# CS/SE 2XB3 Lab 5 Report
# Enrolled in CSL02

Wang, Mingzhe
400316660
wangm235@mcmaster.ca

Li, Xing
400292346
li64@mcmaster.ca

Moon, Hyosik
400295620
moonh8@mcmaster.ca

July 26, 2021

## Contents

**1 Building Heaps** **2**
1.1 Estimation . . . . . 2
1.2 Experiment Results. build_heap_1, build_heap_2 . . . . . 3
1.3 Experiment Results. build_heap_3 . . . . . 3
1.4 What Causes Poor Performance of build_heap_3? . . . . . 4

**2 k-heap** **5**
2.1 Advatages and Disadvantages . . . . . 5
2.2 Asymptotic Complexity of Sink . . . . . 5

# 1 Building Heaps

## 1.1 Estimation

- build_heap_1 Bottom-up. Since each sink has worst case $O(\log n)$ and there are $n/2$ loops, the time complexity of "build_heap_1 Bottom-up" is $O(\frac{1}{2}n \log n) = O(n \log n)$.

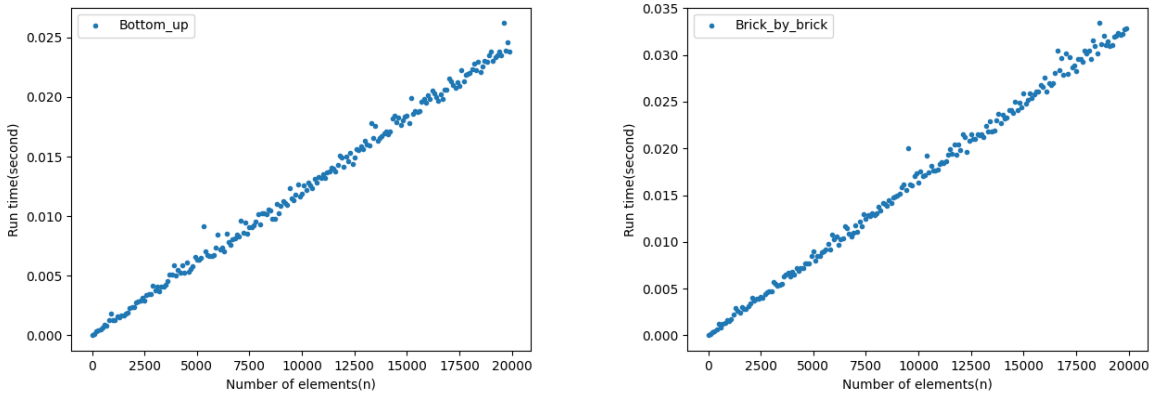
- build_heap_2 Brick-by-brick. As each insert has worst case $O(\log n)$ and there are n inserts, the time complexity of "build_heap_2 Brick-by-brick" is $O(n \log n)$.

- build_heap_3 Sink-top-down. As each sink has worst case $O(\log n)$ and Sink-top-down calls n sinks at least, the time complexity of "build_heap_3 Sink-top-down" is at least $O(n \log n)$. However, as it cannot guarantee the result is heap. When the biggest value is at the bottom of the heap, it should repeat the heap process until it goes to the top. Because each time it goes up just one level, it needs $\lfloor logn \rfloor$ repetitions of the for-loop. Thus, the time complexity of "build_heap_3 Sink-top-down" could be as bad as $O(n \log^2 n)$.

## 1.2 Experiment Results. build_heap_1, build_heap_2

In order to obtain the experimental data of Bottom_up(heap1) and Brick_by_Brick(heap2) , we tested them by increasing the number of elements from 100 to 20,000, step by 100. We ran them 5 times at each point and took the average runtimes.

Different from what we expected, both of them showed $O(n)$ time complexities. This is because in the case of Bottom_up(heap1), when we set the $h$ as the height of a heap(0 at the bottom), and $n$ is the total number of elements of a heap, then it has the number of $\lceil \frac{n}{2^{h+1}} \rceil$ elements at each height. Because each element has at most *one* swap, it showed that the time complexity of $\Sigma_{h=1}^{\lfloor logn \rfloor} \lceil \frac{n}{2^{h+1}} \rceil * O(1) = O(n * \Sigma_{h=1}^{logn} \frac{1}{2^h}) = O(n)$. Similarly, in the case of Brick_by_Brick(heap2), each element has the time complexity of $O(h)$ at a height $h$. As a result, it showed the time complexity of $\Sigma_{h=0}^{\lfloor logn \rfloor} \lceil \frac{n}{2^{h+1}} \rceil * O(h) = O(n * \Sigma_{h=0}^{logn} \frac{h}{2^h}) = O(n * \Sigma_{h=0}^{\infty} \frac{h}{2^h}) = O(n)$. [2]

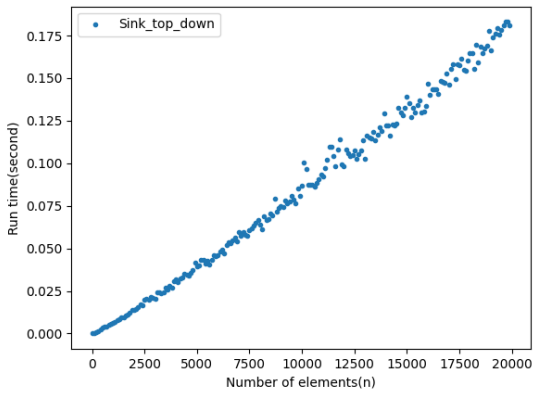



(a) Bottom_up(build_heap_1) (b) Brick_by_Brick(build_heap_2)

