Questions

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

1. \[ t(n) = t\left(\frac{n}{2}\right) + n + 2 \]

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. \[ t(n) = t\left(\frac{n}{2}\right) + \frac{n}{2} + 2 \]

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

3. \[ t(n) = 2t\left(\frac{n}{2}\right) + n^2 \]

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

4. \[ t(n) = 3t\left(\frac{n}{2}\right) + cn \]

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](http://www.cim.mcgill.ca/~langer/250/fastmultiplication.pdf) if you are interested in the details.

5. When we considered the mergesort algorithm in class, we had a base case of \( n = 1 \). What if we had stopped the recursion at a larger base case? For example, suppose that when the list size has been reduced to 4 or less, we switch to running bubble sort instead of mergesort. Since bubble sort is \( O(n^2) \), one might ask whether this would increase the \( O(\ ) \). Would this modified mergesort become an \( O(n^2) \) algorithm? Solve the recurrence for mergesort by assuming a base case where \( t(n) = n^2 \) for the case of \( n \leq 4 \).

Note: while this might seem like a toy problem, it makes an important point. Sometimes when we write recursive methods, we find that the base case can be tricky to encode. If there is a slower method available that can be used for small instances of the problem, and this slower method is easy to encode, then use it!
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:

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

$$
= [t\left(\frac{n}{4}\right) + \frac{n}{2} + 2] + n + 2
$$

$$
= [t\left(\frac{n}{8}\right) + \frac{n}{4} + 2] + \frac{n}{2} + 2 + n + 2
$$

$$
= t\left(\frac{n}{2^k}\right) + \frac{n}{2^{k-1}} + \ldots + \frac{n}{2} + n + 2k
$$

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

$$
= 1 + \sum_{i=1}^{\log n} 2^i + 2 \log(n)
$$

$$
= \sum_{i=0}^{\log n} 2^i + 2 \log(n), \quad \text{see geometric series formula below}
$$

$$
= \left(2^{\log n+1} - 1\right)/(2 - 1) + 2 \log(n)
$$

$$
= (2^{\log n} \cdot 2 - 1)/(2 - 1) + 2 \log(n)
$$

$$
= 2n - 1 + 2 \log(n)
$$

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. This is basically the same as the previous problem except that now we have to do half as much work ($\frac{n^2}{2}$) 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 $\frac{n^2}{2}$.

\[
t(n) = t\left(\frac{n}{2}\right) + \frac{n}{2} + 2
\]

\[
= (t\left(\frac{n}{4}\right) + \frac{n}{4} + 2) + \frac{n}{2} + 2
\]

\[
= (t\left(\frac{n}{8}\right) + \frac{n}{8} + 2) + \frac{n}{4} + 2) + \frac{n}{2} + 2
\]

\[
= \frac{n}{n} + \frac{n}{n} + 2 + \cdots + \frac{n}{8} + 2 + \frac{n}{4} + 2 + \frac{n}{2} + 2
\]

\[
= t(1) + 1 + 2 + 4 + 8 + \cdots + \frac{n}{2} + 2 \log n
\]

\[
= t(1) + \sum_{i=0}^{\log \frac{n}{2}} 2^i + 2 \log(n)
\]

\[
= t(1) + (2^{\log n} - 1)/(2 - 1) + 2 \log(n)
\]

\[
= n + 2 \log n
\]

This is $O(n)$. 

last updated: 13th Oct, 2016 at 06:54
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 \( t(1) = 1 \).

