CS2XB3 - Lab 05

Group Number: 06
Lab Session: L02

Group Members:

Sarvesh Paarthasarathy - 400285716 | paarthas@mcmaster.ca
Joshua Sam Varugheese - 400255799 | varugj1@mcmaster.ca

Building Heaps (50%): (Implementation given in heap.py)

Complexity Estimations:

Function Name: build_heap1
Complexity Estimation: O(N Log(N))

Reasoning: The time complexity of sink function is O(log(n)) (Seen from lecture) and the build_heap1 method has a time complexity of O(n log(n)) because we apply the sink method roughly n/2 times.

Function Name: build_heap2
Complexity Estimation: O(N Log N)

Reasoning: The time complexity of insertion in heap is O(log(n)) (Seen from lecture) and build_heap2 method has a time complexity of O(n log(n)) because we insert n times while building the heap.

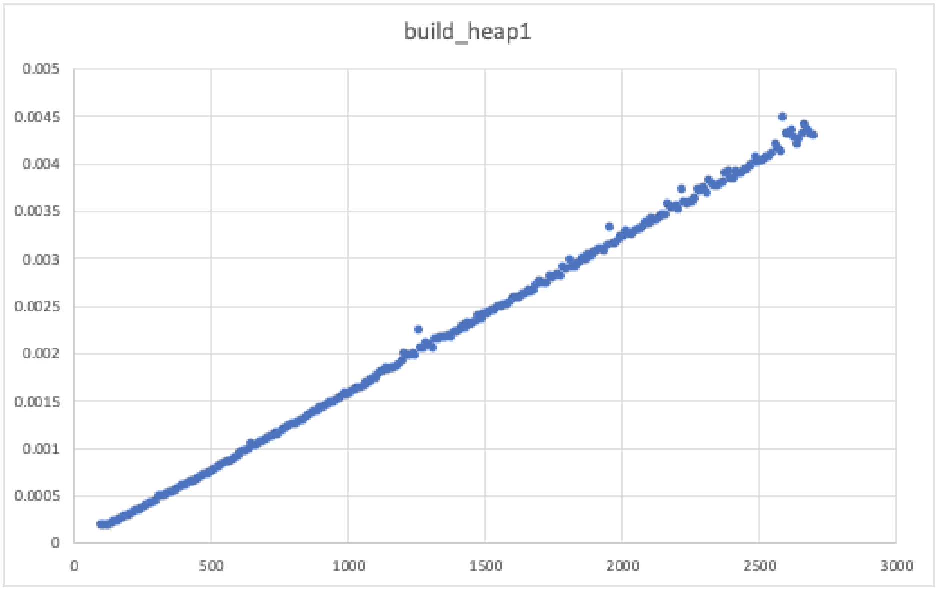
Function Name: build_heap3
Complexity Estimation: O(N Log N)

Reasoning: The time complexity of sink as seen earlier is O(log(n)). Since we are calling the method sink on every node of the heap, the build_heap3 method has a time complexity of O(n log(n)).

<u>Timeit experiments and graphs:</u>

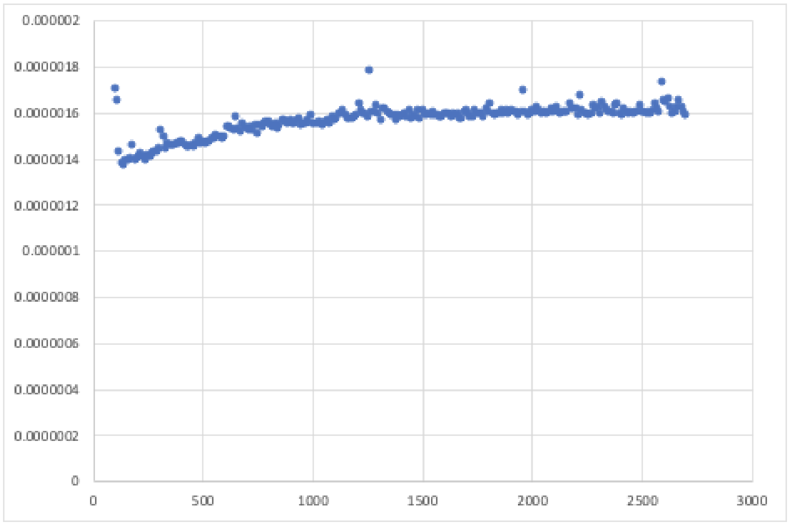
1. Function Name: build_heap1

Graph:



From this plot, we can assume that it is either a **linear** graph or a **(n log(n))** graph.

