CS271: Data Structures

Name: YOUR NAMES HERE

Instructor: Dr. Stacey Truex

Unit 1: Practice 1

Time Breakdown

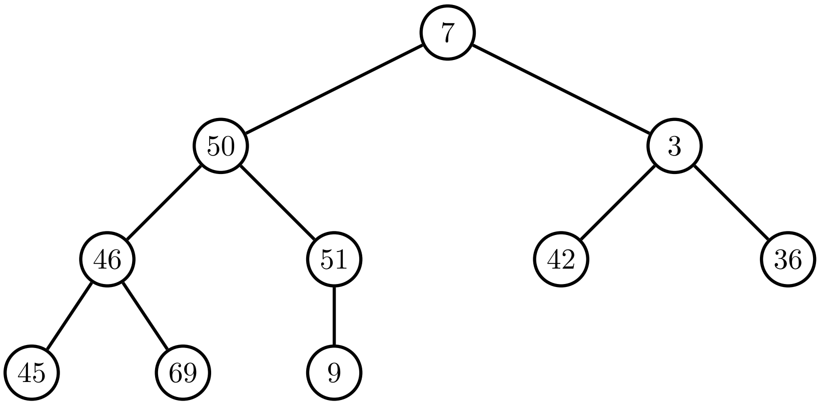
Each group member certifies that 1 hour (60 minutes) – please note that **no more than 1 hour** is required – was spent *collaboratively* on Unit 1 Practice 1 material. The breakdown of how those 60 minutes were spent is as follows:

-

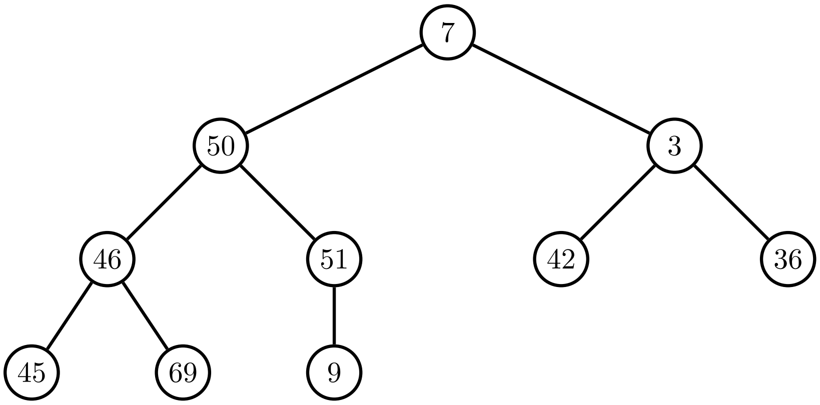
Practice

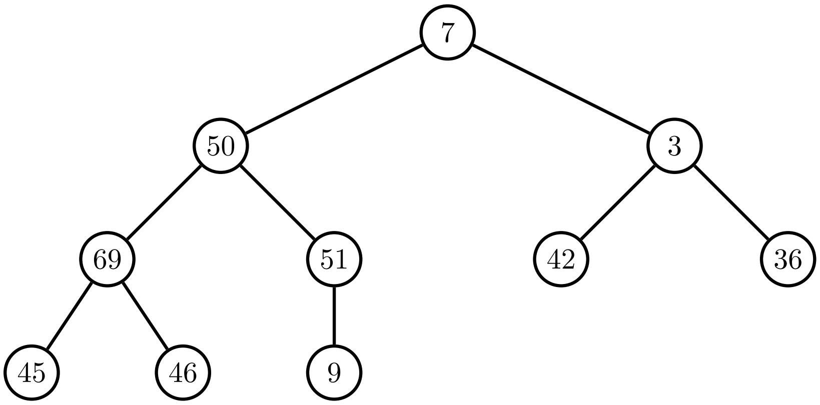
1. Draw the following array as a nearly complete binary tree:

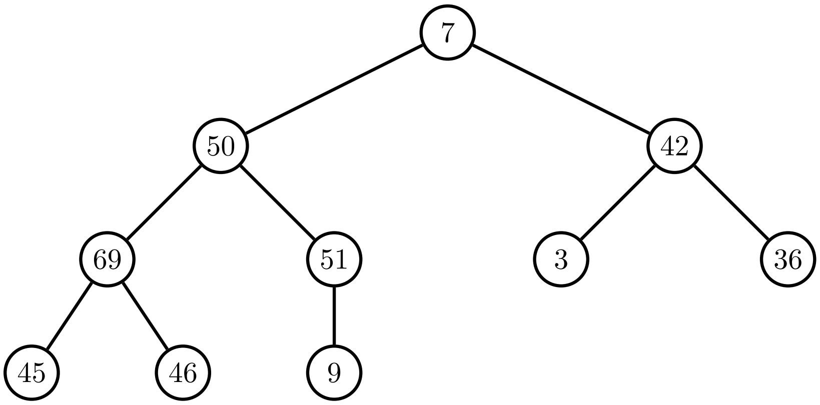
$$[7, 50, 3, 46, 51, 42, 36, 45, 69, 94]$$

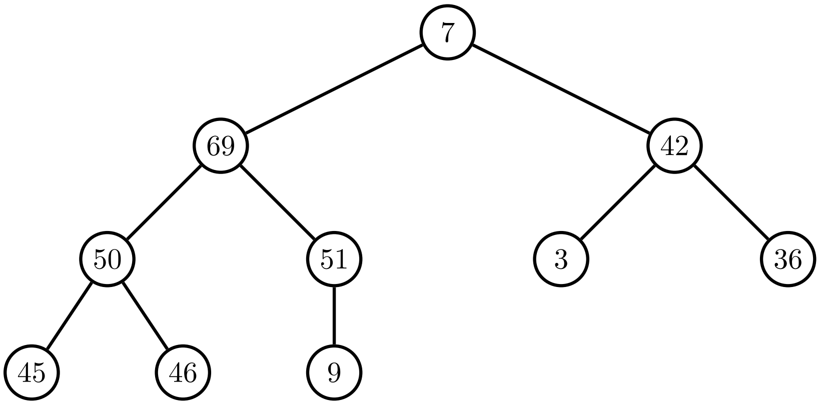


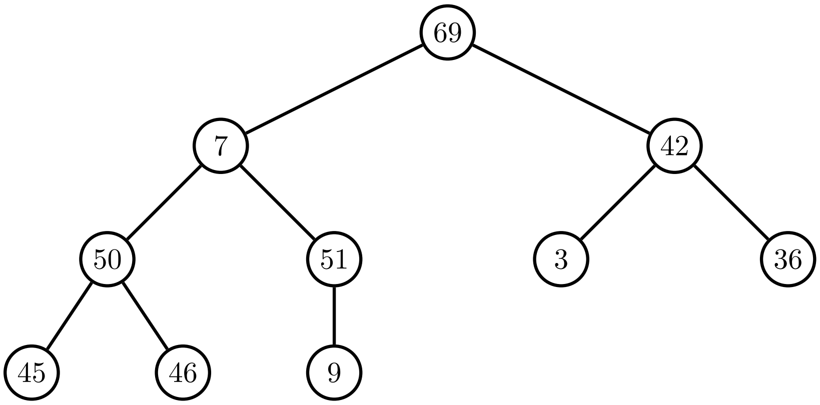
2. Fill in the following images corresponding to the state of the binary tree from (1) at the end of each iteration of the `for` loop in the Build-Max-Heap algorithm. List which values were swapped in the call to Max-Heapify.