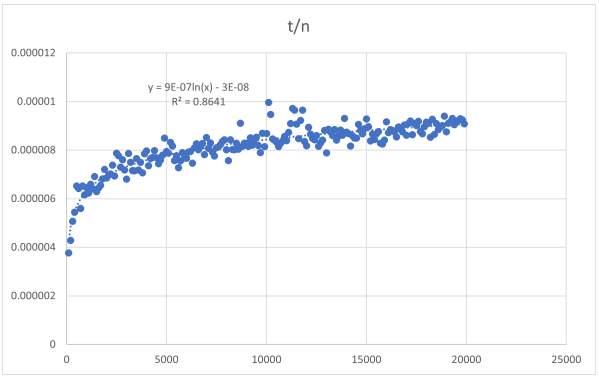
Figure 1: Time complexities of heap1, heap2

## 1.3 Experiment Results. build_heap_3

We tested Sink_top_down(heap3) under the same test condition. As we expected, it showed the time complexity of $O(nlogn)$ (Figure 2a, 2b). When we see the Figure 2a, it seems to be a leaner function but its runtime is longer than the previous heaps (heap1, heap2). When we divdie it $n$, it shows a logarithmic function. When we take log at each x, y value, then its slope is about 1.128 which means it's not an exact linear function (Figure 3).




(a) Sink_top_down(heap3)




(b) Runtime / n

Figure 2: Time complexity of heap3

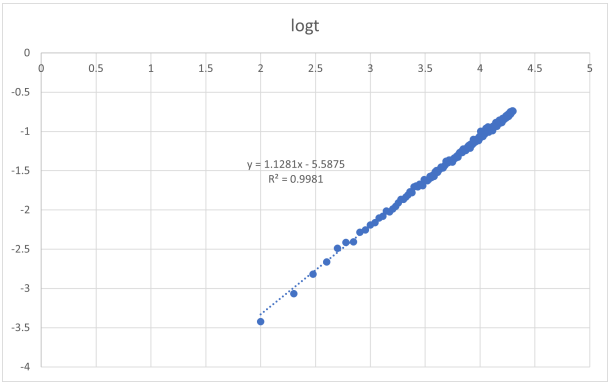



Figure 3: Time complexity of heap3 (log)

## 1.4 What Causes Poor Performance of build_heap_3?

For build_heap_3, an element of the implementation that causes poor performance is that at the end of each iteration of heapifying every node, it needs to check if the result structure is a heap by calling `is_heap`. However, `is_heap` itself is a function that needs $O(n)$ complexity. Thus, this behavior will induce poor performance of build_heap_3.

As the hint remind us, we notice that the maximum number of times you need to heapify every node before the whole thing is a heap should be no lager than the complete binary tree (heap)'s height. Thus, we find a improvement that at the end of each iteration of heapifying every node we do not perform `is_heap` check anymore, instead we simply set

the number of times of heapifying every node to be the heap's height for all the cases. By doing this, we abandon the `is_heap`'s complexity $O(n)$ in the inner loop, which could improve the implementation.

Note:

- The improved implementation can be found in the `heap.py` named `build_heap3_opt`.
- For calculating the complete binary tree (heap)'s height by its number of nodes. The function we used is as the following:

  ```
  math.ceil(math.log2(n + 1)) - 1
  ```

# 2 k-heap

## 2.1 Advatages and Disadvantages

A `k-Heap` can be used for creating a dictionary. In the case of english, it has 26 letters, and strings can be represented by the combination of alphabets. We can organized words systematically by using 26-heap data structure. Likewise, a `k-Heap` is useful to organize data which have many subtrees. [3]

From the perspective of time complexity, when a `k-Heap` swaps a value, it just need to compare the value with the parent. Because a `k-Heap`'s height is $O(\log_k n)$, swap's time complexity is very low compared to a binary tree.

On the other hand, when it comes to sink, it has to compare all k-children at each level, increasing the number of comparisons by $O(k \log_k n)$. However, even though a `k-Heap` requires more comparisons compared to a binary heap, practically it's much faster than a binary heap because a binary heap typically requires more cache misses and virtual memory page faults than a `k-Heap`, each one taking far more time than the extra work incurred by the additional comparisons from a `k-Heap`. [1]

## 2.2 Asymptotic Complexity of Sink

When there are $n$ nodes, the maximum height of a `k-Heap` is $\log_k n$. Each node calls $k$ children, so sink's time complexity is $O(k \log_k n)$. Similar to binary heap, in the case of build_heap, it has the time complexity of $\frac{n}{2} * O(1) = O(n)$. [4]

# References


[1] d-ary heap,
https://stackoverflow.com/questions/29126428/binary-heaps-vs-d-ary-heaps

[2] Binary heap,
https://en.wikipedia.org/wiki/Binary_heap

[3] k-ary heap,
https://en.wikipedia.org/wiki/D-ary_heap

[4] m-ary tree,
https://www.quora.com/What-are-advantages-of-using-a-d-ary-tree-instead-of-a-binary-tree