CSE 4304-Data Structures Lab. Winter 2024-25
Date: 03 December 2025
Target Group: All groups
Topic: Heap, Heapsort, Priority Queues.

Instructions:

- Task naming format: fullID_T01L01_2A.c/cpp
- If you find any issues in the problem description/test cases, comment in the Google Classroom.
- If you find any tricky test cases that I didn't include but that others might overlook, please comment. I'll be happy to add them.
- Use appropriate comments in your code. This will help you recall the solution more easily in the future.
- The obtained marks will vary based on the efficiency of the solution.
- Do not use <bits/stdc++.h> library.
- Modified sections will be marked with BLUE color.
- You are allowed to use the STL stack unless it's specifically mentioned to use manual functions.

Group	**Tasks**
1A/1B/2A/2B	1 2 3 4

Task 1: Implementing the basic operations of a Binary Heap.

Given an arbitrary set of numbers of size, your task is to build a **max-heap** from the set of numbers and sort them using Heap-sort.

Take input as long as you don't get -1. For each test case, show the state of the Max-Heap and the Sorted array.

Input	**Output**
4 1 3 2 16 9 10 14 8 7 -1	Max Heap: 16 14 10 8 7 9 3 2 4 1 Sorted: 16 14 10 9 8 7 4 3 2 1
7 9 6 19 8 17 11 2 5 3 13 -1	Max Heap:19 13 17 9 8 6 11 2 5 3 7 Sorted: 19 17 13 11 9 8 7 6 5 3 2

Note:

- Assume that we are using 1-based indexing.
- Use Separate functions for 'heapify', 'Build_max_heap', and 'Heap_sort'.
- You can call the build_max_heap function only once at the beginning of the heapsort function
- Another way of creating the heap can be to insert the values one by one and call build_max_heap each time. Pls don't use this approach. Here our task is to convert the existing array into a heap, not to insert items one by one (this can be a separate task). Pls follow the recommended approach from the slide/CLRS book.

Task 2: Basic Priority Queue Operations

Use the Heap that you created in Task 1 and convert it into a **'Min Priority Queue'**, and implement the following functionalities:

1. int **Heap_Minimum**(int heap[]): Returns the minimum value.
2. int **Heap_extract_min**(int heap[]): Removes the minimum value and returns it.
3. **Min_heap_insert**(int value, int heap[]): Inserts the 'value' into the heap and makes necessary arrangements.
4. **Heap_decrease_key**(int i, int k, int heap[]): Decreases the value at i-th position by an amount of k and makes necessary changes.
5. **Heap_increase_key**(int i, int k, int heap[): Increases the value at i-th position by an amount of k and makes necessary changes.

Input

The first line of input will contain a set of numbers. Show the corresponding min-heap for that.

After that, the input will be like 'function_id necessary_params (if any)'. Show the output and 'state of the heap' after each function call.

Input	**Output**
70 90 60 190 80 170 110 20 50 30 130 -1	Min Heap: 20 30 60 50 70 170 110 190 90 80 130
1	20 20 30 60 50 70 170 110 190 90 80 130
2	20 30 50 60 90 70 170 110 190 130 80
1	30 30 50 60 90 70 170 110 190 130 80
3 45	30 45 60 90 50 170 110 190 130 80 70
4 4 65	25 30 60 45 50 170 110 190 130 80 70
2	25 30 45 60 70 50 170 110 190 130 80
5 1 170	45 50 60 70 80 170 110 190 130 200
3 47	45 47 60 70 50 170 110 190 130 200 80

Note:

- Assume that we are using 1-based indexing.
- Complexity for Heap_extract_min, Min_heap_insert, Heap_decrease_key, Heap_increase_key operations should be O(logN)
- Only use buildmaxheap once for building the initial heap from given input.
- Do not use any STL function.

(Self-study)
C++ has Some built-in functions for performing operations on Queue, Heap/ Priority Queue. Check the following links for a better understanding:

- Basic STL functions to use queues: https://www.geeksforgeeks.org/queue-cpp-stl/
- https://www.geeksforgeeks.org/heap-using-stl-c/
- STL function to swap two queues: https://www.geeksforgeeks.org/queue-swap-cpp-stl/
- https://www.geeksforgeeks.org/heap-using-stl-c/

Task 3: Smash them all

You are given an array of integers ‘stones’ where ‘stones[i]’ is the weight of the i-th stone.
We are playing a game with the stones. On each turn, we choose the **heaviest two stones** and smash them together. Suppose the heaviest two stones have weights x and y, with x<=y. The result of the smash is:

- If x==y, both stones are destroyed.
- If x!=y, the stone of weight x is destroyed, and the stone of weight y has a new weight (y-x).