Graph after dividing by N:

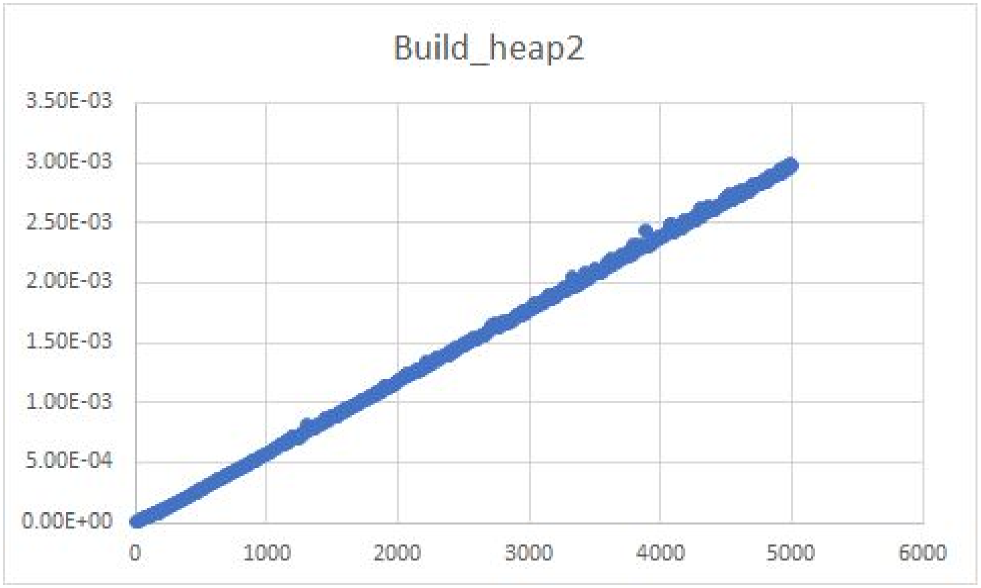


The resulting graph is constant (ignoring all the outliers). Thus we can conclude that the graph is constant and the time complexity for the **function build_heap1** is **O(n).**

---

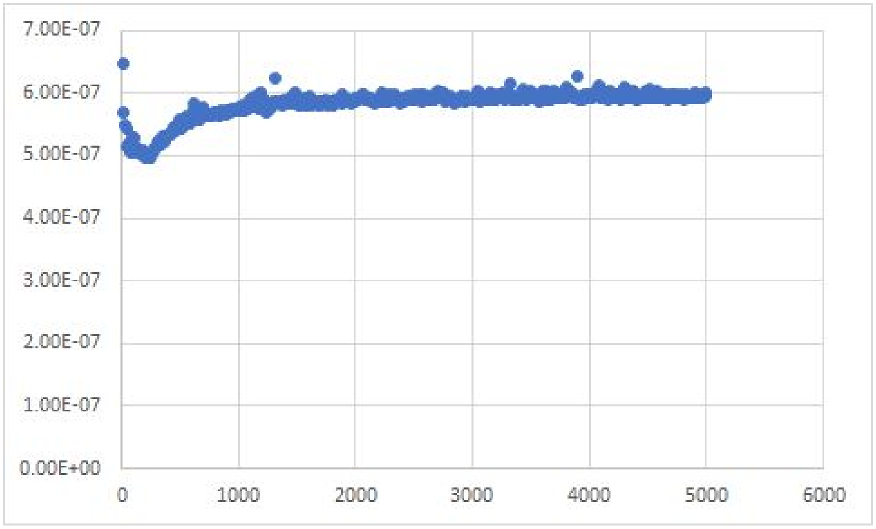
2. Function Name: build_heap2

Graph:



From this plot, we can assume that it is either a **linear** graph or a **(n log(n))** graph.