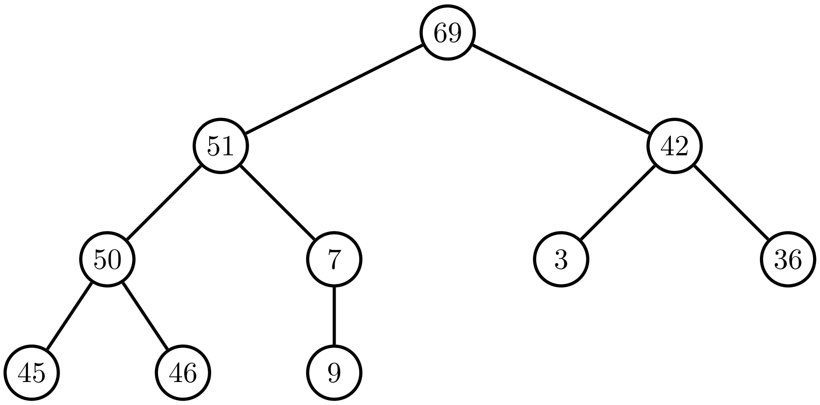

7
50
3
46
51
42
36
45
69
9

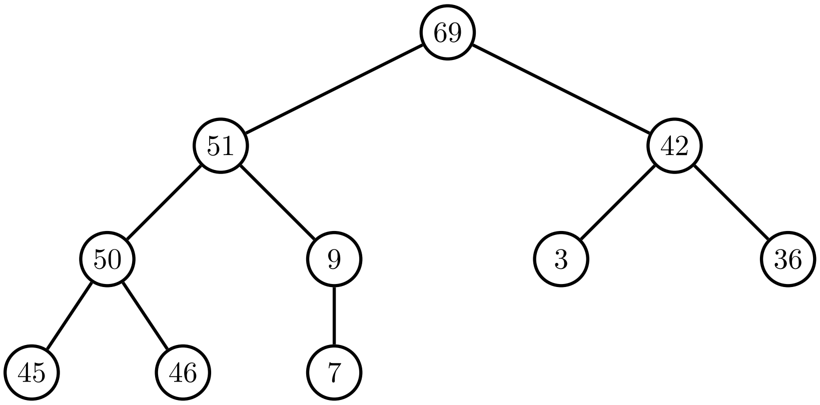

7
50
3
69
51
42
36
45
46
9


7
50
42
69
51
3
36
45
46
9


7
69
42
50
51
3
36
45
46
9


69
7
42
50
51
3
36
45
46
9


69
51
42
50
7
3
36
45
46
9



(a) Iteration 1:

   i. Swapped **values**: Compared 51 and 9. Nothing Swapped

(b) Iteration 2:

   i. Swapped **values**: Compared 46 and (45, 69). Swapped 46 and 69

(c) Iteration 3:

   i. Swapped **values**: Compared 42 and (3, 36). Swapped 3 and 42

(d) Iteration 4:

   i. Swapped **values**: Compared 50 and (60, 51). Swapped 50 and 60

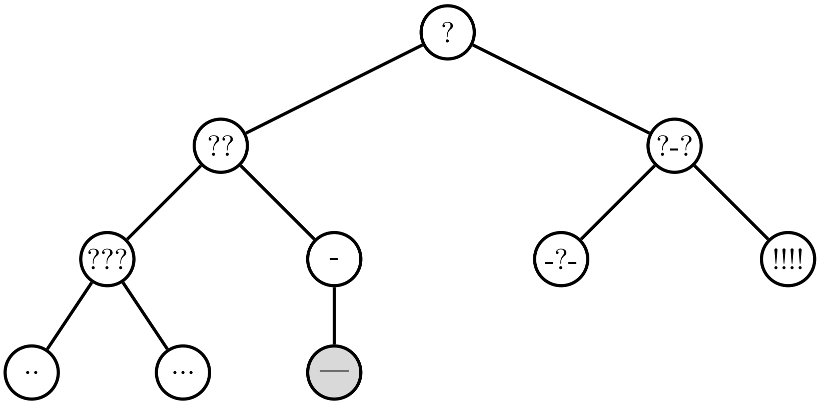
(e) Iteration 5:

   i. Swapped **values**: Compared 7 and (69, 42). Swapped 50 and 60

   ii. Swapped **values**: Compared 7 and (50, 51). Swapped 50 and 60

   iii. Swapped **values**: Compared 7 and 9. Swapped 7 and 9.

