CST 370 – Fall B 2020
Homework 5
Due: 12/08/2020 (Tuesday) (11:55 PM)

How to turn in?

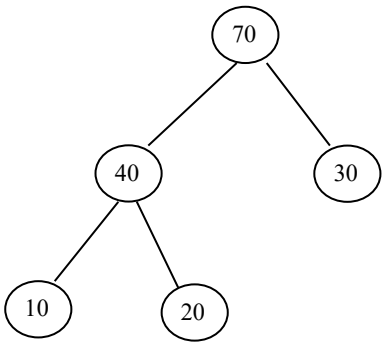
- Submit **three C++ or Java programs (including the bonus program hw5_3)** on the iLearn. When you submit your homework programs, don't forget to include "Title", "Abstract", "ID", "Name", and "Date".
- Note that the **due time is 11:55(PM)**. This is the iLearn's timestamp, not your submission time. Since there could be a long delay between your computer and iLearn, you should **submit early**.

1. Write a C++ (or Java) program called **hw5_1.cpp (or hw5_1.java)** to conduct heap operations.

Input format: This is a sample input from a user.

```
5
10 20 30 40 70
10
displayMax
insert 50
insert 15
deleteMax
delete 40
display
update 2 55
display
update 2 5
display
```

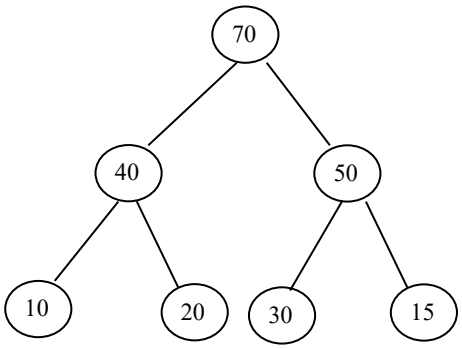
The first line (= 5 in the example) indicates that there are five numbers in the second line (= 10, 20, 30, 40, and 70 in the example). In the program, you can **assume that all numbers in the heap are unique (= no duplicates).** Your program should read the numbers and display if it's a max heap or not. If it's not a max heap, your program should construct a heap with the numbers. Thus, this is the heap built with the input data.



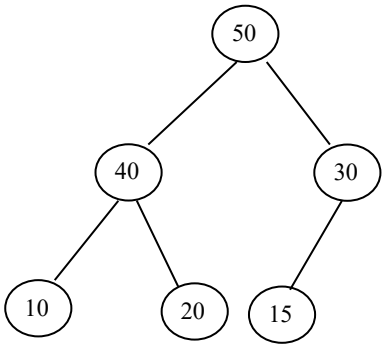
The third line (= 10 in the example) indicates the number of commands you have to conduct to the heap. The commands include "displayMax", "insert" (= insert a number to the heap and do the "heapify" to adjust the heap), "deleteMax", "delete" (= delete a number from the heap and do the "heapify" to adjust

the heap) , “update” (= update a number in the heap and do the “heapify” to adjust the heap), and “display” (= display all nodes in the heap on the screen).

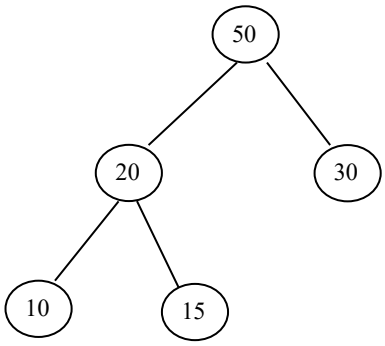
Thus, for the “displayMax” command, your program should display “70” on the screen. After that, your program should insert “50” and “15” to the heap. This is the result after the two insert operations.



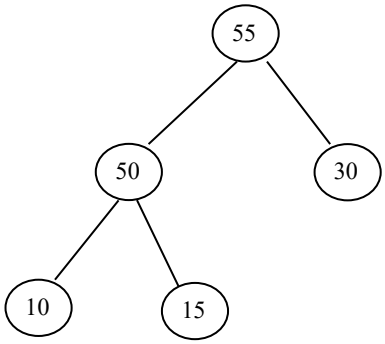
For the “deleteMax” command, your program should delete the max (= 70) from the heap. This is the result.



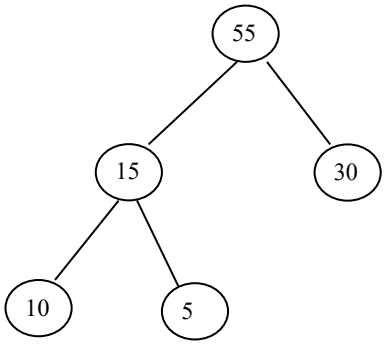
For the “delete” command, your program should search the heap from the root and remove it by overwriting it with the last element in the heap. After that, you should do the “heapify” to adjust the heap. For the sample command (= delete 40), your program should swap 40 with 15. Then heapify the heap. This is the result after the operation. Note that the delete command is not a standard operation of the heap. However, you should implement it for the educational purpose.



For the “update” command, your program should search the heap from the root and update the value in the index with the value provided. After that, you should do the “heapify” to adjust the heap. For the sample command (= update 2 55), your program should update the value in the index 2 (= “20” at the moment) with 55. Then, heapify the heap. This is the result after the operation. Note that the update command is not a standard operation of the heap. However, you should implement it for the educational purpose.



