**CMPSC 412 – Lab-4** (50 points)

**Priority queue, and Heap Sort**

**Due date: 2/24/2022**

**Lab Exercises:**

The priority queue is an abstract data type that contains the following methods:

insert(item, priorityValue)

Inserts item into the priority queue with priority value priorityValue.

peek()

Returns (but does not remove) the item with highest priority in the priority queue.

delete()

Removes and returns the item with highest priority in the priority queue.

changePriority(item, newPriority)

Changes the priority of an item to a new priority value.

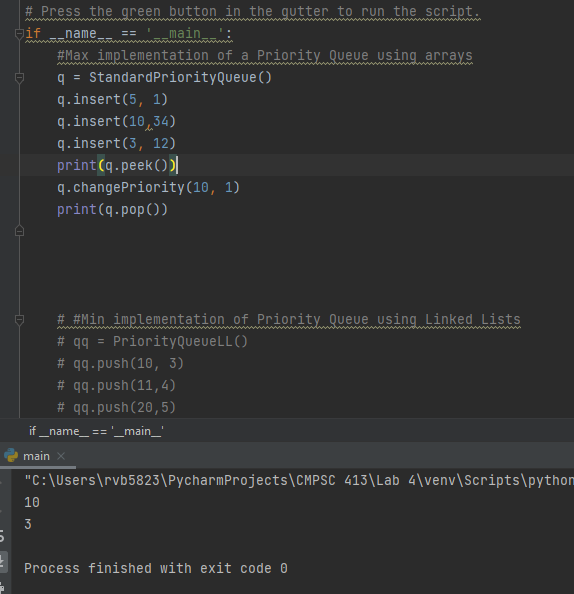
**Exercise-1:**

Write down the algorithm and implement a priority queue (both min and max) using an **array** of elements. Determine the runtime for each of the following:

1. In the worst case, describe the runtime to insert an item into the priority queue.
2. In the worst case, describe the runtime to remove the element with highest priority.
3. In the worst case, describe the runtime to change the priority of an element (find an element and change the priority of the element).

Show an example for each.

1. In the worst case, the insert will take n log n time in order to execute. The reason for this is because every time we append a new item to our array, we use Tim-sort on the entire array which executes in n log n
2. In the worst case, popping an element will execute in constant time. This is because it is a single operation being executed, regardless of the number of elems in the array.
3. In the worst case, changePriority will execute in N + N log N, which is the same as N log N, due to our sorting after every change



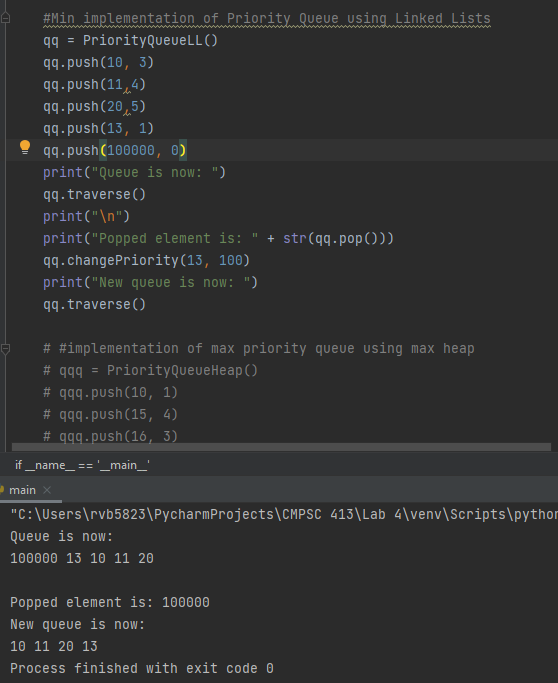
**Exercise-2:**

Write down the algorithm and implement a priority queue (both min and max) using a **linked list** of elements. Determine the runtime for each of the following:

1. In the worst case, describe the runtime to insert an item into the priority queue.
2. In the worst case, describe the runtime to remove the element with highest priority.
3. In the worst case, describe the runtime to change the priority of an element.

Show an example for each.

1. In the worst case, it will take O(N) time to traverse through the entire LL and perform an insertion
2. In the worst case, it will take O(1) time to remove the highest priority element as it will always be the first element.
3. In the worst case, it will take O(2N) or O(N) time to change the priority of an element. This will occur because we traverse the entire list to remove the element and then possibly traverse it again to reinsert it in the correct location.