3. Fill in the following images corresponding to the state of the binary tree from (2) at the end of each iteration of the `for` loop in the Heapsort algorithm. Indicating which values are not considered to be in the heap by setting the node to outheap (from inheap). List which values were swapped in the call to Max-Heapify.



(a) Iteration 1:

  i. Swapped **values**:

  ii. Swapped **values**:

(b) Iteration 2:

  i. Swapped **values**:

4. Explain (don't prove) why $\lfloor n/2 \rfloor$ calls to Max-Heapify in Build-Max-Heap results in a runtime of $O(n)$ while $n-1$ calls Max-Heapify in Heapsort results in a runtime of $O(n \log n)$.

   The build-max-heap runs heapify bottom-up. Most nodes are near the leaves, so their heapify calls are cheap (often O(1) as heapify terminates instantly for leaf nodes). Only a few nodes near the root cost up to O(log n).

   However, for heap sort keep switch the root and the leaf node and uses heapify from the root constantly, which results in O(log n) for all iteration. Since this happens n-1 times, the total cost is O(n log n).

5. Analysis of Max-Heapify

   (a) Write a recurrence equation for Max-Heapify in the worst-case. (Hint: the child of the root of an n-node heap is the root of a heap with $\leq \frac{2n}{3}$ nodes)

   $$T(n) = \mathrm{T}(\tfrac{2n}{3}) + \text{b (for some positive constant b)}$$

   (b) Find a tight upper bound for your recurrence and prove that it is correct using induction.

   Level of the tree: $log_{\frac{2}{3}} n$

   Number of nodes in level i: 1

   Cost per node: 1

Total Cost in Tree: $log_{\frac{2}{3}} n$
Guess: $T(n)$ = O(log n)

Inductive Hypothesis: for all k in [1, n), T(k) ≤ c * log(k) for some positive constant c
Inductive Step:

$$\begin{aligned}
T(n) &\leq T\left(\left\lfloor \tfrac{2n}{3} \right\rfloor\right) + b \\
&\leq c \log\left(\tfrac{2n}{3}\right) + b \\
&= c\left(\log n + \log \tfrac{2}{3}\right) + b \qquad (\log(ab) = \log a + \log b) \\
&= c \log n + c \log \tfrac{2}{3} + b \\
&= c \log n - c \log \tfrac{3}{2} + b \\
&\leq c \log n \qquad \text{if } c \log \tfrac{3}{2} \geq b \text{ (e.g., } c \geq \frac{b}{\log(3/2)})
\end{aligned}$$

Let $a > 0$ be the constant runtime when $n \leq 3$. Base Case(s):

- $T(3) = T(2) + b = a + b \leq c \cdot \log 3 + b$ when $a \leq c \cdot \log 3$

For constants $c = \max\left(\frac{b}{\log(3/2)}, \frac{a}{\log 3}\right)$ and $n \geq 3$, $T(n) \leq c \log n$ holds. Therefore, $T(n) = O(\log n)$.

6. Analysis of Build-Max-Heap

   (a) What is the loop invariant of Build-Max-Heap?

   Lemma 1 (loop invariant)**.** *...*

   (b) Prove the correctness of Build-Max-Heap using your invariant.

   Proof. **Initialization:** Before the first iteration, i =?. In this case, the loop invariant says ... This is true because ...

   Maintenance: Assume that the loop invariant is true before some iteration i, that is, ... During iteration i, ... Before the next iteration of the loop, i is ... Once this happens, our conclusion is rewritten as "..." Therefore, the maintenance step holds. □

   Lemma 2 (loop termination condition)**.** *After the for loop terminates, ...*

   Proof. In the final iteration of the outer for loop, i =?. Therefore, ... □

   Theorem 1. Build-Max-Heap *correctly ...*

Proof. According to Lemma 2, after the loop, ... Therefore, we can conclude that Build-Max-Heap correctly ... □