As another example of the “update” command, this is the result after the command “update 2 5”.



Sample Run 0: Assume that the user typed the following lines

```
5
10 20 30 40 70
10
displayMax
insert 50
insert 15
deleteMax
delete 40
display
update 2 55
display
update 2 5
display
```

This is the correct output. For the input data, your program should display that it's not a heap. Then, 70 is the current max of the heap for the command "displayMax". The result (= 50 20 30 10 15) is the result of "display" command after "delete 40". After that, the two results of "55 50 30 10 15" and "55 15 30 10 5" are the results of commands "update 2 55" and "update 2 5".

```
This is NOT a heap.
70
50 20 30 10 15
55 50 30 10 15
55 15 30 10 5
```

Sample Run 1: Assume that the user typed the following lines

```
6
20 10 8 1 3 5
5
display
deleteMax
displayMax
update 4 9
display
```

This is the correct output.

```
This is a heap.
20 10 8 1 3 5
10
10 9 8 5 3
```

2. Write a C++ (or Java) program named **hw5_2.cpp (or hw5_2.java)** that displays the performance of three different sorting algorithms (= **heap sort**, **merge sort**, and **quick sort**) on the screen.

Your program should read the input size from a user and generate random integer numbers of the input size. After that, your program should run the sorting algorithms for the input data and display execution time (= elapsed time) for each sorting algorithm.

For the program, you have to **use an array** to hold the input data of a large size such as 50,000, 100,000, or even more than 1,000,000. **If your program uses a vector or any other container**, you will get **zero without grading**.

For the homework, you can **use any programs available on the Internet for the sorting algorithms**. However, you have to **change them properly to hold the input data in an array**. Also, you have to **cite the source program's website(s) in your head comment**.

This is a sample result of your program:

```
Enter input size: 350
```

```
====================== Execution Time ====================
Heap sort:    0.001234 milliseconds
Merge sort:   0.001234 milliseconds
Quick sort:   0.001234 milliseconds
==========================================================
```

In the sample execution, your program generates 350 random integer numbers. For the homework, you have to display the **"elapsed time" in milliseconds.** Note that the elapsed time in **the sample run (= 0.001234 milliseconds) is not an actual execution time**. It is a meaningless number entered by the instructor. But in your case, your program should display actual execution time.
When you display the elapsed time, **exclude the time to generate the input data**. You have to **measure only the sorting time**. And also, all sorting algorithms should use the same input data.

Note that **we will test your program on AWS**. Thus, you have to **make sure that your program runs well on AWS** which may use a different compiler than your own laptop.

This is another sample result:

```
Enter input size: 500000

====================== Execution Time ====================
Heap sort:    1234.56789 milliseconds
Merge sort:   1234.56789 milliseconds
Quick sort:   1234.56789 milliseconds
==========================================================
```

3. **[Bonus]** This is a bonus problem. **If you pass all test cases of this homework, your score can be "32" points out of "24" points.** Note that **hw5_1** and **hw5_2** are **12 points, respectively**. **hw5_3** is **8 points**.

Write a C++ (or Java) program called **hw5_3.cpp (or hw5_3.java)** to simulate the operations of **linear probing** covered in the lecture. For the sample operation, watch https://youtu.be/yAaUTDuiaqY

Input format: This is a sample input from a user.

```
5
10
insert 17
insert 12
displayStatus 2
tableSize
insert 20
tableSize
search 20
search 15
displayStatus 1
displayStatus 2
```

The first line (= 5 in the example) is the initial size of the hash table. The size will be always a prime number. The second line (= 10 in the example) indicates the number of commands you have to conduct to the hash table. The commands include "insert" (= insert a key to the table), "displayStatus" (= display the status of an entry in the table), "tableSize" (= display the size of the table), and "search" (= search a key in the table).

For the first two "insert" commands, the table will be like below.

Index	Key Value
0	
1	
2	17
3	12
4	

Note that if the load factor becomes **greater than 0.5** after a new insert, you have to conduct the **rehashing**. In other words, you have to find the first prime number after doubling the current table size and move the keys in the current table to the new table. After that, you have to insert the new key value. This is the result after the "insert 20" command.

Index	Key Value
0	
1	12
2	
3	
4	
5	
6	17
7	
8	
9	20
10	

Sample Run 0: Assume that the user typed the following lines

```
5
10
insert 17
insert 12
displayStatus 2
tableSize
insert 20
tableSize
search 20
search 15
displayStatus 1
displayStatus 2
```

This is the correct output. For the "displayStatus" command, your program should display the status of an entry of the table. For example, your program should display "17" for the first "displayStatus 2" command. For the second "displayStatus 2" command, it should display "Empty".

```
17
5
11
20 Found
15 Not found
12
Empty
```

Sample Run 1: Assume that the user typed the following lines

```
7
7
insert 100
insert 16
insert 37
displayStatus 3
displayStatus 2
search 37
tableSize
```

This is the correct output.

```
16
100
37 Found
7
```

Sample Run 2: Assume that the user typed the following lines

```
4229
8
insert 4229
tableSize
insert 1000
insert 2000
insert 3000
insert 4000
displayStatus 0
displayStatus 1
```

This is the correct output.

```
4229
4229
Empty
```