**Execution Document**

**Data Structures used:**

* **ArrayList**
* **PriorityQueue**

**ArrayList**

The ArrayList's implementation builds on an array that doubles in size when the maximum size is reached. (It's very poorly implemented for now, just the methods needed in this project are used.)

Because of the array structure, you can add at the last index by just accessing **Array.length - 1**.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Add at end | Delete | Get | Find |  |  |
| Wanted Time: | O(1) | O(n) | O(1) | O(n) |  |  |
| **Time:** | **O(1)** | **O(n)** | **O(1)** | **O(n)** |  |  |
| Wanted Space: | O(n) |  |  |  |  |  |
| **Space:** | **O(n)** |  |  |  |  |  |

**PriorityQueue**

The priority queue is much similar to the ArrayList, with a array that doubles in size when the limit is reached.

The structure used is a binary heap (every node has at most two children) and the heap can be defined to be a max or min heap based on the Comparator provided during construction.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Add | Delete | Get | Find | Poll | Peek |
| Wanted Time: | O(log(n)) | O(log(n)) | O(1) | N/A | O(log(n)) | O(1) |
| **Time:** | **O(log(n))** | **O(log(n))** | **O(1)** | **O(n)** | **O(log(n))** | **O(1)** |
| Wanted Space: | O(n) |  |  |  |  |  |
| **Space:** | **O(n)** |  |  |  |  |  |