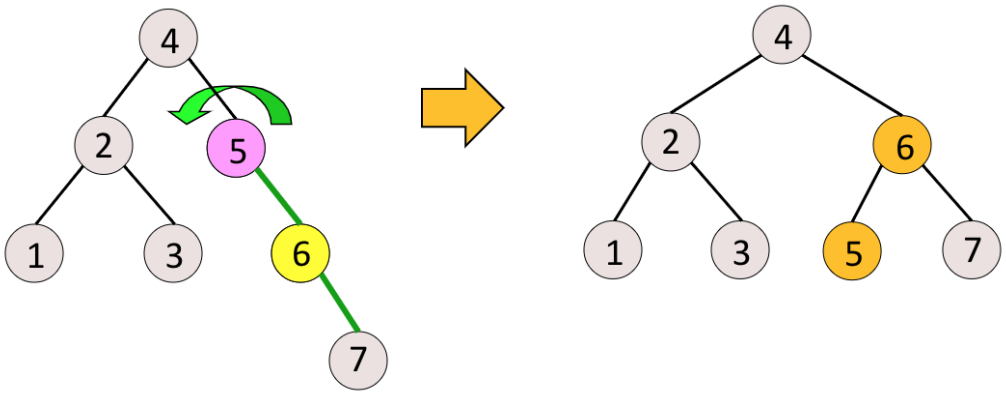
Input	Output
[('insertion',4), ('insertion',6), ('insertion',6), ('insertion',7), ('deletion',7)]	0 0 -1 0 0 4 6 4 6
[('insertion',4), ('insertion',2), (‘deletion’,2), ('deletion',2), ('deletion',4)]	0 0 0 -1 0
[('insertion',4), ('insertion',2), ('insertion',10), ('insertion',9), ('insertion',15), ('insertion',1),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

- 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 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 `AVLTree::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`
	`-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. Mid-square hashing (2 pts)

a. Implement a 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. You can modify `hash_function.cpp` and `hash_function.h` files for this problem.

b. Input & output

Input: Three commands (The order is always `'n'`, `'r'`, and `'key'`)

- `('n', integer)`: the size of a key.
- `('r', integer)`: the size of an index.
- `('key', integer)`: a key to be hashed (in decimal).

Output: The result (i.e., index) of hashing in decimal.

c. Example Input & Output

Input	Output
`[('n', 4), ('r', 4), ('key', 10)]`	`9`
`[('n', 10), ('r', 4), ('key', 1023)]`	`8`
`[('n', 10), ('r', 4), ('key', 15)]`	`0`

d. Example execution

```
>> ./pa3.exe 5 "[('n', 4), ('r', 4), ('key', 10)]"
[Task 5]
9
```

6. Shift-Register Sequence (3 pts)

a. Implement a function to generate a random permutation using a **shift-register sequence**. Given M (a power of 2), a constant k ($1 \le k \le M - 1$), and an initial sequence number d_1, then generates sequence d_2, d_3, d_4, ... We will use a below algorithm which is same as described in our Lecture note.

Algorithm

Start with some number d_1 such that $1 \le d_1 \le M - 1$
Repeat to generate successive numbers d_2, d_3, d_4, ...
- Double the previous number (Left shift)
- If the result ≥ M, then
  - Subtract M and
  - Take the "bitwise modulo-2 sum (bitwise XOR)" of
  - the result &
  - the selected constant k

You can modify `shift_register.cpp` and `shift_register.h` files for this problem.

b. Input & Output

Input: Four commands (The order is always `'m'`, `'k'`, `'d'`, and `'i'`)

- `('m', integer)`: the integer which is a power of 2
- `('k', integer)`: the integer between 1 and M - 1
- `('d', integer)`: the initial sequence number d_1 in decimal
- `('i', integer)`: the target index of the sequence (starts from 1)

Output: `i`-th number of the generated sequence in decimal (d_i)

c. Example Input & Output

Input	Output
`[('m', 4), ('k', 3), ('d', 1), ('i', 3)]`	3
`[('m', 8), ('k', 3), ('d', 5), ('i', 1)]`	5
`[('m', 16), ('k', 3), ('d', 2), ('i', 4)]`	3

d. Example execution

```
>> ./pa3.exe 6 "[('m', 8), ('k', 3), ('d', 2), ('i', 5)]"
[Task 6]
7
```

7. 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 **pseudo-random probing** as a collision handling method. The index of the key k after i-th collision, $h_i(k)$, is:

$$h_i(k) = (h(k) + d_i)\ mod\ 2^r$$

when h is the hash function implemented in `task_5`, and d_i is the random permutation of integers 1, 2, …, 2^r-1 generated using the function implemented in `task_6`.

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 new key into deleted slot.

You can modify `hash_table.cpp` and `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'`)
- `('k', integer)`: the constant for shift-register sequence.
  (The third command is always `'k'`)
- `('d', integer)`: the initial sequence number of random permutation.
  (The fourth command is always `'d'`)
- `('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), ('k', 3), ('d', 1), ('insert', 15), ('insert', 2), ('insert', 3)]	0: 15 1: 2 2: 3 3: empty
[('n', 4), ('r', 2), ('k', 3), ('d', 1), ('insert', 15), ('insert', 2), ('insert', 3), ('delete', 2), ('delete', 3)]	0: 15 1: deleted 2: deleted 3: empty
[('n', 4), ('r', 2), ('k', 3), ('d', 1), ('insert', 15), ('insert', 2), ('insert', 3), ('delete', 2), ('delete', 3), ('insert', 0)]	0: 15 1: deleted 2: deleted 3: 0

d. Example execution

```
>> ./pa3.exe 7 "[('n', 4), ('r', 2), ('k', 3), ('d', 1),
('insert', 15), ('insert', 2), ('insert', 3)]"
[Task 7]
0: 15
1: 2
2: 3
3: empty
```