CSE 4304-Data Structures Lab. Winter 2022-23
Batch: CSE 21
Lab: 06
Date: September 11, 2023
Target Group: All
Topic: Heap, Priority Queues

Instructions:

- Regardless you finish the tasks in the lab, you have to submit the solutions in the Google Classroom. In case I forget to upload the tasks there, CR should contact me. The deadline will always be at 11.59 PM of the day in which the lab has taken place.
- Task naming format: fullID_T01L02_2A.c/cpp
- If you find any issues in the problem description/test cases, comment in the google classroom.
- If you find any test case that is tricky that I didn't include but others might forget to handle, please comment! I'll be happy to add.
- Use appropriate comments in your code. This will help you to easily recall the solution in the future.
- 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.

Group	Tasks
2A	1 2 6 11
2B	1 2 12 14
1B	1 2 5 13
1A	1 2 6 14
Assignment	**3, 15 The tasks that were not covered in your lab**

Task-01: Implementing the basic operations of a **Heap**.

Suppose an arbitrary array of size N is given as input. Your task is to build a **max**-heap from the set of numbers and sort them using Heap-sort.

Take input as long 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:

- Use Separate functions for 'heapify', 'Build_max_heap', 'Heap_sort'.
- Only call the build_max_heap function once in the heapsort function (at the beginning to create a heap)!!

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 p, int k, int heap[]): Decreases the value at p-th position by an amount of k and makes necessary changes.
5. **Heap_increase_key**(int key, int amount, int heap[): Increases the value at p-th position by an amount of k and makes necessary changes.

Input

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 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

Mark loves cookies. He wants the sweetness of all his cookies to be greater than the value of **K**. To do this, Mark repeatedly mixes two cookies with the least sweetness. He creates a special combined cookie with:
Sweetness = (1 x Least sweet cookie + 2 x *2nd least sweet cookie*).
He repeats this procedure until all the cookies have a sweetness ≥ **K**.
You are given Mark's cookies. Print the number of operations required to give the cookies a sweetness ≥ **K** Print **-1** if this isn't possible.

Input format
The first line consists of integers **N** representing the number of cookies, and **k**- the minimum required sweetness, separated by a space.
The next line contains **N** integers describing the array ***A*** where A_i is the sweetness of the i^{th} cookie in Mark's collection.

Output format
Output the number of operations that are needed to increase the cookie's sweetness ≥ **K**
Output **-1** if this isn't possible.

Sample Input	**Sample Output**
6 7 12 9 1 3 10 2	2

Explanation
Combine the first two cookies to create a cookie with *sweetness* = 1 x 1 + 2 x 2 = 5
After this operation, the cookies are (3, 5, 9, 10, 12)
Then, combine cookies with sweetness and sweetness, to create a cookie with resulting *sweetness* = 1 x 3 + 2 x 5 = 13
Now, the cookies are (9, 10, 12, 13).
All the cookies have a sweetness >= 7

Thus, **2** operations are required to increase the sweetness.

Note: You should use ***Heap*** to solve this problem. Sorting might be another way of solving this problem, but that will take $O(nlogn)$ in the worst case. But Heap can lead us to a linear solution.

Task 5

Given a set of integers and a value k, return the k most frequent elements. The first line of input contains the set of integers and the following line contains the value of k. Take input until you get -1.

You may return the answer in any order.

Input	**Output**
10 242 23 12 10 12 23 56 10 10 12 23 56 13 18 14 65 34 242 23 12 10 12 23 56 10 242 23 12 10 12 23 56 13 18 14 65 34 -1 5	10 23 12 56 242
3 1 4 4 5 2 6 1 -1 2	4 1 (1 4 is also valid ans)
1 1 1 2 2 3 -1 2	1 2
7 10 11 5 2 5 5 7 11 8 9 -1 4	5 11 7 10
1 2 2 3 3 -1 2	2 3
1 -1 1	1

Note:

- Assume that the numbers are <= 1000
- Assume that the solution is unique
- You must use **priority queue.**

Task 6

Given a string ‘s’, sort it in decreasing order based on the frequency of the characters. The frequency of a character is the number of times it appears in the string.

Return the sorted string. If there are multiple answers, return any of them.

Input	**Output**	**Explanation**
tree	eert	'e' appears twice while 'r' and 't' both appear once. So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid answer
cccaaa	aaaccc	Both 'c' and 'a' appear three times, so both "cccaaa" and "aaaccc" are valid answers. Note that "cacaca" is incorrect, as the same characters must be together.
Aabb	bbAa	"bbaA" is also a valid answer, but "Aabb" is incorrect. Note that 'A' and 'a' are treated as two different characters.
acabcdcee	cccaaeedb	
ddabbbc	bbbddac	‘bbbddca’ also valid

Hint: Utilize the concepts of **priority queue** to solve this task. Store the character with priorities. Then extract them as needed.

Constraints:

- 1 <= s.length <= 5 * 10^5
- s consists of uppercase and lowercase English letters and digits.

Task 11

You are given an array of integers ‘stones’ where ‘storen[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 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 start with a positive number N followed by N positive integers. The input ends when the value of N is 0. You don’t need to process this case.

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

Sample Test Case(s)
Input

```
3
1 2 3
4
1 2 3 4
3
6 5 4
10
9 5 48 2 4 5 6 3 5 4
10
3 4 5 4 7 2 3 8 4 5
0
```

Output

```
9
19
24
224
147
```

Task 13 – Median Search

Problem Statement

Given a sequence of N non-negative integers. Let's define the median of such a sequence. If N is odd the median is the element that stands in the middle of the sequence after it is sorted. One may notice that in this case, the median has a position (N+1)/2 in a sorted sequence if sequence elements are numbered starting with 1. If N is even then the median is the semi-sum of the two "middle" elements of the sorted sequence, i.e. semi-sum of the elements in positions N/2 and (N/2)+1 of the sorted sequence. But the original sequence might be unsorted.

Your task is to write a program to find the median of a given sequence.

Input

The first line of input contains the only integer number N – the length of the sequence. The sequence itself follows in subsequent lines, one number in a line.

Output

You should print the value of the median with **exactly one digit** after the decimal point.

Sample Test Case(s)

Input

4
3
6
4
5

Output

4.5

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 the 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 upto 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 15: Median of Stream of Integers

Given that integers are read from a data stream. Find the median of elements read so for in an efficient way. For simplicity assume, there are no duplicates. For example, let us consider the streams 5, 15, 1, 3 …

After reading 1st element of stream - 5 -> median - 5
After reading 2nd element of stream - 5, 15 -> median - 10
After reading 3rd element of stream - 5, 15, 1 -> median - 5
After reading 4th element of stream - 5, 15, 1, 3 -> median - 4, so on...

Making it clear, when the input size is odd, we take the middle element of sorted data. If the input size is even, we pick the average of the middle two elements in the sorted stream.

Hint: We can use a max heap on the left side to represent elements that are less than the effective median, and a min-heap on the right side to represent elements that are greater than the effective median.

After processing an incoming element, the number of elements in heaps differs utmost by 1 element. When both heaps contain the same number of elements, we pick the average of heaps root data as effective median. When the heaps are not balanced, we select effective median from the root of the heap containing more elements.