1. Consider a Binary Search Tree $T$ with vertices $a, b, c, d, e, f, g, h$ and $\text{ROOT}[T] = a$ and with the following values ($N$ means NIL)

<table>
<thead>
<tr>
<th>vertex</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>N</td>
<td>e</td>
<td>e</td>
<td>a</td>
<td>d</td>
<td>g</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>left</td>
<td>h</td>
<td>N</td>
<td>N</td>
<td>e</td>
<td>c</td>
<td>N</td>
<td>f</td>
<td>N</td>
</tr>
<tr>
<td>right</td>
<td>d</td>
<td>N</td>
<td>g</td>
<td>N</td>
<td>b</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

(a) Which is the successor of $c$. Illustrate how the program $\text{SUCCESSOR}$ will find it.

Solution: The successor of $c$ is $f$. As $c$ has a right child $g$, $\text{SUCCESSOR}$ will call $\text{MIN}[g]$ which will go to the left as long as possible, ending (in one step) at $f$.

(b) Which is the minimal element? Illustrate how the program $\text{MIN}$ will find it.

Solution: $h$. Start at root $a$. Go to left: $h$. Go to left: NIL. Return $h$.

(c) Illustrate the program $\text{DELETE}[e]$

Solution: There are two approaches (equally correct) to $\text{DELETE}[x]$ when $x$ has two children. One can effectively replace $x$ by the maximum of its left tree or the minimum of its right tree.

Solution 1: $e$ has a left child $c$. Applying $\text{MAX}[c]$ gives $g$. $g$ has a left child $f$. So we splice $f$ into $g$’s place by resetting $right[c] = f$ and $p[f] = c$ and we put $g$ in $c$’s place, setting $left[d] = g$, $left[g] = c$, $right[g] = b$, and $p[g] = d$.

Solution 2: $e$ has right child $b$. Applying $\text{MIN}[b]$ gives $b$ itself. We splice $b$ into $e$’s place by resetting $p[c] = b$ and $left[b] = c$ and $p[b] = d$ and $left[d] = e$

2. Draw binary search trees of height 2,3,4,5,6 on the set of keys $\{1,4,5,10,16,17,21\}$.

Solution: For height 2 with 7 it must be balanced so 10 is the root with children 4, 17 which in turn have children 1, 5 and 16, 21. For height 6 it must be a line. There are two ways: 1 as root and then taking right children go down 4, 5, 10, 16, 17, 21. (Or take 21 as root
and go in reverse.) For the other heights there are many solutions. E.g.: height 3: 10 has left child 5 with left child 4 with left child 1 and 10 has right child 17 which has two children 16, 21. Height 4: root 16 has left child 10 with left child 5 with left child 4 with left child 1 and 16 has right child 17 with right child 17. Height 5 would be similar with 17 as root.

3. What is the difference between the binary-search property and the heap property? (*) Can the heap property be used to print out the keys of an \( n \)-node tree in sorted order in \( O(n) \) time? Explain how or why not.

Solution: Actually there is not that much the same. In min-heap the parent is less than both children. In BST the parent is bigger than the leftchild and less than the right child and moreover the parent is bigger than all of the left subtree and less than all of the right subtree.

There is no way to get a sorted order from a min-heap in time \( O(n) \). Recall you can build a min-heap from scratch in time \( O(n) \). So is you could then get a sorted order in further time \( O(n) \) then in total \( O(n)+O(n) = O(n) \) you would sort an array from scratch. We can’t do that – sorts take time \( O(n \lg n) \) unless there is additional information about the data.

4. What would the BST tree look like if you start with the root \( a_1 \) with key[\( a_1 \)] = 1 (and nothing else) and then you apply INSERT[\( a_2 \)], . . . , INSERT[\( a_n \)] in that order where key[\( a_i \)] = \( i \) for each \( 2 \leq i \leq n \)? Suppose the same assumptions of starting with \( a_1 \) and the key values but the INSERT commands were done in reverse order INSERT[\( a_n \)], . . . , INSERT[\( a_2 \)].

Solution: In the first case it would be like a line to the right, with each \( a_i \) the right child of the previous \( a_{i-1} \). In the second case \( a_n \) would be the right child of \( a_1 \) and then it would look like a line to the left (but always staying to the right of \( a_1 \)) with \( a_{i-1} \) being the left child of \( a_i \).

5. Let \( A[1 \cdots N] \) be an array of numbers. Consider the following program: (* means times, Assume \( N \geq 2 \) and that the \( A[I] \) are distinct to avoid trivialities.)

KAUSHIK[A]
1. BUILD-MAX-HEAP[A]
2. \( I \leftarrow 1 \)
3. WHILE \( I*I \leq N \)
4. EXTRACT-MAX[A]
5. \( I++ \)
6. END WHILE

(a) How long does step 4 take as a function of \( N \) and \( I \).

Solution: When you’ve reached \( I \) you have already removed \( I - 1 \) items from the heap so the heap now has size \( N - I + 1 \) and so the time is \( O(\ln(N - I + 1)) \).

(b) What is the total time for the WHILE loop.

Solution: The WHILE loop goes from \( I = 1 \) to \( I = \lfloor \sqrt{N} \rfloor \) so the total time for the WHILE loop is

\[
\sum_{I=1}^{\lfloor \sqrt{N} \rfloor} O(\ln(N - I))
\]

For this sum, however, all the terms are asymptotic to each other as the smallest term \( \ln(N - \sqrt{N}) \) is asymptotic to \( \ln(N) \). Thus the sum is the number of terms \( \sqrt{N} \) times the first term so the total time for the WHILE loop is \( O(\sqrt{N} \ln N) \).

(c) What is the total time for KAUSHIK[A].

Solution: It is \( O(N) \) as the time for step 1, BUILD-MAX-Heap dominates.

(d) Describe the value \( z \) returned in terms of the values of the array \( A \).

Solution: We removed the max \( \lfloor \sqrt{N} \rfloor \) times and then returned the max. So the \( z \) returned is the \( \lfloor \sqrt{N} \rfloor + 1 \)-st largest value.