Questions

1. For this and the following questions, solve the recurrence assuming n is a power of 2 and $t(1) = 1$.

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

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

   Before you solve it, what do you predict? Is this $O(\log_2 n)$ or $O(n)$ or $O(n^2)$ or what?

2. Solve

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

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

3. Solve

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

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

4. The Tower of Hanoi recurrences is

$$t(n) = c + 2t(n-1)$$

   and we saw in the lecture that the solution is

$$t(n) = c2^{n-1} + 2^{n-1}t(1).$$

   What do these two terms represent?

5. Solve

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

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

   BTW, before you solve it, think to yourself. What do you predict? Is this $O(\log_2 n)$ or $O(n)$ or $O(n^2)$ or what?

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 $x = 2, N = \log_2 n$.

2. 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 i.e. less than $O(n)$? 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 $t(1) = 1$.

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

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. Many small versions of the problem are generated with the recursive calls, but they end up costing about as much as the the n^2 cost at the first level of the recursion. The reason, roughly speaking, is that n^2 costs much more for larger problems than smaller problems.

4. The first term $c2^{n-1}$ is the time spent moving the disks that are *not* the smallest, and the term $2^{n-1}t(1)$ is the time spent moving the smallest of the n disks. This is the base case where $n = 1$ and there is no recursive call.

5.

$$
\begin{aligned}
t(n) &= n + t(\frac{n}{2}) \\
&= n + \frac{n}{2} + t(\frac{n}{4}) \\
&= n + \frac{n}{2} + \frac{n}{4} + t(\frac{n}{8}) \\
&= n + \frac{n}{2} + \frac{n}{4} + \dots \frac{n}{2^{k-1}} + t(\frac{n}{2^k}) \\
&= n + \frac{n}{2} + \frac{n}{4} + \cdots + 4 + 2 + t(1), \quad \text{when } 2^k = n \\
&= n + \frac{n}{2} + \frac{n}{4} + \cdots + 4 + 2 + 1 - 1 + t(1) \\
&= \sum_{i=0}^{\log_2 n} 2^i + t(1) - 1 \qquad\qquad \textit{see below} \\
&= (2^{(\log_2 n)+1} - 1)/(2-1) + t(1) - 1 \\
&= 2n - 1 \quad + \; t(1) - 1
\end{aligned}
$$

The geometric series $\sum_{i=0}^{\log_2 n} 2^i$ is just the same one you've seen many times in the course. The only difference now is that the range of i is expressed in a new way.

