CPSC 331
TA: Harnoor K.

Shaheryar Syed
30052162

Writeup for Assignment 3 – Heap:

Based on lecture material we know that in theory the heap sort algorithm follows the time complexity of O(nlogn) for it's average and worst case. However we will dive into the creation methods at the same time:

For the heapify heap creation method we can see an average-case and worst-case time complexity of O(n). This is because when creating a max heap for the heapify method we start at the last non-leaf node in a tree and move towards the root while also maintaining the heap property. This requires n operations to achieve. We can also see this holding true when looking at the result where the average case can be the random array, and the worst case can be the sorted array. We see for the heapify creation method the number of required swaps was ~728, and for a sorted array ~992. This difference in swaps is not significant enough to say that the time complexities of the average case and worst-case scenario's are any different, and these results show a time complexity of O(n) for both cases as one result is not dramatically different than the other.

For the one by one heap creation method we can see an average-case and worst case time complexity of O(n log(n)). This is because when using this method elements are added at the end of the tree and then moved up to maintain the heap property, as such each element when moving up must be compared and swapped throughout the height of our tree with the other elements. This requires a lot more operations than a traditional heapify heap creation method. This is also shown by our results where the average case would be the random array, and worst case the sorted array. The random array needed ~1228 swaps for creation whereas the sorted array required ~7987 swaps. Clearly then there is a significant difference between the two and the time complexity for both show that the swaps follow an O(nlog(n)) time complexity.

For the heap sort algorithm we can see an average-case and worst-case time complexity of O(nlog(n)). This is because inherently the heap sort algorithm requires starting at the root, removing it, then running through the heapify process to ensure each element maintains the heap property. By itself heapifying top down, this is clearly a O(log(n)) configuration because the heapify method has to go through the entire height of the tree through a traversal process which requires log n time. However, there are also n elements in the tree, so the whole time complexity becomes O(nlog(n)).