Tab 1

# HEAP

## Instructions for students:

- Complete the following classes/methods on Heap.
- You may use any language to complete the tasks (Java / Python).
- All your methods **for an individual task** must be written in one single **.java** or **`.py`** or **`.ipynb`** file. DO NOT CREATE separate files for each task.

## NOTE:

- **YOU CANNOT USE ANY BUILT-IN FUNCTION EXCEPT** len **IN PYTHON. [negative indexing, append is prohibited]**
- **YOU HAVE TO MENTION SIZE OF ARRAY WHILE INITIALIZATION**
- **YOUR CODE SHOULD WORK FOR ALL RELEVANT SAMPLE INPUTS**

**Dear Students, you have been given instructions and driver code for the last 6 labs. For the last two labs of this semester, no driver code will be**

**given. You will develop everything (necessary functions, class, driver code, etc.) in your preferred language (Java or Python).**

## Task 1:

Create a **MinHeap class** with the appropriate insert and delete method. The constructor will receive a capacity parameter to determine the capacity of the heap array. In addition, create a sort method inside the MinHeap class to sort the array (representing heap) **using heapsort.**

**Note:**
The MinHeap Class will have two instance variables; An Array & the size of the heap. Moreover, it'll have 4 functions/methods; insert(), swim(), extractMin(), sink(), sort().
**The instance variables must be Private**.

**Hint:** Use a size variable inside the heap class to maintain the size of the heap. Note that the length of the array and the size of the heap are not the same!

## Task 2:

Complete task 1 for MaxHeap.

**Note:**
For MaxHeap class, the instance variables & instance methods would be the same as MinHeap except for extractMin(), instead it'll be named extractMax().
**The instance variables must be Private**.

# Task 3:

You are given an array of n tasks, where the value of index i represents the processing time for i-th task. You are also given m machines, each capable of processing one task at a time. Your goal is to distribute the tasks among the machines such that the overall completion time for each machine is minimized. Represent the output as an array where each index corresponds to a machine, and the value at each index is the total processing time of tasks assigned to that machine.

**Input Format:**

- **tasks:** An array of integers where each integer represents the processing time of a task.
- **m:** An integer representing the number of machines.

**Output Format:**

- An array of integers of size m, where the value at index i represents the total processing time of tasks assigned to the i-th machine.

**Sample Given Input:**
tasks = [2, 4, 7, 1, 6]
m = 4

**Sample Output:**
[2, 4, 7, 7]

**Explanation:**
To simplify the use of a heap, we focus only on the current load of each machine (not its index). The min-heap will store the current loads of the machines. Initially, all machines will have a load of 0.
Once all tasks are assigned, the heap will contain the loads of all machines. The array of this heap will represent the output.

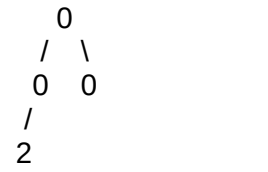
**Steps to Solve:**

- ➔ Initialize a min-heap object with m length, all set to 0 (representing the load of each machine).
- ➔ For each task, do the following:
  - ◆ Extract the smallest load from the heap.
  - ◆ Add the task's processing time to this load.
  - ◆ Reinsert the updated load into the heap.
- ➔ The final array representation for your heap is the answer.

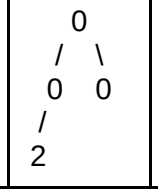
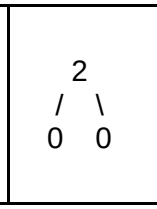
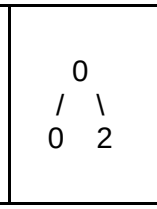
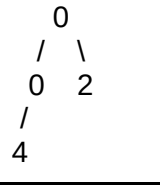
**Example Walkthrough:**
**Initial Heap:**

| **Heap Array** | [0, 0, 0, 0] |
|---|---|
| **Heap Visual** | 0<br>/ \<br>0 0<br>/<br>0 |