At the end of the game, there is **at most one stone left**. Return the weight of the last remaining stone. If there are no stones left, return 0.

2 7 4 1 8 1 -1	1	Combine 7,8. State: (2 4 1 1 1) Combine 2,4. State: (2 1 1 1) Combine 2,1. State: (1 1 1) Combine 1,1. State: (1) That's the value of the last stone.
10 10 10 10 10 -1	10	
10 10 5 10 10 10 -1	5	
50 30 10 40 20 -1	10	
50 30 10 40 60 20 -1	10	
10 50 30 10 40 60 20 -1	0	
1 7 5 4 2 2 1 4 8 1 -1	1	
1 7 5 4 2 2 1 4 8 -1	0	
3 3 -1	0	
1 -1	1	

Note:

- Use the Heap_extract_min(/max) function and heap_insert function to implement the task.
- Only use buildmaxheap() once for building the initial heap from given input.
- Do not use any STL function.

Task 4 – Being old is not too bad

At Kings Cross Station, the magical gateway to Hogwarts, a fantastical new ticketing system has been conjured for eager students ready to board the Hogwarts Express. In this enchanted realm, students must fill out a form with their name, age, and house to secure their passage to Hogwarts.

The Ministry of Magical Orderliness has decreed that the wisest and most experienced wizards should be prioritized in this system. As they are busy managing the complexities of the magical world, you have been appointed the responsibility of implementing this seniority-based ticketing system. Although the task may seem daunting for someone just beginning their journey in Data Structures, you're fortunate to have a solid grasp of priority queues. With this knowledge at your fingertips, you're ready to bring order to the chaos!

Input and Output

The input might have three types of options-

1. When you encounter **I**, take a wizard's information in the format: **name, age, house**
2. When you encounter **S**, serve the oldest wizard waiting to be served.
3. When you encounter **X**, terminate the input process.

To be Noted:

- You are **not allowed** to use **STL** heaps/priority queues (however, you're allowed to reuse the functions from Tasks 1 and 2).
- You have to insert the **whole unit** (name, age, house) into the priority queue.
- Any sort of heap/priority queue operation should not cost more than **O(log(n))**.
- Serving the oldest wizard must be done in **constant time**.
- In case of tie, the wizard who filled the form first will be prioritized.

Sample Input	**Output**
I Harry Potter 17 Gryffindor	
I Hermione Granger 18 Gryffindor	
I Ron Weasley 16 Gryffindor	
S	Name: Hermione Granger, Age: 18, House: Gryffindor
I Draco Malfoy 17 Slytherin	
S	Name: Harry Potter, Age: 17, House: Gryffindor
S	Name: Draco Malfoy, Age: 17, House: Slytherin
I Luna Lovegood 15 Ravenclaw	
I Neville Longbottom 16 Gryffindor	
I Dolores Umbridge 38 Slytherin	

S	Name: Dolores Umbridge, Age: 38, House: Slytherin
I Rubeus Hagrid 50 Gryffindor	
I Gilderoy Lockhart 35 Ravenclaw	
S	Name: Rubeus Hagrid, Age: 50, House: Gryffindor
S	Name: Gilderoy Lockhart, Age: 35, House: Ravenclaw
I Ginny Weasley 17 Gryffindor	
I Severus Snape 60 Slytherin	
I Sirius Black 36 Gryffindor	
I Cedric Diggory 19 Hufflepuff	
S	Name: Severus Snape, Age: 60, House: Slytherin
S	Name: Sirius Black, Age: 36, House: Gryffindor
I Albus Dumbledore 120 Gryffindor	
I Minerva McGonagall 75 Gryffindor	
I Tom Riddle 80 Slytherin	
S	Name: Albus Dumbledore, Age: 120, House: Gryffindor
I Bellatrix Lestrange 40 Slytherin	
X	