1. (15) Let $A[1 \cdots N]$ be an array with all entries integers between 0 and $N - 1$. How long would \textsc{Radix-Sort} take to sort $A$ assuming that we use base 2 (that is, binary)? (Assume the entries $A[i]$ are already given as binary strings in the input.) You must give an argument for your answer. Give (no proofs required!) a faster way to sort this data.

Solution: There are $\lg n$ digits and each Counting Sort takes $O(n)$ so time is $O(n \lg n)$. Straight Counting Sort on the original data takes $O(n)$. 

2. (15) \textsc{Toom-3} is an algorithm similar to the Karatsuba algorithm discussed in class. (Don’t worry how \textsc{Toom-3} really works, we just want an analysis given the information below.) It multiplies two $n$ digit numbers by making five recursive calls to multiplication of two $n/3$ digit numbers plus thirty additions and subtractions. Each of the additions and subtractions take time $O(n)$. Give the recursion for the time $T(n)$ for \textsc{Toom-3} and use the Master Theorem to find the asymptotics of $T(n)$. Compare with the time $\Theta(n^{\log_{2}3})$ of Karatsuba. Which is faster when $n$ is large?

Solution: See assignment.

3. (20) [See Solutions to hw1] Let $A$ be an array of length 127 in which the values are distinct and in increasing order.

(a) In the procedure \textsc{Build-Max-Heap(A)} precisely how many times will two elements of the array be exchanged? (Reason, please!)

(b) Now suppose the values are distinct and in decreasing order. Again, in the procedure \textsc{Build-Max-Heap(A)} precisely how many times will two elements of the array be exchanged? (Reason, please!)
4. (20) Give an algorithm TINYPIECES that does the following. As input you have an array \( \text{PRICE}[1 \cdots N] \) where, for \( 1 \leq i \leq N \), \( \text{PRICE}[i] \) is the price of a rod of length \( i \). You are given a rod of total length \( N^5 \). You wish to cut it into pieces (but all pieces must be of length at most \( N \)) so as to maximize the total price. Your algorithm should output \( \text{VALUE} \), where this represents the maximal total price. (Note: You are not being asked to find the actual cutting of the rod.) Analyze (in \( \Theta \)-land) the total time your algorithm takes. You must give a description in clear words of what the algorithm is doing.

**Solution:**
Create \( R[0 \cdots N^5] \), \( R[J] \) being the revenue (maximal total price) of a rod of length \( J \). Initialize \( R[0] = 0, R[1] = \text{PRICE}[1] \).

FOR \( J = 2 \) TO \( N^5 \)
\( R[J] = 0 \) (*initialization, will change*)
For \( I = 1 \) to \( \text{max}[J,N] \) (*try first cut at \( I \) *)
\( R[J] = \max[R[J], \text{PRICE}[I] + R[I - J]] \)
END FOR (*so now \( R[J] = \max_I (\text{PRICE}[I] + R[I - J]) \)*)
END FOR

\( \text{VALUE} \leftarrow R[N^5] \)
RETURN \( \text{VALUE} \)

Key point: The inner \( I \)-loop only has at most \( N \) values (reflecting that the first cut must be in a position \( \leq N \)) and so takes \( O(N) \), the outer loop has \( O(N^5) \) so the total time is \( O(N^6) \).

**Comment:** A number of students found the solution for a rod of length \( N \) and multiplied by \( N^4 \). Clever, but wrong. For example, \( N = 10 \), all prices are zero except \( \text{PRICE}[3] = 1 \). So a rod of length 10 gets 3. But a rod of length \( 10^5 \) gets 33333 as, roughly, the leftovers get put together.

5. (20) Describe the algorithm \( \text{QUICKSORT}(p,r) \) which sorts the elements \( A[i], p \leq i \leq r \). (You can assume \( p \leq r \).) You may, and should, use auxilliary arrays. Subroutines must be described in full. Explain in clear words what the algorithm is doing. Give (without proof!) both the average and the worst-case time for \( \text{QUICKSORT}(1,n) \).

**Solution:** See text or notes.

6. (15) Here is a psuedocode sorting algorithm that uses Binary Search Tree. We wish to sort \( A[1 \cdots N] \). (There are no records here, each \( A[I] \) is itself the key.) Begin with an empty BST \( T \).

Part I: FOR \( I = 1 \) to \( N \); INSER T \( A[I] \) into \( T \); ENDFOR
Part II: Apply IN-ORDER-TREE-WALK to T
Analyse both the average time and the worst case time for this algorithm.

Solution: Average time to insert into a tree with \( i - 1 \) elements is \( O(\lg i) \) so average time for Part I is \( O(\sum_{i=1}^{n} \lg i) = O(n \lg n) \). Worst time is with a path so insertion takes \( O(i) \) so Part I is \( O(\sum_{i=1}^{n} i) = O(n^2) \). Part II takes \( O(n) \) always. So total average time is \( O(n \lg n) + O(n) = O(n \lg n) \) while worst total time is \( O(n^2) + O(n) = O(n^2) \).

Comment: Some students wrote that the FOR loop, going from 1 to \( N \), takes time \( O(N) \). This a serious misconception! You need calculate how much time the inside (here INSERT) takes as a function of \( I \) and then the total time is the sum from \( I = 1 \) to \( I = N \) of these times.