**Report of assignment 4**

**Design and Analysis of Algorithms**

# Teacher: Aidana Aidynkyzy

Made by: Alpamys Abylaikhan

My partner: Dovutbekov Ulyqbek

Group: SE-2430

**1.Algorithm Overview**

The sort I am analyzing is Min Heap Sort, it follows the heap data structure. It takes an array and then puts the minimum element of each sub-heap to the end of the array. The result is sorted array in descending order.

**2.Complexity Analysis**

**Time complexity:**

Best case (Big Omega nlogn): we need to do heapify calls in any case.

Average case (Theta nlogn): for random arrays.

Worst case (O(nlogn)): it happens if there are reverse sorted arrays.

**Space complexity:**

The algorithm sorts in place, on the input array itself.  
Recursive part is O (log n), Iterative heapify part is O (1).

So total complexity becomes:  
T(n) = T(n/2) + O (1)

A = 1, b = 2.   
n ^ (log\_2 1) = n ^ 0 = 1

It’s Master Theorem Case 2, 1=1, so it resolves to:

O (n logn)

**3.Code review and Optimization**

1)Recursive heapify introduces unnecessary stack overhead.

2)The algorithm always performs heapify even if the array is already sorted.

3) I recommend using iterative heapify instead to avoid memory overhead.

4) One more recommendation is to add conditional termination, so that it exits the heapify if no swaps.

Code quality:

Pros: clean codes and comments where needed.

Cons: minor rebundancy in heapify comparison.

**4.Empirical results**

**Note that benchmark results were executed on: 100, 1000, 10000, 100000 input sizes.**

The graphic shows how HeapSort’s running time increases as the array size increases. The operating time increases too, however at small values there are fluctuations. Overall, this graph proves that algorithms scales according to theorized O (n log n) complexity.

**5.Conclusion**

Overall, I can say that the given heapsort is clean and efficient, albeit having some places to optimize. Its complexity analysis is O (n log n), notice that O is exact here. Its strength lies in clear structure and correct handling od edge cases. Its weaknesses are that recursive heapify can lead to unnecessary memory usage, therefore leading to garbage collection overload and also no early termination if the array is already sorted. It can be optimized for further space efficiency.