Xingya Ren
260784116

# COMP 251 Assignment 1 Long Answer

## 3.1 O(1)

Proof:

Proof by induction

hypothesis: the complexity of operation 1 is O(1)

Base case: when $j = 2$, the complexity is O(1)

since we are inserting the first node

Assume the complexity is O(1)

if $(j+1)$ is prime we need to insert $((j+1)^2, (j+1))$

if the step before is operation 1

then the node inserted was $(j^2, j)$

$j^2 < (j+1)^2$

So the node $((j+1)^2, (j+1))$ is the largest in the heap,

it takes O(1) to insert it.

else if the step before is operation 2.

then the root would be $(p^{\alpha+1}, p)$

$p^\alpha = j$

$p^{\alpha+1} = p^\alpha \cdot p = j \times p$

$j \times p < j \times j \quad (\because p < j)$

$\therefore jp < j^2 < (j+1)^2$

So the node $((j+1)^2, (j+1))$ is the largest in the heap.

it takes O(1) to insert it at the bottom.

□

3.2 $O(\log n)$

Proof:

The heap is a min heap and in operation two we need to heapify the heap. The pseudo code for that:

```
MIN_HEAPIFY( A, i )
    l ← leftNode (i)
    r ← rightNode (i)
    if l ≤ A.heap_size and A[l] < A[i]
    then smallest ← i
    else
        smallest ← i
    if r ≤ A.heap_size and A[r] < A[i]
        smallest ← r
    if smallest ≠ i
        then exchange A[i] with A[smallest]
        MIN-HEAPIFY (A, smallest)
```

Time determine if there is a conflict and find the smallest children is $\theta(1)$ and the time to fix the subtree rooted at one of $i$'s children is $O$(size of subtree) and it is given that the height is $h = \theta(\log n)$.

Thus, the time complexity for Operation 2 is $O(\log n)$ □

3.3

. number of prime numbers concerned $\leq \sqrt{n}$

$q_i$ : the prime number we are looking at

$$N = \sum_{i=1}^{\sqrt{n}} \log_{q_i}(n)$$

$\log_a n < \log_b n$ when $a > b$

each term in $\sum_{i=1}^{\sqrt{n}} \log_{q_i}(n)$ is $\leq \log_2(n)$

$$\sum_{i=1}^{\sqrt{n}} \log_{q_i}(n) < \sum_{i=1}^{\sqrt{n}} \log_2(n) = \sqrt{n}\log_2(n)$$

$N \leq O(\sqrt{n} \times \log_2 n)$ $\sqrt{n} \times \log_2 n$ is negligible compared to $n$.

$$\frac{N}{n} \leq \lim_{n\to\infty} \frac{\sqrt{n}\log n}{n}$$

$$\lim_{n\to\infty} \frac{N}{n} = 0$$ □

3.4 From 3.1, the complexity for operation 1 is $O(1)$

3.2, the complexity for operation 2 is $O(\log n)$

Given the complexity for primality check is $O(\sqrt{n})$

For iteration 1: $O(1) + O(\sqrt{n}) = O(\sqrt{n})$

For iteration 2: $O(\log n) + O(\sqrt{n}) = O(\sqrt{n})$

Start with 3 because the complexity of the very first step (inserting (4,2)) is $O(1)$

From 3 to n we iterate $(n-2)$ times. So the complexity is $O(n-2) = O(n)$

⇒ total complexity : $O(n)\,O(\sqrt{n}) = O(n\sqrt{n})$ □

3.5 Prove: for a given heap of height $h$ with $n$ nodes, we have $h = \Theta(\log n)$.

When the heap is a binary heap:

For the lower bound:

when the tree is full: $n = 2^{h+1} - 1$

so $n \leq 2^{h+1} - 1$

$n+1 \leq 2^{h+1}$

$\log_2(n+1) \leq \log_2(2^{h+1})$

$\log_2(n+1) \leq h+1$

$h \geq \log_2(n+1) - 1 > \log_2 n - 1$ if $n \geq 1$ and $h \in \Omega(\log n)$

For the upper bound:

For depth $i = 0, \cdots, h-1$ there are $2^i$ keys

there is at least one key at depth $h$.

then, $n \geq 1 + 2 + 4 + \cdots + 2^{h-1} + 1$

$n \geq 2^h - 1 + 1$

$h \leq \log_2 n$ and $h \in O(\log n)$

Therefore, since $h \in \Omega(\log n)$ and $h \in O(\log n)$,

$h = \Theta(\log n)$

If the heap is not a binary heap but a d-ary heap:

Assume the root is at level 0. Then the number of nodes at a completely filled level $i$ would be $d^i$.

So the number of nodes up to and including level $k$ would be:

$$\sum_{i=0}^{k} d^i = \frac{d^{k+1} - 1}{d-1}$$

Then the last node – the $n^{th}$ node – can either be the last node at level $k$, or in an incomplete level $k+1$.

Thus, $\frac{d^{k+1}}{d-1} \leq n < \frac{d^{k+2}-1}{d-1}$

$\Rightarrow k \leq \log_d (n(d-1)+1) - 1 < k+1$

Equality is only if the last node is the last leaf on level k, which also has distance k from the root. If not, then there is a level k+1, then the log term would not be an integer and applying the ceiling operator would give the right height of k+1. Thus, if the last element is at position n, the height of the heap is:

$$h = \lceil \log_d (nd - n + 1) \rceil - 1$$

Thus, the height of a given d-ary heap is also $h = \Theta(\log n)$

□

THANK YOU FOR YOUR TIME! :)

I have discussed with Yuexin (Virna) Xi 260716384
Yonggue Kim 260531176
Sean Smith 260787775
about Question 3.3 and 3.4,
and referred to all of the Reddit posts on those two questions
and Question 3.2 is based on the Lecture Slides on Heap.