CSE 4304-Data Structures Lab. Winter 23-24
Batch: CSE 22
Date: Sept 25, 2024
Target Group: All
Topic: Heaps

<u>**Instructions**</u>:
- Regardless of how you finish the lab tasks, you must submit the solutions in Google Classroom. In case I forget to upload the tasks there, CR should contact me. The deadline will always be 11:59 PM on the day the lab took place.
- 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 forget to handle, please comment! I'll be happy to add them.
- Use appropriate comments in your code. This will help you to recall the solution in the future easily.
- 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**
2A	1 2 12 14 [Bonus: 17]
1B	1 2 12 14 [Bonus: 17]
1A	18 19 11 20 [Bonus: 21]
2B	18 19 11 20 [Bonus: 21]
Assignments (all groups)	??, and the ones you weren't included in your respective Lab

<u>Task-01</u>: Implementing the basic operations of a **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 <u>heapsort function</u>

Task 2

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_Minimim**(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.

(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 12 – Capitalistic Summing
Problem Statement
The task is easy; you just need to add a bunch of numbers. But there's a catch! Addition operation requires cost now, and the cost is the summation of those two to be added. So, to add 1 and 10, you need a cost of 11. If you want to add 1, 2, and 3. There are several ways:

```
1 + 2 = 3, cost = 3          1 + 3 = 4, cost = 4          2 + 3 = 5, cost = 5
3 + 3 = 6, cost = 6          2 + 4 = 6, cost = 6          1 + 5 = 6, cost = 6
Total = 9                    Total = 10                   Total = 11
```

I hope that you've already grasped the essence of your task: to sum a set of integers in a way that minimizes the cost.

Input
Each test case will be a stream of positive integers. The input ends when the value of an input is -1. You don't need to process this case.

Output
For each case, print the minimum total cost of addition in a single line.

Input	**Output**	**Explanation**
1 2 3 -1	9	
1 2 3 4 -1	19	
6 5 4 -1	24	
9 5 48 2 4 5 6 3 5 4 -1	224	
3 4 5 4 7 2 3 8 4 5 -1	147	

Task 14 – Bourgeoisie Product
Problem Statement
You have an integer array A which consists of the net worth of N people of a country. For each index i, find the product of the wealth of the richest, second richest, and third richest person in the range [1, i].
Note: Two net-worths can be the same value-wise but they should be distinct index-wise.

Input
The first line contains an integer N, denoting the number of people whose wealth is in the array A. The next line contains N space-separated integers, each denoting the ith integer of the array A, the net worth of the ith person.

Output
Print the answer for each index in each line. If there is no second-richest or third-richest number in the array A up to that index, then print "-1", without the quotes.

Sample Test Case(s)

Sample Input	Output	Explanation
5 1 2 3 4 5	-1 -1 6 24 60	Explanation There are 5 integers 1,2,3,4 and 5. For the first two indexes, since the number of elements is less than 3, so -1 is printed. For the third index, the top 3 numbers are 3,2 and 1 whose product is 6. For the fourth index, the top 3 numbers are 4,3, and 2 whose product is 24. For the fifth index, the top 3 numbers are 5,4 and 3 whose product is 60
8 2 1 3 3 4 1 2 4	-1 -1 6 18 36 36 36 48	 top3 elements: 2,1,3 top3 elements: 2,3,3 top3 elements: 3,3,4 top3 elements: 3,3,4 top3 elements: 3,3,4 top3 elements: 3,4,4

Task 17 – 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 Task 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: Minerva McGonagall, Age: 120, House: Gryffindor
I Bellatrix Lestrange, 40, Slytherin	
X	

<u>Task-18</u>: Implementing the basic operations of a **Heap**.

Given an arbitrary set of numbers of size, your task is to build a **min-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 Min-Heap and the Sorted array.

Input	**Output**
4 1 3 2 16 9 10 14 8 7 -1	Min Heap: 1 2 3 4 7 9 10 14 8 16 Sorted: 1 2 3 4 7 8 9 10 14 16
7 9 6 19 8 17 11 2 5 3 13 -1	Min Heap: 2 3 5 6 7 8 9 11 13 17 19 Sorted: 2 3 5 6 7 8 9 11 13 17 19
12 8 38 23 20 14 1 30 9 -1	Min Heap: 1 8 12 9 20 14 38 30 23 Sorted: 1 8 9 12 14 20 23 30 38

Note:

- Assume that we are using 1-based indexing.
- Use Separate functions for 'heapify', 'Build_min_heap', and 'Heap_sort'.
- You can call the build_min_heap function only once at the beginning of the <u>heapsort function</u>

__Task 19__

Use the Heap that you created in Task 1 and convert it into a ‘**Max Priority Queue**’ and implement the following functionalities:

1. int **Heap_Maximum**(int heap[]): Returns the maximum value.
2. int **Heap_extract_max**(int heap[]): Removes the maximum value and returns it.
3. **Max_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 max-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: 190 130 170 90 80 60 110 20 50 30 70
1	190 190 130 170 90 80 60 110 20 50 30 70
2	190 170 130 110 90 80 60 70 20 50 30
1	170 170 130 110 90 80 60 70 20 50 30
3 45	170 130 110 90 80 60 70 20 50 30 45
4 4 65	170 130 110 50 80 60 70 20 25 30 45
2	170 130 80 110 50 45 60 70 20 25 30
5 1 170	300 80 110 50 45 60 70 20 25 30
3 47	300 80 110 50 47 60 70 20 25 30 45

Note:

- Assume that we are using 1-based indexing.

Task 11

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	

Task 20 –
Problem Statement
Given an array of both negative and positive integers, find out the K-th largest sum of contiguous subarray.

Input
The first line contains an integer N, denoting the number elements in the array. It will be followed by N (positive and negative) integers. And the last number is the value of K.

Output
Print the K-th largest sum of contiguous subarray.

Sample Test Case(s)

Sample Input	**Output**	**Explanation**
3 20, -5, -1 3	14	All sum of contiguous subarrays are (20, 15, 14, -5, -6, -1) so the 3rd largest sum is 14.
4 10, -10, 20, -40 6	-10	

Note : You can't use STL sort for this task

Task 21 – Justice for everyone

The newly formed SAVAGE Committee of the Institute of Ultimate Torture has launched a new initiative this year! Previously, only the student with the highest CGPA received a shiny gold medal, leaving everyone else feeling a bit overlooked. In a quest for fairness (and perhaps a touch of chaos), the Committee has decided to award prizes for every rank based on CGPA. After all, why should just one student have all the glory when we can share the spotlight—and a few giggles—with everyone?

The awards for each position based on their ranks are as follows -

1. First : Shiny Gold Medal
2. Second : First to lose
3. Third : Best among the leftovers
4. i-th : Thank you for participating (i-3)

Input

The first line denotes the number of students (N) followed by the information (**Name, department, CGPA.**) of N students.

Output

Awards received by the students in the order of their inputs

To be Noted:

- You are **not allowed** to use **STL** heaps/priority queues (however you're allowed to reuse the functions from Task 1 and 2).
- You have to insert the **whole unit** (name, department, CG) into the priority queue.
- Any sort of heap/priority queue operation should not cost more than **O(log(n)).**
- You cant use STL Sort

Sample Input	**Output**
5 Andy, CSE, 3.9 Samy, CSE, 3.92 Jerry, CSE, 3.99 Tammy, CSE, 3.94 Ashton, CSE, 3.97 Shey, CSE, 3.91	Thank you for participating (3) Thank you for participating (1) Shiny Gold Medal Best among the leftovers First to lose Thank you for participating (2)