Programming Assignment #3

Lecturer: Prof. Jaesik Park
Teaching Assistants: ByeongHoon So, Hyunmin Lee, Junha Lee

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

Due date: 11:59PM May 16, 2020

Evaluation policy:

- Late submission penalty.
  - 11:59PM May 16 ~ 11:59PM May 17.
    - Late submission penalty (30%) will be applied to the total score.
  - After 11:59PM May 17.
    - 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 submit 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 "repl.it + C++11" environment.
- What you need to submit.
  - a zip file named "pa3.zip" that contains
    - pa3.cpp
    - bst.cpp and bst.h
    - sort.cpp and sort.h

Any questions?

- Please use LMS - Q&A board.

1. Quiz (2 pts)

From given statements about the sorting algorithm, choose **every wrong statement.**

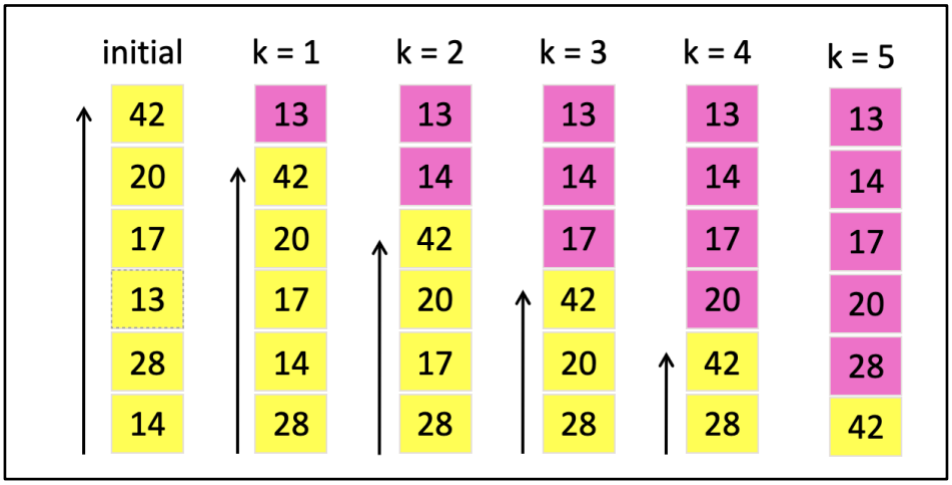
- Statements
  (1) If the relative order of elements with the same keys are retained after the sorting, it is a stable sort.
  (2) Bubble sort takes a more significant number of record swap than Selection sort.
  (3) The time complexity of Selection sort is O(n) in the best case.
  (4) Merge sort can be implemented in a non-recursive manner.
  (5) Merge sort is always faster than Insertion sort.
  (6) Bucket sort is a kind of non-comparison sort.
  (7) For Quick sort, the choice of a pivot doesn't affect the performance.

Print out your answer. If you think there are multiple answers, print out a sequence of answers in **ascending order** with the string separated with the spacebar. If you believe every statement is correct, print out an empty string. You can modify `task_1` function in `pa3.cpp`.

- Example execution

```
>> ./pa3.exe 1
[Task 1]
1 2 3
```

2. Bubble Sort (2 pts)



a. Implement a function that sorts a given array using the **bubble 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

- `('insert',integer)`: insert `integer` into the array
- `('bubbleSort',NULL)`: sort the array using the bubble 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.

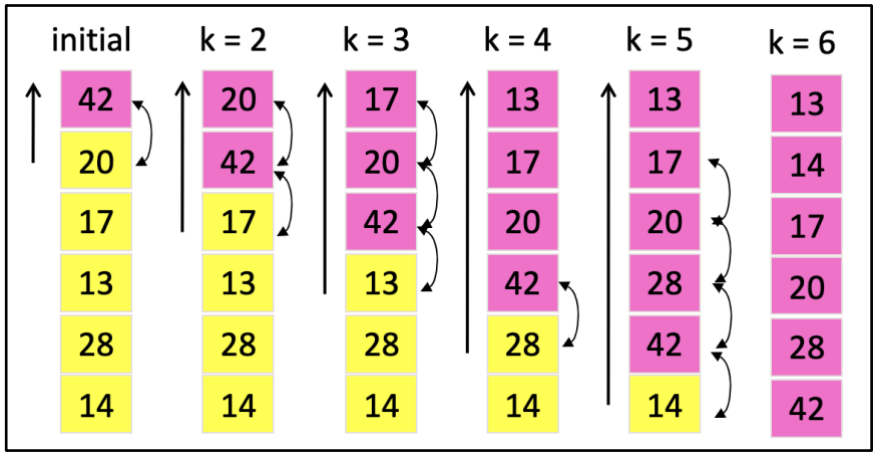
c. Example Input & Output

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

d. Example execution

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

3. Insertion Sort (2 pts)



a. Implement a function that sorts a given array using the **insertion 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

- `('insert',integer)`: insert `integer` into the array
- `('insertionSort',NULL)`: sort the array using the insertion sort algorithm

Output:

- Every value in the array for each sorting step including the initial step, with 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.

