CS 577 Homework 1

Alex Pizzuto

Question One

We seek to arrange the following functions in descending order of growth rate: n^2, $2^{4\log n}$, $n^{2\log n}$, $n(\log n)^5$, 3^n. First, we cite our final answer, with analysis to follow. In the large n regime,

$$3^n \ > n^{2\log n} \ > 2^{4\log n} \ > n^2 \ > n(\log n)^5$$

We will prove this by showing that each function in this chain is bounded by below by the following function. First, we show that $3^n \ > n^{2\log n}$. We do this by taking the logarithm of both sides.

$$\log 3^n = n\log 3 \ \text{ and } \ \log\left(n^{2\log n}\right) = 2\log n\log n$$

The function on the left is clearly polynomial while the one on the right is logarithmic, thus $3^n > n^{2\log n}$. Next, to show $n^{2\log n} \ > 2^{4\log n}$,

$$\begin{aligned} 2^{4\log n} &= 2^{\log n}2^{\log n}2^{\log n}2^{\log n} \\ &= n^{\log 2}n^{\log 2}n^{\log 2}n^{\log 2} \\ &= n^{4\log 2} \end{aligned}$$

which is polynomial, and clearly less than $n^{2\log n}$ in the large n regime as $n^{2\log n}$ is exponential. Now, proving that $2^{4\log n} \ > n^2$, we know that $2^{4\log n} = n^{4\log 2}$. Then, clearly, $2^{4\log n} \ > n^2$ as $n^{4\log 2} > n^2$ (assuming that we are working in base 10), because $\log 2 < 0.68$ thus $4\log 2 > 2$, so the exponent of $n^{4\log 2}$ is larger than that of n^2. Finally, we show that $n^2 \ > n(\log n)^5$. We do this by taking the ratio of the two functions and prove that $\frac{n^2}{n(\log n)^5} > 1$.

$$\frac{n^2}{n(\log n)^5} = \frac{n}{(\log n)^5}$$

The numerator is a polynomial, and the denominator is logarithmic, thus in the large n regime, the fraction is greater than 1, and $n^2 \ > n(\log n)^5$. As we have proven that each member is greater than its successor in this inequality chain, we have proven we have sorted in the correct order.

Question Two

Now, we derive the upper bounds for the following worst-case running times of certain algorithms:

a

$$
\begin{aligned}
T(n) &= 2T(n/2) + c \\
&= 2(2T(n/4) + c) + c \\
&= 4T(n/4) + (1+2)c \\
&= 4(2T(n/8) + c) + (1+2)c \\
&= 8T(n/8) + (1+2+4)c \\
&\vdots \\
&= 2^k T(n/2^k) + c\sum_{i=0}^{k-1} 2^i \qquad \text{where} \quad \frac{n}{2^k} = 1 \\
&= nT(1) + c(2^k - 1) \\
&= nT(1) + c(n-1) \\
&\Rightarrow T(n) = \mathcal{O}(n)
\end{aligned}
$$

b

$$
\begin{aligned}
T(n) &= 8T(n/2) + c \\
&= 8(8T(n/4) + c) + c \\
&= 8^2 T(n/2^2) + (1+8)c \\
&= 8^2(8T(n/2^3) + c) + (1+8)c \\
&= 8^3 T(n/2^3) + (1+8+8^2)c \\
&\vdots \\
&= 8^k T(n/2^k) + c\sum_{i=0}^{k-1} 8^i \qquad \text{where} \quad \frac{n}{2^k} = 1 \\
&< n^3 T(1) + c \cdot 8^k = n^3(T(1) + c) \\
&\Rightarrow T(n) = \mathcal{O}(n^3)
\end{aligned}
$$

Question Three (5.1)

Our goal is to find the median of a database of size $2n$ that is broken into 2 sorted databases each of size n, with as few queries as possible (we work in the idealized situation where the database sizes are powers of 2, but a similar analysis would hold if we were to rigorously include floors and ceilings). First, some definitions. We call one database X, and the other Y (then $X(i)$ is the i^{th} smallest element in X).

In order to accomplish this, we employ a divide and conquer approach. First, perform a query of $X(n/2)$ and $Y(n/2)$, and call the results x and y respectively. Then, if $x < y$, then the first half of database X is smaller than $X(k)$ (as X is sorted), and it is smaller than the first half of database Y (as $x < y$ and Y is sorted). Thus, at least half of the combined database (the second half of both databases) is larger than the first half of X, thus x is less than the median, and we can discard the first half of X. Similarly, y is greater than the first half of Y as well as the first half of X, so it is greater than the median, so we can discard the second half of Y. As we have eliminated the

same number of elements less than the median as we have greater than the median, the median of the new combined database is still the median of the new database, so we then proceed recursively. Thus, we have the following function, median(n, a, b, c, d), where we would call at first median(n, 1, n, 1, n) (assuming the not so conventional indexing beginning at 1):

```
median(n, a, b, c, d) {
if n=1 then
  return min(X(n), Y(n))
end
k = n/2;
if X((a + b)/2) < Y((c + d)/2) then
  return median(k, (a+b)/2, b, c, (c+d)/2);
else
  return median(k, a, (a+b)/2, (c+d)/2, d);
end
}
```

Algorithm 1: Algorithm to find Median of 2 sorted databases

where a, b, c, d refer to the starting and ending indices of the databases, ie looking at elements of X from a to b and elements of Y from c to d. We are now traversing the databases in much the same way that you would a binary tree, and it is clear by the fact that we are dividing the databases size by 2 in each recursive call that we will only have to perform $\mathcal{O}(\log n)$ queries. As a brief derivation of this, we note that we can write our running time as

$$\begin{aligned} T(n) &= T(n/2) + 2 \\ &= (T(n/4) + 2) + 2 \\ &\vdots \\ &= T(n/2^k) + 2k \\ &= T(1) + 2\log_2 n \\ &\Rightarrow \mathcal{O}(\log n) \end{aligned}$$