Graph after dividing by N:

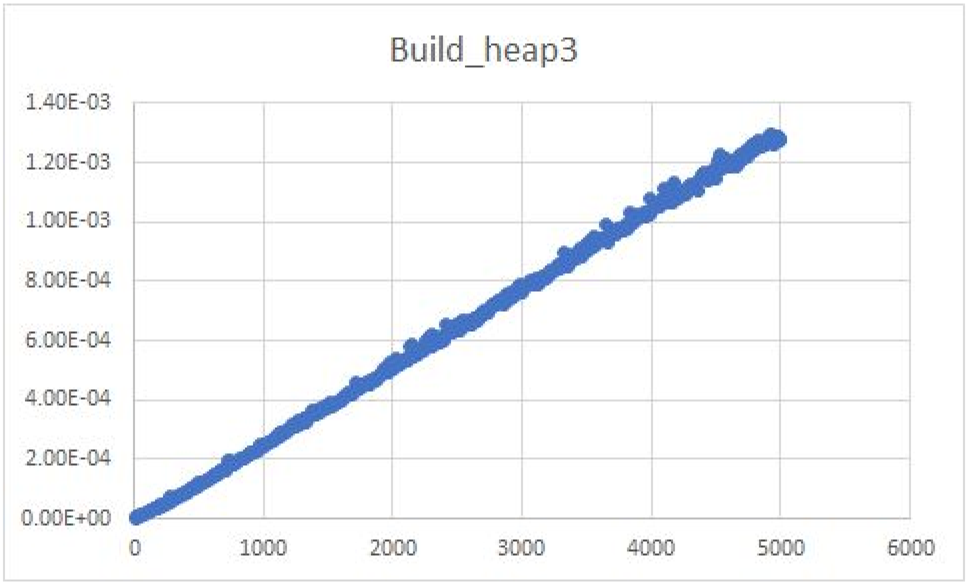


The resulting graph is constant (ignoring all the outliers). Thus we can conclude that the graph is constant and the time complexity for the **function build_heap2** is **O(n)** similar to the previous function.

---

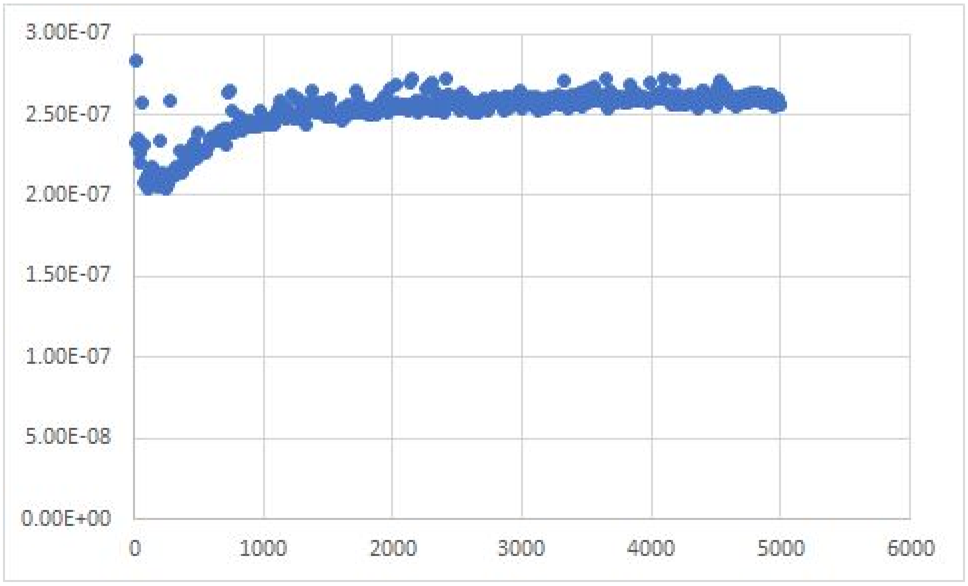
3. Function Name: build_heap3

Graph:



From this plot we can assume that graph is either a **linear graph** or a **(n log(n)) -** graph.

Graph after dividing by N:



The resulting graph is constant (ignoring all the outliers). Thus we can conclude that the graph is constant and the time complexity for the **function build_heap3** is **O(n),** similar to the previous function.

In build_heap3 we are iterating through all the nodes of the heap and calling the sink method.This is done n times. We know that the first sink method will swap the root node with the largest element. This continues until the max-heap is formed. For the subsequent swaps after the first swap, the sink is performed from the root node which is redundant. The optimization we could include is ignoring the nodes on which the sink method is already called. This could result in better performance and run time complexity.

K-Heap (50%): (Implementation given in k-heap.py.)

An advantage k-heap has over normal binary heap is that it uses less memory because the heap tree structure is more broader or we can say that the height of the heap is smaller comparatively. Another advantage is that a parent node can have multiple children and is not limited like a binary tree. Having a heap with multiple children has its own applications like reduced run-time complexity.

The asymptotic behaviour of the sink function in k-heap is believed to be $\log_k(n)$ because each node has upto k children.