MATH 163/COEN 179 Assignment 2
Due Date: Friday May 14

1. Implement a recursive quicksort with a cutoff to insertion sort for subfiles less than M elements, and experiment to determine the value of M for which the program runs fastest of a random file of 50000 floats.

The essential change to the code is rather than testing if (right > left), test if(right >= left + M - 1) If true, continue with the quicksort code, including the two recursive calls. If false, switch to insertion sort, with no further recursive calls.

You can modify the recurrence we found for the expected number of comparisons to find the value of M which minimizes this expectation. It turns out to be 10, no matter the size of the array. The case of swaps is more difficult. The best cutoff value in practice would depend on the relative speeds of comparisons and swaps, as well.

2. In class, we solved the mergesort recurrence

$$T(n) = T\left(\left\lfloor \frac{n}{2}\right\rfloor\right) + T\left(\left\lceil \frac{n}{2}\right\rceil\right) + n; \quad T(1) = 0$$

with the master theorem to show that $$T(n) = \Theta(n \log n)$$.

a) Prove that if n is a power of 2, then $$T(n) = n \log n$$ exactly.

b) Compute the first 40 or so values of $$T(n)$$. Do you detect a pattern in the values of $$T(n)$$ when n is not a power of 2? If not, generate a few more values until you do. Extra special brownie points if you can prove the pattern.

Let $$n = 2^k$$. Then $$T(2^k) = 2T(2^{k-1}) + 2^k$$, or equivalently

$$\frac{T(2^k)}{2^k} - \frac{T(2^{k-1})}{2^{k-1}} = 1$$

But then

$$\frac{T(2^k)}{2^k} - \frac{T(1)}{1} = \sum_{i=1}^{k} \frac{T(2^i)}{2^i} - \frac{T(2^{i-1})}{2^{i-1}} = \sum_{i=1}^{k} 1 = k$$

Since $$T(1) = 0$$. We see that $$T(2^k) = k2^k$$, or equivalently $$T(n) = n \log n$$.

You could also prove this by induction.

The general pattern you should find is that $$T(n)$$ increases by the same amount as $$n$$ increases between consecutive powers of 2. Another way of saying the same thing is
that, if \(2^k \leq n < 2^{k+1}\), (or equivalently, if \(k = \lfloor \log n \rfloor\)), then \((2^k, k2^k), (n, T(n))\), and \((2^{k+1}, (k + 1)2^{k+1})\) are collinear. Since the equation of the line through the first and last points is \(y = (k + 2)x - 2^{k+1}\), we see that \(T(n) = (k + 2)n - 2^{k+1}\). This statement can be proved by strong induction.

3. What is the average behavior for the number of steps required to build a heap from a random array of \(n\) entries? Consider both the top-down and bottom-up cases. (Hint: one case should be easy.)

The worst case for bottom-up is \(\Theta(n)\) steps, whereas processing the array would take at best \(n\) steps (each entry must be looked at). It follows that the average case would also require \(n\) steps. The average-case behavior for top-down is a lot harder problem. In fact, I don’t have a solution yet. I’ll keep you posted.

4. a) Determine all possible randomly built binary trees with 2, 3, or 4 nodes. (You may wish to consult p. 271 to make sure you found them all.)

b) Indicate which orderings of 12, 123, and 1234 go with each tree. (Trees may have more than one ordering associated to them.)

c) Calculate the expected depth of the node with key 1, for trees with 2, 3, and 4 nodes. Assume each ordering is equally likely. (Note: This is different than assuming each tree is equally likely.)

The expectations are \(\frac{1}{2}, \frac{5}{6}(= \frac{1}{2} + \frac{1}{3}), \frac{11}{12}(= \frac{1}{2} + \frac{1}{3} + \frac{1}{4})\) respectively

d) Show that for a randomly built binary tree of \(n\) nodes, the expected depth of the node with the smallest key is

\[
\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n}
\]

Let \(X\) be the random variable which gives the depth of 1 in a random binary tree built from some ordering of 1, 2, \ldots, \(n\). Then

\[
X = X_2 + X_3 + \cdots + X_n,
\]

where \(X_i\) represents the random variable that takes the value 1 if the node \(i\) is a direct ancestor of the node 1 and 0 otherwise. Basically, we are counting the direct ancestors in the randomly built tree to determine the depth. Now, the probability that \(i\) is a direct ancestor of 1, is the probability that in a random ordering that \(i\) precedes the nodes 1, 2, \ldots, \(i - 1\). (In this case, \(i\) is the smallest node found when it is added to the tree, so it always moves leftward, until it is added as the leftmost leaf.) The probability that \(i\) is the first node of its size or less to appear is \(1/i\), and thus

\[
E(X_i) = \frac{1}{i} \cdot 1 + \left(1 - \frac{1}{i}\right) \cdot 0 = \frac{1}{i}.
\]
Then we obtain

$$E(X) = E(X_2) + E(X_3) + \cdots E(X_n) = \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n}$$

You could also develop a recurrence relation for $E(X)$ and solve it.

5. Cormen et al., Problem 7-3
   a. Consider a transposition ($i < j$, but $A[i] > A[j]$). If $i$ and $j$ are in the first two-thirds, then line 6 swaps them. If $i$ and $j$ are in the last third, then line 7 swaps them. If $i$ is in the first two-thirds and $j$ is the the last third, then line 6 may move $A[i]$ to a new location in the first two-thirds but the entries will still be transposed. If $A[i]$ is moved to the middle third, then line 7 will swap $A[i]$ and $A[j]$. If $A[i]$ ends up in the first third after line 6, then $A[j]$, being smaller belongs in the first third of the array, and therefore will be moved by line 7 to the middle third of the array. (This is the key point). Then line 8 will effect the necessary swap.
   b. $T(n) = 3T(2n/3) + O(1)$.
   c. $T(n) = \Theta(n^{\log_3^2}) = \Theta(n^{2.709\ldots})$. This is a terrible sort.

6. Cormen et al., Exercise 8.3-2
   (Depends on implementation.)
   To stabilize any sort, one can add a field to each entry which contains the initial index.
   Then sorting lexicographically will break the tie according to initial index.

7. Cormen et al., Exercise 12.4-2
   Consider a complete binary tree of depth $k$ with $t$ nodes appended as a chain of one-child nodes (except, of course, for the last one). So $n = 2^{k+1} - 1 + t$. The height of the tree is $k + t$. To find the average depth for a node, we consider the sum of the depths of the nodes. The total in the complete binary tree part is $(k - 1)2^{k+1} + 2$ and the total for the tail is

   $$k + 1 + k + 2 + \cdots + k + t = tk + \frac{t(t + 1)}{2}.$$ 

   The average depth is thus

   $$\frac{(k - 1)2^{k+1} + 2 + tk + \frac{t(t + 1)}{2}}{n}.$$
Since $k$ is $O(\lg n)$, we want $t$ to be $\omega(\lg n)$ and thus $t = \omega(k)$. It follows that the average depth is

$$\frac{O(n \lg n) + \Theta(t^2)}{n} = O(\lg n) + \Theta(t^2/n).$$

If this is to be $\Theta(\lg n)$, clearly $t = o(n)$ which means $(k - 1)2^{k+1} = \Theta(\lg n)$ and the average depth is

$$\Theta(\lg n) + \Theta(t^2/n).$$

To keep this at $\Theta(\lg n)$ overall, we must have $t^2 = O(n \lg n)$ or $t = O(\sqrt{n \lg n})$. Any value of $t$ which is $\omega(\lg n)$ but $O(\sqrt{n \lg n})$ will work, and the desired upper bound is $O(\sqrt{n \lg n})$. 