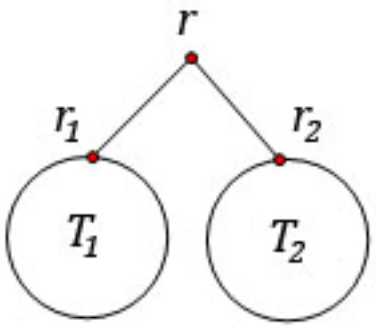
Math 15 – Discrete Structures – Homework 20 Solutions – 5.3

5.3#14: Consider the following way of sorting an array *X* with *n* elements.

```
Put the elements of X into a binary search tree.
Do an InOrder traversal of your tree.
```

(a) Approximate the average-case complexity of this algorithm. (For the average case, assume that you get a balanced binary search tree. Don't bother with the definition of average case complexity.)

SOLN: Recall the definition of a *binary search tree*:
Let *S* be a set that is totally ordered by $\leq$. Then a binary search tree on *S* is
B_1. The empty tree, or
B_2. A single vertex, $r \in S$. In this case *r* is the root of the tree.
R. If $\mathcal{T}_1$ and $\mathcal{T}_2$ are binary search trees with roots r_1 and r_2, respectively, and if $a \leq r$ for all nodes $a \in \mathcal{T}_1$ and $r \leq b$ for all nodes $b \in \mathcal{T}_2$, then the tree



Is a binary search tree with root *r*.

Recall the `InOrder(`$\mathcal{T}$`)` traversal function, starting with the two base cases:
B_1. If $\mathcal{T}$ is the empty tree, `InOrder(`$\mathcal{T}$`)` = "" (the empty listing).
B_2. If $\mathcal{T}$ is the single node *r*, then `InOrder(`$\mathcal{T}$`)` = "*r*".
R. If $\mathcal{T}$ has root *r* and subtrees $\mathcal{T}_1$ and $\mathcal{T}_2$, then

`InOrder(`$\mathcal{T}$`)` = "`InOrder(`$\mathcal{T}_1$`)`,*r*,`InOrder(`$\mathcal{T}_2$`)`"

where the commas are part of the listing unless $\mathcal{T}_1$ or $\mathcal{T}_2$ is empty.
===
Evidently, the `InOrder()` traversal will visit every node exactly once, so it has complexity $\Theta(n)$. To place the *n* elements of *X* in the tree will require $c_1 + c_2 + \cdots + c_n$ comparisons, where c_i is the number of edges between the root and the position where x_i ends up – this is also the depth of the node. Suppose the tree ends up balanced then there were be no comparison for the root node, one comparison for the 2 nodes at depth 1, 2 comparisons for the 4 nodes at depth 2, and so on. So for a balanced tree of depth *p*, the number of comparisons is $C = \sum_{i=0}^{p} 2^i \cdot i = 1 \cdot 0 + 2 \cdot 1 + 4 \cdot 2 + 8 \cdot 3 + \cdots + 2^p \cdot p$. A full binary tree

with depth p has $\sum_{i=0}^{p} 2^i = 2^{p+1} - 1$ nodes so we can take this as kind of a worst case, and assume $n \approx 2^{p+1} \Leftrightarrow p \approx p + 1 \approx \log_2 n$ so the number of comparisons goes like

$C \approx 1 \cdot 0 + 2 \cdot 1 + 4 \cdot 2 + 8 \cdot 3 + \cdots + \frac{n}{2} \cdot \log_2 n > \frac{n}{2} \cdot \log_2 n.$

Also, $C \approx 1 \cdot 0 + 2 \cdot 1 + 4 \cdot 2 + 8 \cdot 3 + \cdots + \frac{n}{2} \cdot \log_2 n$

$$\leq 1 \cdot \log_2 n + 2 \cdot \log_2 n + 4 \cdot \log_2 n + 8 \cdot \log_2 n + \cdots + \frac{n}{2} \cdot \log_2 n$$

$$= \left(1 + 2 + 4 + 8 + \cdots + \frac{n}{2}\right) \log_2 n \leq n \cdot \log_2 n$$

So that $\frac{n}{2} \log_2 n \leq C \leq n \cdot \log_2 n$ which shows that $C \in \Theta(n \cdot \log_2 n)$. Putting these together then we see the average-case complexity of the sorting algorithm is approximately $\Theta(n + n \cdot \log_2 n) = \Theta(n \cdot \log_2 n)$.

(b) Compute the worst case complexity for this algorithm. (Hint: The worst case occurs when the array is in order to begin with!)

SOLN: If the array is in order to begin with then the binary tree becomes a root with only left children all the way down…no branches – sort of the opposite of a full binary tree. This means that the depth is $n - 1$ so the first part of the sorting algorithm requires $\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$ comparisons and the worst case complexity is thus $\Theta(n^2)$.

5.3#16: In this problem we will approximate the complexity of Prim's algorithm for finding a minimal spanning tree of a network $\mathcal{N}$:

```
Preconditions: 𝒩 is a connected network
Postconditions: 𝒯 is a minimal spanning tree
𝒯 ← v_i e_1 v_j, where e_1 is the shortest edge of 𝒩.
while 𝒯 does not contain all of 𝒩's vertices
    e ← the shortest edge between a vertex in 𝒯 and a vertex not in 𝒯.
    Add edge e and the new vertex to 𝒯.
```

Let the input size n be the number of vertices in $\mathcal{N}$.

(a) Suppose that no vertex has degree greater than 5. Use Euler's theorem (the sum of the degrees of the vertices of G equals twice the number of edges) why the number of edges in the minimal spanning tree is at most a linear function of n.

SOLN: Suppose G has $|E|$ edges. Euler's theorem says the sum of the degrees of the vertices is $2|E|$ so $2|E| \leq 5n \Leftrightarrow |E| \leq \frac{5}{2}n$, so $|E| \in \mathcal{O}(n)$.

(b) Approximate the worst-case number of comparisons needed for this part of Prim's algorithm:

```
e ← the shortest edge between a vertex in 𝒯 and a vertex not in 𝒯.
```

SOLN: The algorithm to find the max of a set of n elements is $\mathcal{O}(n)$. Since there are $\mathcal{O}(n)$ edges, the worst case number of comparisons for this line is $\mathcal{O}(n)$.

(c) Give an approximate big-$\mathcal{O}$ estimate for the worst-case complexity of Prim's algorithm.

SOLN: The algorithm calls the line above at most once for each vertex, so the worst-case complexity is $\mathcal{O}(n^2)$.

5.3#22: Find the best-, worst-, and average-case values for rolling two standard six-sided dice.

SOLN: Best = 2, worst = 12, average = $6 \cdot \frac{7}{36} + \sum_{k=1}^{5} \frac{14k}{36} = 7$