Birla Institute of Technology and Science, Pilani Hyd Campus
CF F211: Data Structures and Algorithms
2nd Semester 2024-25 Lab No: 10
Priority Queue, Binary Heap, Job Scheduling

General Instructions

- You should use STL only for the program where it is mentioned explicitly. For other programs you should solve the given task without using it.
- In addition to the methods already specified in the given code, you may also come up with your own methods to execute the tasks.

Program 1: (Prog1.cpp attached)

In the lecture class, you were taught how we can Sort a List or Sequence using a Priority Queue (PQ-Sort). Depending on whether we take a Min-Priority Queue or a Max-Priority Queue, the sorted list order will vary.
Given that P is the priority queue and S is the input list, below is the C++ code to sort S using P:

```
/* Ref Lect number 24 */
while (!S.empty()){
    int e = S.front();
    S.pop_front();
    P.push(e);
}
while (!P.empty()){
    int e = P.top();
    S.push_back(e);
    P.pop();
}
```

```
class C
{
  public:
    // Overriding the comparison operator
    // Telling the Compiler how to compare any two items.

    int operator()(const int a, const int b){
          return a > b; // for a min-priority queue
    }
};
```

Sometimes, it is important to implement a custom Comparator method to help the compiler in deciding how to compare (sort) two given types or more importantly user defined classes. If we choose to design P as a Min-Priority Queue, then the above Comparator (class C) must be passed to the declaration of P. So, if we want to sort the list in ascending order, we should initialize P as follows:

```
priority_queue<int, vector<int>, C> P;
```

The output of the Prog3.cpp is as shown below:

```
Input List:-
10 5 7 14 2 6
List sorted in Ascending Order:-
2 5 6 7 10 14
List sorted in Descending Order:-


Input Points:-
[ (2, 7) (7, 4) (11, 2) (15, 9) ]
Points sorted in increasing order of Y coordinates:-


...Program finished with exit code 0
Press ENTER to exit console.
```

Task: Implement the missing code: class D to write a custom comparator class to create a Max-Priority Queue, Implement the sortDescending() method to sort the list using Max-Priority Queue and also, implement the sortByYCoordinate() method to sort this list of points in increasing order of their Y-coordinates.

Program 2: (Prog2.cpp attached)

The attached code already has **sort()** method that implements Heap Sort for a Min-Heap but specific functions to support the sorting procedure are not implemented. Given a **partially complete Binary Min Heap (Prog1.cpp),** here are your tasks:

- **Task 1**: Complete **extractMin()** method to remove and return the minimum item in the Heap.
- **Task 2**: Implement **bubbleDown()** method to perform bubbling-down (heapify) operation for the element at the given index. Refer to Lecture slides if needed.

The driver code is already available. You should get the output as shown below:

Output:

```
Heap Elems: [3, 6, 4, 7]
Min Elem = 3
3 removed!
Heap Elems: [4, 6, 7]
Min Elem = 4
4 removed!
Heap Elems: [6, 7]
Heap Elems: [1, 6, 8, 7]
Min Elem = 1

Input List:-
7 4 -2 15 9 11
Sorted List:-
-2 4 7 9 11 15


...Program finished with exit code 0
Press ENTER to exit console.
```

Do we need to write a custom implementation of MAX_HEAP as well?

NO, given a MIN_HEAP, we can also simulate it to work as a MAX_HEAP. Here's how we can do it:

Given input List = [7, 4, -2, 15, 9, 11]

Convert all items to their negatives: [-7, -4, 2, -15, -9, -11] and initialize the Min Heap with this input.

Now, extractMin() will return -15, we invert the sign again (change back to original) = 15.

This way our Min Heap will behave as a Max Heap.

Now that you solved both the tasks in Program 2, use your Min-Heap class implementation to execute the task given in Program 3.

Program 3: (Prog3.cpp attached)

In the lecture slides you were taught about the application of Heaps to solve the problem of Job scheduling in Operating Systems.

Problem: There are **M** CPUs and **N** jobs. As soon as an i'th CPU (0 <= i < M) becomes idle, a job will be allocated. You are given that each j'th job (0 <= j < N) has its own CPU time required to finish execution. Here is your task:

- **Task 3**: Implement **longestJobFirst()** method to implement the Longest Job First Scheduling Algorithm. Take reference from *shortestJobFirst()* method to execute this task.

The driver code is already available. You should get the output as shown below:

Output: