AN OLD MIDTERM SOLUTIONS

1. (15) Let $A[1 \cdots N]$ be an array with all entries integers between 0 and $N-1$. How long would `RADIX-SORT` take to sort A assuming that we use base 2 (that is, binary)? (Assume the enries $A[I]$ 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) `Frank` is an algorithm similar to the Karatsuba algorithm discussed in class. It multiplies two n digit numbers by making five recursive calls to multiplication of two $n/4$ digit numbers plus forty additions and subtractions. Each of the additions and subtractions take time $O(n)$. Give the recursion for the time $T(n)$ for `Frank` 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? (If you don't have a calculator handy, tell what quantities you need compare to tell which is faster.)
   `Solution:`$T(n) = 5T(n/4)+40 \cdot O(n) = 5T(n/4)+O(n)$. As $\log_4 5 > 1$ this is Low Overhead and so $T(n) = \Theta(n^c)$ where $c = \log_4 5$. As $\log_4 5 \sim 1.16 < \log_2 3 \sim 1.58$, `Frank` is better. (BTW, `Frank` is fictitious.)

3. (20) Let A be an array of length 1023 in which the values are distinct and in increasing order.

   (a) In the procedure `BUILD-MAX-HEAP(A)` *precisely* how many times will two elements of the array be exchanged? (Reason, please!)
   `Solution:`BUILD-MAX-HEAP(A) starts from I = LENGTH(A)/2 DOWN to 1, every I will do Max-Heapify.
   For $256 \leq I \leq 511$,there should be one exchange.
   For $128 \leq I \leq 255$, there should be 2 exchanges.
   For $64 \leq I \leq 127$, there should be 3 exchanges.
   For $32 \leq I \leq 63$, there should be 4 exchanges.
   For $16 \leq I \leq 31$, there should be 5 exchanges.
   For $8 \leq I \leq 15$, there should be 6 exchanges.
   For $4 \leq I \leq 7$, there should be 7 exchanges.

For $2 \leq I \leq 3$, there should be 8 exchanges.
For $1 \leq I \leq 1$, (the root!) there should be 9 exchanges.
Total: $1(9)+2(8)+4(7)+8(6)+16(5)+32(4)+64(3)+128(2)+256(1) = 1013$ exchanges. (Mathgeeks may find a precise formula when the length is $2^t - 1$.)

(b) Now suppose the values are distinct and in decreasing order. Again, in the procedure `BUILD-MAX-HEAP(A)` *precisely* how many times will two elements of the array be exchanged? (Reason, please!)
`Solution:`Never! Each element will be placed originally in precisely its correct final spot.

4. (20) Consider the recursion

$$T(3n) = 9T(n) + 4n^2$$

with initial value $T(1) = 5$.

(a) (5) Find $T(9)$ *precisely.*
`Solution:`$T(3) = 9T(1) = 4 \cdot 1^2 = 9 \cdot 5 + 4 = 49$
$T(9) = 9T(3) + 4 \cdot 3^2 = 9 \cdot 49 + 36 = 477$

(b) (5) Use the Master Theorem to give the asymptotics of $T(n)$ in Theta-land. (Brief explanation, please.)
`Solution:`As $\log_3 9 = 2$ this is Just Right Overhead so $T(n) = \Theta(n^2 \lg n)$

(c) (10) Using a suitable auxilliary variable find a *precise* formula for $T(n)$ where n is a power of 3. (Write $n = 3^t$. Your formula can use n and/or t.)
`Solution:`Set $S(n) = T(n)/n^2$. We rewrite (there are other ways) the recursion as: $T(n) = 9T(n/3) + (4/9)n^2$. Dividing by n^2 gives the LHS as $T(n)/n^2 = S(n)$ and on the right we have $9T(n/3)/n^2 = S(n/3)$ so $S(n) = S(n/3) + (4/9)$. Also $S(1) = T(1)/1^2 = 5$. When $n = 3^t$ we are tripling t times (starting at 1) so that $S(n) = 5 + (4/9)t$. Going back to the original equation

$$T(n) = n^2 S(n) = 5n^2 + n^2(4/9)t$$

or, to remove t

$$T(n) = n^2 S(n) = 5n^2 + n^2(4/9)\log_3 n$$

(This was the toughest question on the exam!)

5. (20) Give an algorithm `TINYPIECES` that does the following. As *input* you have an array $PRICE[1 \cdots N]$ where, for $1 \leq i \leq N$, $PRICE[i]$ 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 $VALUE$, where this represents the maximal total price. (Note: You are not being asked to find the actual cutting of the rod.) Analyze (in Θ-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 $R[0] = 0, R[1] = PRICE[1]$.
FOR $J = 2$ TO N^5
$R[J] = 0$ (*initialization, will change*)
For $I = 1$ to $\min[J, N]$ (*try first cut at I *)
$R[J] = \max[R[J], P[I] + R[J - I]]$
END FOR (*so now $R[J] = \max_I(P[I] + R[I - J])$*)
END FOR
$VALUE \leftarrow R[N^5]$
RETURN $VALUE$

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 $O(N^6)$.

6. (20) Describe the algorithm $\texttt{QUICKSORT}(p,r)$ 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 $\texttt{QUICKSORT}(1,n)$.
`Solution:` See text or notes.

7. (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; INSERT $A[I]$ into T; ENDFOR
Part II: Apply IN-ORDER-TREE-WALK to T
Analyze *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 $FORI$ loop, going from 1 to N, takes average time $O(\lg N)$ for each I and thus $O(N \lg N)$. Turns out not to matter for this problem but for other problems it can lead to the wrong answer. 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. (If everything else was good only a few points were deducted but please be careful about this in the future!)