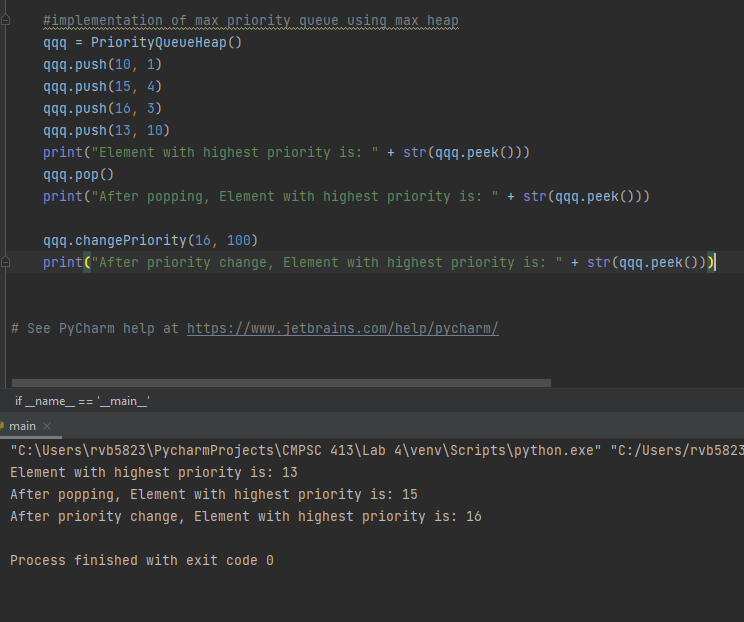
**Exercise-3:**

Write down the algorithm and implement a priority queue (both min and max) using a **heap tree-based** data structure (both min and max). Determine the runtime for each of the following:

1. In the worst case, describe the runtime to insert an item into the priority queue.
2. In the worst case, describe the runtime to remove the element with highest priority.
3. In the worst case, describe the runtime to change the priority of an element.

Show an example for each.

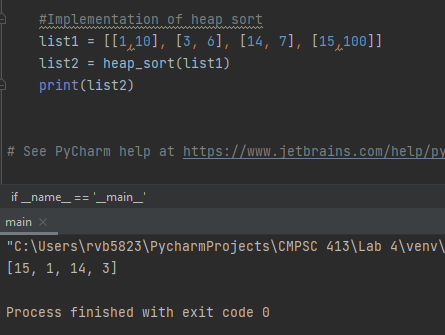
1. In the worst case, it will take O (log N) time to perform an insertion. This is because the item will be appended to the end of the array and then bubbled up to its correct location which will require log N comparisons.
2. In the worst case, it will take O(logN) to remove an element from this priority queue. This is because the first most element is popped and replaced with a child node. The empty space is then replaced by the last node and then bubbled up (log N operations)
3. In the worst case, it will take log N operations to change the priority of an element. Based on the new priority, we will either bubble the element up or down one side of the heap. However, in my implementation, I first iterate through the entire array until the element is found, so it can take up to O(N) time



**Exercise-4:**

Write down the algorithm and implement a heap sort (both ascending and descending) using heap data structure. Determine the runtime for each of the following.

1. In the worst case, describe the runtime to sort in ascending order.
2. In the worst case, describe the runtime to sort in ascending order.
   1. If using a max heap, It will take O(N log N) to sort in descending order. We will pop a total of n elements, and after each pop we have to rebalance our tree which we said takes log N time. This makes the total time complexity for sorting O(N log N)
   2. The same will hold true for sorting in ascending order using a max heap. If you use the opposing heap structure (i.e. a max heap to sort ascending order), it will take O(N^2) to sort.



**Exercise-5:**

Tabulate to compare the time complexity of insert, remove and change priority operations for array/linked list/ heap priority queues.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Insert | Remove | Change |
| Array | N log N | 1 | N log N |
| Linked List | N | 1 | N |
| Heap | Log N | Log N | N |

**Exercise-6: Conclusion**

Write a paragraph about what have you learnt from this lab exercises?

From this lab exercise I learned that there are multiple ways to implement a priority queue. Ideally, a heap is the best overall data structure to use for implementing a priority queue as it allows for fast insertion times as well as moderately fast removal times. However, this does not mean a heap implementation is best. For example, if you know your situation calls you to access the highest priority element far more often than you insert into the queue, a Linked List implementation may be more optimal.

**Deliverables:**

* Codes
* Report with algorithms, screenshots of results for each exercise and conclusion (exercise-6).
* Attach the codes as appendix in your report.