**Analysis**

**Time complexity Analysis**

Class: Heap

Method: Heapify



| ***Instruction*** | ***Times*** |
| --- | --- |
| 1 int index = 0; | 1 |
| 2 while (hasLeftChild(index)) | n |
| 3 int smallerChildIndex = getLeftChildIndex(index); | n - 1 |
| 4 if (hasRightChild(index) && comparator.compare(rightChild(index), leftChild(index)) < 0) | n - 1 |
| 5 smallerChildIndex = getRightChildIndex(index); | n -1 |
| 6 if (comparator.compare(heap[index], heap[smallerChildIndex]) < 0) | n -1 |
| 7 break; | n -1 |
| 8 else | n -1 |
| 9 swap(index, smallerChildIndex); | n - 1 |
| 10 index = smallerChildIndex; | n - 1 |

**T(A) = 1 + n + 8(n-1)**

**T(A9 = 8n + n - 8 + 1**

**T(A) = 9n - 7**

**Time complexity: O(n)**

Class: **MainController**

Method: updateTaskList

****

| ***Instruction*** | ***Times*** |
| --- | --- |
| taskViewer.setContent(null); | 1 |
| VBox taskList = new VBox(); | 1 |
| for (int i = 0; i < taskArray.length; i++) | n |
| Task task = taskArray[i]; | n – 1 |
| TaskItem taskItem = new TaskItem(task, this); | n - 1 |
| taskList.getChildren().add(taskItem); | n - 1 |
| VBox.setVgrow(taskItem, Priority.ALWAYS); | n - 1 |
| taskViewer.setContent(taskList); | 1 |
| taskViewer.setFitToWidth(true); | 1 |

**T(A) = 1 + 1 + 1 + n + 4 \* (n - 1) + 1 + 1**

**5 + n + 4n - 4**

**5n + 1**

**Time complexity: O(n)**

**Space complexity Analysis**

Class: addTaskMenuController

Method: submitTask()



| **Type** | **Variable** | **Size** |
| --- | --- | --- |
| Input | title | n |
| content | n |
| priority | 32 |
| userCategory | n |
| deadLine | 100 |
| e (exceptions) | 200 |
| Auxiliar | datePicked | 24 |
| date | 24 |
| hasError | 16 |
| Output | none | - |

Total Input Space Complexity = n + n+ 32 + 100 + 16 + 24 + 24 + 200 bits = 396 bytes + 3n

The spatial complexity of this algorithm is 𝑂(n)

**Class:** Queue

**Method:** Remove



| **Type** | **Variable** | **Size** |
| --- | --- | --- |
| Input | node | n |
| Auxiliar | pointer | n |
| prev | n |
| data | n |
| Output | data | n |

Total Input Space Complexity = n + n + n + n + n + n = 6n = O(n)

The spatial complexity of this algorithm is 𝑂(n)