\[
t(n) = 2t\left(\frac{n}{2}\right) + n^2
\]

\[
= 2[2t\left(\frac{n}{4}\right) + \left(\frac{n}{2}\right)^2] + n^2
\]

\[
= 2^2t\left(\frac{n}{2^2}\right) + \frac{n^2}{2} + n^2
\]

\[
= 2^2[2t\left(\frac{n}{4^2}\right) + \left(\frac{n}{2^2}\right)^2] + \frac{n^2}{2} + n^2
\]

\[
= 2^3t\left(\frac{n}{2^3}\right) + \frac{n^2}{4} + \frac{n^2}{2} + n^2
\]

\[
= 2^k t\left(\frac{n}{2^k}\right) + \frac{n^2}{2^{k-1}} + \frac{n^2}{2^{k-2}} + \ldots + \frac{n^2}{2} + n^2
\]

\[
= n \cdot t(1) + n^2 \sum_{i=0}^{\log(n)-1} \frac{1}{2^i}
\]

\[
= n + n^2\left(1 - \left(\frac{1}{2}\right)^{\log(n)}\right) / \left(1 - \frac{1}{2}\right)
\]

\[
= n + 2n^2(1 - \frac{1}{n})
\]

\[
= n + 2n^2 - 2n
\]

\[
= 2n^2 - n
\]

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.

\[
t(n) = 3 \cdot \left( \frac{n}{2} \right) + cn
\]

\[
= 3 \cdot \left[ 3 \cdot \frac{n}{4} + c \cdot \frac{n}{2} \right] + cn
\]

\[
= 3^2 \cdot \frac{n}{4} + 3c \cdot \frac{n}{2} + cn
\]

\[
= 3^2 \cdot \left( \frac{3}{4} t\left( \frac{n}{8} \right) \right) + cn \cdot \frac{3}{2} + cn
\]

\[
= 3^3 \cdot \frac{n}{8} + cn \cdot \left( \frac{3}{2} \right)^2 + 3c \cdot \frac{n}{2} + cn
\]

\[
= 3^k \cdot t\left( \frac{n}{2^k} \right) + cn \cdot \left( \frac{3}{2} \right)^{-k} + \cdots + \left( \frac{3}{2} \right)^2 + \frac{3}{2} + 1
\]

\[
= 3^k \cdot t(1) + cn \cdot \left( \frac{3}{2} \right)^{-k} - 1
\]

\[
= 3^{\log_2 n} \cdot t(1) + 2cn \cdot \left( \frac{3}{2} \right)^{\log_2 n} - 1
\]

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

\[
3^{\log_2 n} = n^{\log_2 3}
\]

and so

\[
\left( \frac{3}{2} \right)^{\log_2 n} = \frac{n^{\log_2 3}}{2^{\log_2 n}} = n^{(\log_2 3) - 1}.
\]

Thus,

\[
t(n) = n^{\log_2 3} \cdot t(1) + 2cn \cdot n^{(\log_2 3) - 1} - 2cn
\]

\[
= n^{\log_2 3} \cdot t(1) + 2c \cdot n^{\log_2 3} - 2cn
\]

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.
5. Assume \( n \) is a power of 2 (to simplify the argument). We would have:

\[
t(n) = 2t\left(\frac{n}{2}\right) + cn
\]

\[
= 2 \left( 2t\left(\frac{n}{4}\right) + c\frac{n}{2} \right) + cn
\]

\[
= 4t\left(\frac{n}{4}\right) + c(n + n)
\]

\[
= 8t\left(\frac{n}{8}\right) + c(n + n + n)
\]

\[
= 16 t\left(\frac{n}{16}\right) + c(n + n + n + n)
\]

\[
= 2^k t\left(\frac{n}{2^k}\right) + ckn
\]

We want to stop at \( t(4) \) on the right side. So we let \( k \) be such that \( \frac{n}{2^k} = 4 \), that is, \( 2^k = \frac{n}{4} \), which gives

\[
t(n) = \frac{n}{4} t(4) + cn \log_2 \frac{n}{4}
\]

Now the tricky part of this problem is that you may be thinking that you need to somehow put an \( n^2 \) dependence in at this point to account for the switch to insertion sort. But you don’t need to do this! Since we are only switching to insertion sort when \( n \leq 4 \), the term \( t(4) \) is a constant. This is the time it takes to solve a problem of size 4.

By inspection of the last line above, we see that the biggest term that depends on \( n \) is \( n \log n \) which is the same as mergesort. Thus, switching to insertion sort when the problem is reduced to size \( n = 4 \) does not change the dependence on \( n \).