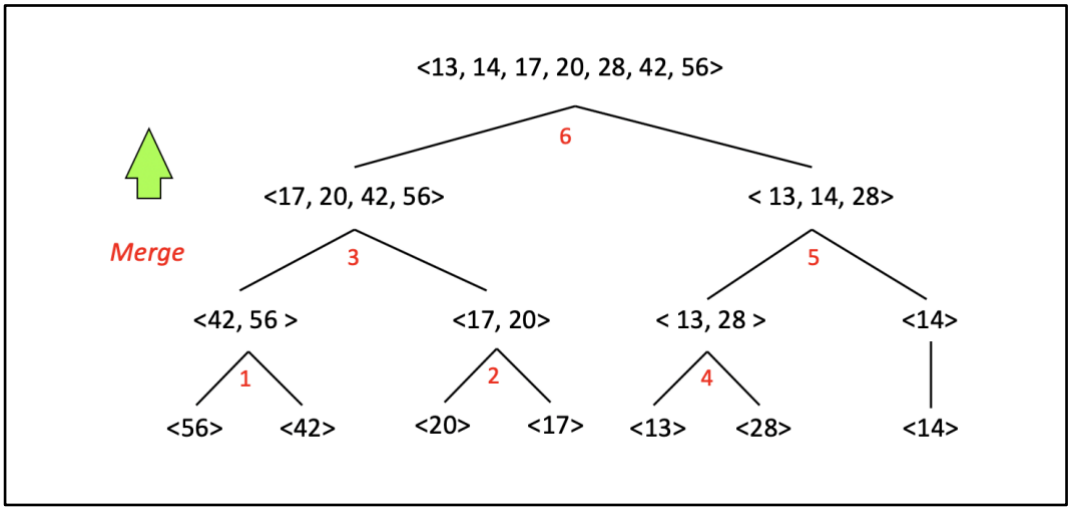
c. Example Input & Output

Input	Output
[('insert',42), ('insert',20), ('insert',17), ('insert',13), ('insert',28), ('insert',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
[('insert',6), ('insert',5), ('insert',4), ('insert',3), ('insert',2), ('insert',7), ('insertionSort',NULL)]	6 5 4 3 2 7 5 6 4 3 2 7 4 5 6 3 2 7 3 4 5 6 2 7 2 3 4 5 6 7 2 3 4 5 6 7

d. Example execution

```
>> ./pa3.exe 3 "[('insert',42), ('insert',20), ('insert',17),
('insert',13), ('insert',28), ('insert',14), ('insertionSort',
NULL)]"
[Task 3]
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
```

4. 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 a 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

- `('insert',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
[('insert',56), ('insert',42), ('insert',20), ('insert',17), ('insert',13), ('insert',28), ('insert',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
[('insert',6), ('insert',5), ('insert',4), ('insert',3), ('insert',2), ('insert',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 4 "[('insert',56), ('insert',42), ('insert',20),
('insert',17), ('insert',13), ('insert',28), ('insert',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
```

5. BST Insertion (2 pts)

a. Implement a function that **inserts** 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::insert`
Input: Key of the element to be inserted.
Output: Return `1` if the key already exists in the tree, `0` otherwise.
(If the key already exists, do not insert the element)

c. `task_5` prints
   i. the return for each insertion and
   ii. the results of preorder and inorder traversal of the constructed tree.

d. Example Input & Output

Input	Output
[('insert',4), ('insert',6), ('insert',0)]	0 0 0 4 0 6 0 4 6
[('insert',4), ('insert',-2), ('insert',10), ('insert',9), ('insert',15), ('insert',-5)]	0 0 0 0 0 0 4 -2 -5 10 9 15 -5 -2 4 9 10 15
[('insert',4), ('insert',-2), ('insert',4), ('insert',10), ('insert',15), ('insert',-2)]	0 0 1 0 0 1 4 -2 10 15 -2 4 10 15

e. Example execution

```
>> ./pa3.exe 5 "[('insert',4), ('insert',6), ('insert',0)]"
[Task 5]
0
0
0
4 0 6
0 4 6
```

6. BST Deletion (4 pts)

a. Implement a function that **deletes** an element from a binary search tree (BST). You can modify `bst.cpp` and `bst.h` files for this problem.

b. Input & output of `BinarySearchTree::erase`
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)

c. `task_6` prints
   i. the return for each insertion/deletion and
   ii. the results of preorder and inorder traversal of the constructed tree.

d. Example Input & Output

Input	Output
[('insert',4), ('insert',6), ('insert',0), ('delete',0)]	0 0 0 0 4 6 4 6
[('insert',4), ('insert',-2), (‘delete’,-2), ('delete',-2), ('delete',4)]	0 0 0 1 0
[('insert',4), ('insert',-2), ('insert',10), ('insert',9), ('insert',15), ('insert',-5), ('delete',-5), ('delete',4), ('delete',10)]	0 0 0 0 0 0 0 0 9 -2 15 -2 9 15

e. Example execution

```
>> ./pa3.exe 6 "[('insert',4), ('insert',6), ('insert',0),
('delete',0)]"
[Task 6]
0
0
0
0
4 6
4 6
```