Questions

For the questions below, solve the recurrence assuming n is a power of 2. For Q1-Q4, assume $t(1) = 1$.

1. $$t(n) = t(\frac{n}{2}) + n + 2.$$

   This is similar to binary search, but now we have to do n operations during each call.

   What do you predict? Is this $O(\log_2 n)$ or $O(n)$ or $O(n^2)$ or what?

2. $$t(n) = t(\frac{n}{2}) + \frac{n}{2} + 2$$

   Compare with the previous question. What is the effect of having an $\frac{n}{2}$ term instead of n ?

3. $$t(n) = 2t(\frac{n}{2}) + n^2$$

   This similar to mergesort except we need to do n^2 operations at each call, instead of n.

4. $$t(n) = 3\ t(\frac{n}{2}) + cn$$

   This recurrence arises in an algorithm for fast multiplication of two n digit numbers, which is faster than the grade school algorithm. The method is called Karatsuba multiplication. I mentioned it earlier in the course, and you may see it again in COMP 251.
   See `http://www.cim.mcgill.ca/~langer/250/fastmultiplication.pdf` if you are interested in the details.

Answers

1. Here we are cutting the problem in half, like in a binary search, but we need to do n operations to do so. This term will give us an $n + \frac{n}{2} + \frac{n}{4} + \ldots 1 = 2n - 1$ effect. The constant "2" will give us a $\log n$ effect since it has to be done in each recursive call. Formally, we have:

$$
\begin{aligned}
t(n) &= t(\frac{n}{2}) + n + 2 \\
&= [t(\frac{n}{4}) + \frac{n}{2} + 2] + n + 2 \\
&= [t(\frac{n}{8}) + \frac{n}{4} + 2] + \frac{n}{2} + 2 + n + 2 \\
&= t(\frac{n}{2^k}) + \frac{n}{2^{k-1}} ... + \frac{n}{2} + n + 2k \\
&= t(1) + 2 + ... + \frac{n}{2} + n + 2\log(n) \\
&= 1 + \sum_{i=1}^{\log n} 2^i + 2\log(n) \\
&= \sum_{i=0}^{\log n} 2^i + 2\log(n), , \quad \text{see geometric series formula below} \\
&= (2^{\log n + 1} - 1)/(2 - 1) + 2\log(n) \\
&= (2^{\log n} \cdot 2 - 1)/(2 - 1) + 2\log(n) \\
&= 2n - 1 + 2\log(n)
\end{aligned}
$$

This is $O(n)$ because the largest term that depends on n is the "$2n$" term.

The formula for the geometric series is :

$$\sum_{i=0}^{N-1} x^i = \frac{x^N - 1}{x - 1}$$

Here, I am using $x = 2, N = \log_2 n$. The general formula is derived in lecture 2 on page 5.

2. DOUBLECHECK THIS BELOW.

This is basically the same as the previous problem except that now we have to do half as much work ($\frac{n}{2}$) instead of n at each "call". Will this give us sub-linear behavior? No, it won't since even at the first call we have a term $\frac{n}{2}$.

$$
\begin{aligned}
t(n) &= t(\frac{n}{2}) + \frac{n}{2} + 2 \\
&= (t(\frac{n}{4}) + \frac{n}{4} + 2) + \frac{n}{2} + 2 \\
&= (t(\frac{n}{8}) + \frac{n}{8} + 2) + \frac{n}{4} + 2) + \frac{n}{2} + 2 \\
&= (t(\frac{n}{n}) + \frac{n}{n} + 2) + \cdots + \frac{n}{8} + 2 + \frac{n}{4} + 2 + \frac{n}{2} + 2 \\
&= t(1) + 1 + 2 + 4 + 8 + \cdots + \frac{n}{2} + 2\log n \\
&= t(1) + \sum_{i=0}^{\log \frac{n}{2}} 2^i + 2\log(n) \\
&= t(1) + (2^{\log n} - 1)/(2-1) + 2\log(n) \\
&= n + 2\log n
\end{aligned}
$$

This is $O(n)$.

3. The first term of the recurrence is similar to mergesort, but the second term is different since it is now quadratic rather than linear in n. What is the effect? Again, we let $n = 2^k$ and $t(1) = 1$.

$$
\begin{aligned}
t(n) &= 2t(\frac{n}{2}) + n^2 \\
&= 2[2t(\frac{n}{2^2}) \ + (\frac{n}{2})^2] + n^2 \\
&= 2^2 t(\frac{n}{2^2}) \ + \frac{n^2}{2} + n^2 \\
&= 2^2[2t(\frac{n}{2^3}) + (\frac{n}{2^2})^2] + \frac{n^2}{2} + n^2 \\
&= 2^3 t(\frac{n}{2^3}) + \frac{n^2}{4} + \frac{n^2}{2} + n^2 \\
&= 2^k t(\frac{n}{2^k}) + \frac{n^2}{2^{k-1}} + \frac{n^2}{2^{k-2}} + \ldots + \frac{n^2}{2} + n^2 \\
&= n\ t(1) + n^2 \sum_{i=0}^{\log(n)-1} \frac{1}{2^i} \\
&= n + n^2(1 - (\frac{1}{2})^{\log n})/(1 - \frac{1}{2}) \\
&= n + 2n^2(1 - \frac{1}{n}) \\
&= n + 2n^2 - 2n \\
&= 2n^2 - n
\end{aligned}
$$

Here it is somewhat surprising that the answer is $O(n^2)$. In eyeballing the given recurrence, you might have guessed that there would be a further dependence on $\log n$. But that is not what happens. The many small versions of the problem that exist with the recursive calls end up costing not much. The reason, roughly speaking, is that n^2 costs much more for larger problems than smaller problems.

4. Assume $n = 2^k$, i.e. n is a power of 2.

$$
\begin{aligned}
t(n) &= 3\ t(\frac{n}{2}) + cn \\
&= 3 \cdot [3\ t(\frac{n}{4}) + c\frac{n}{2}] + cn \\
&= 3^2\ t(\frac{n}{4}) + 3c\frac{n}{2} + cn \\
&= 3^2\ [3t(\frac{n}{8}) + \frac{cn}{4}] + cn\frac{3}{2} + cn \\
&= 3^3\ t(\frac{n}{8}) + cn(\frac{3}{2})^2 + 3c\frac{n}{2} + cn \\
&= 3^k\ t(\frac{n}{2^k}) + cn\ ((\frac{3}{2})^{k-1} + \cdots + (\frac{3}{2})^2 + \frac{3}{2} + 1) \\
&= 3^k\ t(1) + cn\ ((\frac{3}{2})^k - 1)/(\frac{3}{2} - 1) \\
&= 3^{\log_2 n}\ t(1) + 2cn\ ((\frac{3}{2})^{\log_2 n} - 1)
\end{aligned}
$$

Using the fact that (see properties of logarithms reviewed in lectures):

$$3^{\log_2 n} = n^{\log_2 3}$$

and so

$$(\frac{3}{2})^{\log_2 n} = \frac{n^{\log_2 3}}{2^{\log_2 n}} = n^{(\log_2 3)-1}.$$

Thus,

$$
\begin{aligned}
t(n) &= n^{\log_2 3}\ t(1) + 2cn \cdot n^{(\log_2 3 - 1)} - 2cn \\
&= n^{\log_2 3}\ t(1) + 2c \cdot n^{\log_2 3} - 2cn
\end{aligned}
$$

which is $O(n^{\log_2 3})$. Note that $n^{\log_2 3} > n$, so the dominant term is $n^{\log_2 3}$ and subtracting cn is negligible effect when n is large.