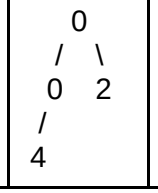
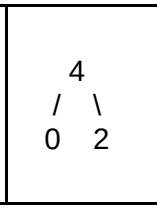
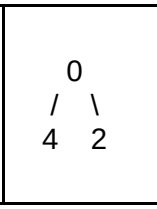
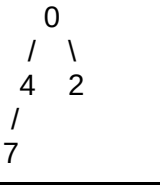
**Task 1 (Processing Time = 2):** Assign to the machine with load 0 (current minimum load).

| | Extract the smallest | Add the extracted with current & reinsert |
|---|---|---|
| **Heap Array** | 0 extracted<br>After extraction: [0, 0, 0, None/null] | 0+2=2 is inserted<br>After insertion: [0, 0, 0, 2] |
| **Heap Visual** |  |  |

**Task 2 (Processing Time = 4):** Assign to the machine with load 0 (current minimum load).

| | Extract the smallest | | | Add the extracted with current & reinsert |
|---|---|---|---|---|
| **Heap Array** | 0 extracted<br>After extraction: [0, 0, 2, None/null] | | | 0+4=4 is inserted<br>After insertion: [0, 0, 2, 4] |
| **Heap Visual** |  |  |  |  |

**Task 3 (Processing Time = 7):** Assign to the machine with load 0 (current minimum load)

| | Extract the smallest | | | Add the extracted with current & reinsert |
|---|---|---|---|---|
| **Heap Array** | 0 extracted<br>After extraction: [0, 4, 2, None/null] | | | 0+7=7 is inserted<br>After insertion: [0, 4, 2, 7] |
| **Heap Visual** |  |  |  |  |

**Task 4 (Processing Time = 1):** Assign to the machine with load 0 (current minimum load).

| | Extract the smallest | | | Add the extracted with current & reinsert | | |
|---|---|---|---|---|---|---|
| **Heap Array** | 0 extracted<br>After extraction: [2, 4, 7, None/null] | | | 0+1=1 is inserted<br>After insertion: [1, 2, 7, 4] | | |
| **Heap Visual** | 0<br>/ \<br>4 2<br>/<br>7 | 7<br>/ \<br>4 2 | 2<br>/ \<br>4 7 | 2<br>/ \<br>4 7<br>/<br>1 | 2<br>/ \<br>1 7<br>/<br>4 | 1<br>/ \<br>2 7<br>/<br>4 |

**Task 5 (Processing Time = 6):** Assign to the machine with the smallest load, that is 1, and add the current processing time (6) with this.

| | Extract the smallest | | | Add the extracted with current & reinsert |
|---|---|---|---|---|
| **Heap Array** | 1 extracted<br>After extraction: [2, 4, 7, None/null] | | | 1+6=7 is inserted<br>After insertion: [2, 4, 7, 7] |
| **Heap Visual** | 1<br>/ \<br>2 7<br>/<br>4 | 4<br>/ \<br>2 7 | 2<br>/ \<br>4 7 | 2<br>/ \<br>4 7<br>/<br>7 |

**Final Output:**
Finally, the array representation of the heap is [2, 4, 7, 7]

**Hints:**

1. Tasks must be assigned one at a time to the machine with the current smallest load.
2. Use a heap (you have built before) to efficiently find the machine with the smallest load.
3. You can create a function that will receive tasks and m and return the final heap for simplicity

# Task 4:

You are given an array of integers and a number k. Your task is to find the top k largest elements from the list. Use a max-heap to solve this problem efficiently.

**Input Format:**

- nums: An array of integers.
- k: An integer representing the number of largest elements to find.

**Output Format:**

- An array of k integers representing the largest elements in descending order.

**Sample Input:**
nums = [4, 10, 2, 8, 6, 7]
k = 3
**Output:**
[10, 8, 7]

## Explanation:

To solve this, we use a max-heap because it allows efficient retrieval of the largest elements. The process involves:

1. Insert all numbers from the input list into a max-heap.
2. Create a new array with size k.
3. Extract the maximum k times and store it to the new array to get the top k largest elements.
4. Return the result array in descending order.

## Steps to Solve:

1. Build a max-heap where the length of array will be equal to length of nums.
2. Extract the maximum k times and store the values in the newly created result array.
3. Return the result array.

## Hints:

1. You must use a max-heap (created above) to solve the problem.
2. k is always smaller than or equal to the size of the array.
3. You can create a function that will receive the nums array and k for simplicity.