COMP 775 - Advanced Design and Analysis of Algorithms Assignment 2

Jordan Sturtz

2-06-2023

Problem 4-1

Question:

Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$. Make your bounds as tight as possible, and justify your answers.

a. $T(N) = 2T(n/2) + n^4$

Answer: $\mathbf{\Theta(n^4)}$

Proof by Master Theorem

To use the master theorem case 3, first, we show that $f(n) = \Omega(n^{\log_b(a+\epsilon)})$ for some $\epsilon > 0$. Since $a = 2$, $b = 2$, we must show that $n^4 = \Omega(n^{\log_2(2+\epsilon)})$ for some $\epsilon > 0$. If we set $\epsilon = 2$, then by inspection we can see that $n^4 = \Omega(n^{\log_2(2+2)}) = \Omega(n^2)$. The last step is to prove that $af(n/b) \leq cf(n)$ for some constant $c < 1$. If we substitute a, b, and $f(n)$, we get

$$\begin{aligned}
af(n/b) &\leq cf(n) \\
2(n/2)^4 &\leq cn^4 \\
n^4/8 &\leq cn^4
\end{aligned}$$

This is obviously true for $c = 1/2$. Therefore by the master theorem, $T(N) = \Theta(f(n)) = \Theta(n^4)$

Justification by Unrolling / Substitution:

First, we unroll with different inputs:

$$\begin{aligned}
T(n) &= 2T(n/2) + n^4 \\
T(n/2) &= 2T(n/4) + n^4/4 \\
T(n/4) &= 2T(n/8) + n^4/8 \\
&\ldots
\end{aligned}$$

We substitute recursively to unroll $T(n)$:

$$\begin{aligned}
T(n) &= 4T(n/4) + n^4/2 + n^4 \\
&= 8T(n/8) + n^4/4 + n^4/2 + n^4 \\
&\ldots \\
&= 2^k T(n/2^k) + \sum_{i=0}^{k} n^4/2^i \\
&= 2^k T(n/2^k) + n^4 \sum_{i=0}^{k} 1/2^i
\end{aligned}$$

This recurrence holds for inputs $n > 2$, so we can set $n/2^k = 2$ to solve for k, which is $k = \log_2 n/2$. If we substitute this back into our previous result, we get:

$$
\begin{aligned}
T(n) &= 2^k T(n/2^k) + n^4 \sum_{i=0}^{k} 1/2^i \\
&= \frac{n}{2} T(2) + n^4 \sum_{i=0}^{k} 1/2^i \\
&= \frac{n}{2} C_1 + n^4 \sum_{i=0}^{k} 1/2^i
\end{aligned}
$$

It is given that $T(2)$ is constant, hence C_1. The last step is to show that $\sum_{i=0}^{k} 1/2^i$ is also constant. This infinite series converges to 2, as seen below:

$$
\begin{aligned}
S &= 1 + \frac{1}{2} + \frac{1}{4} ... \\
2S &= 2 + 1 + \frac{1}{2} + \frac{1}{4} ... \\
2S - S &= 2
\end{aligned}
$$

Since the infinite series converges to 2, our final expression is $T(n) = \frac{n}{2} C_1 + n^4 * C_2$, where C_2 approaches 2. Therefore, $T(n)$ is $\Theta(n^4)$.

b. $T(n) = T(7n/10) + n$

Answer: $\mathbf{\Theta(n)}$

Proof by Master Theorem

To use the master theorem case 3, first, we show that $f(n) = \Omega(n^{\log_b(a+\epsilon)})$ for some $\epsilon > 0$. Since $a = 1$, $b = 10/7$, and $f(n) = n$, we must show that $n = \Omega(n^{\log_{10/7}(1+\epsilon)})$ for some $\epsilon > 0$. To determine acceptable values for ϵ:

$$
\begin{aligned}
n^{\log_{10/7}(1+\epsilon)} &\geq n^1 \\
\log_{10/7}(1+\epsilon) &\geq 1 \\
\epsilon &\geq (10/7)^1 - 1 \\
\epsilon &\geq 3/7
\end{aligned}
$$

So, for any value $\epsilon \geq 3/7$, we see that $n = \Omega(n^{\log_2(1+\epsilon)})$. The last step is to prove that $af(n/b) \leq cf(n)$ for some constant $c < 1$. If we substitute a, b, and $f(n)$, we get

$$
\begin{aligned}
af(n/b) &\leq cf(n) \\
1(n/2) &\leq cn \\
1/2 &\leq c
\end{aligned}
$$

This is true for for $c \leq 1/2$. Therefore by the master theorem, $T(N) = \Theta(f(n)) = \Theta(n)$

Justification by Unrolling / Substitution

First, we unroll with different inputs:

$$\begin{aligned}
T(n) &= T(7n/10) + n \\
T(7n/10) &= T(7^2n/10^2) + 7n/10 \\
T(7^2n/10^2) &= T(7^3n/10^3) + 7^2n/10^2 \\
&...
\end{aligned}$$

We substitute recursively to unroll $T(n)$:

$$\begin{aligned}
T(n) &= T(7n/10) + n \\
&= T(7^2n/10^2) + n + 7n/10 \\
&= T(7^3n/10^3) + n + 7n/10 + 7^2n/10^2 \\
&... \\
&= T(n(7/10)^k) + n\sum_{i=0}^{k-1}(7/10)^i
\end{aligned}$$

We set $n(7/10)^k$ to 1 compute what we should set k to unroll this until $T(1)$:

$$\begin{aligned}
n(7/10)^k &= 1 \\
\log_{7/10}(1/n) &= k
\end{aligned}$$

Substituting into our formula for $T(n)$:

$$\begin{aligned}
T(n) &= T(n(7/10)^k) + n\sum_{i=0}^{k-1}(7/10)^i \\
&= T(1) + n\sum_{i=0}^{\log_{7/10}(1/n)-1}(7/10)^i
\end{aligned}$$

We are interested in asymptotic behavior, so let n approach infinity. We would like to know what constant this sum approaches as n approaches infinity.

Let S be the infinite series:

$$S = (7/10)^0 + (7/10)^1 + (7/10)^2...$$

If we multiply every element of S by (7/10), then we can do the following

$$\begin{aligned}
S &= (7/10)^0 + (7/10)^1 + (7/10)^2... \\
S(7/10) &= (7/10)^1 + (7/10)^2 + (7/10)^3... \\
S - S(7/10) &= (7/10)^0 = 1 \\
S(3/10)) &= 1 \\
S &= 10/3
\end{aligned}$$

The third step is justified because we can pair every element in both infinite series starting with $(7/10)^1$ and subtract one from the other. As n approaches infinity, the sum $S - S(7/10)$ approaches 1 and thus

S approaches the constant 10/3. Finally, we can substitute this constant into the above summation to get the following:

$$\begin{aligned} T(n) &= T(n(7/10)^k) + n\sum_{i=0}^{k-1}(7/10)^i \\ &= T(1) + n(10/3) \end{aligned}$$

So, the algorithmic complexity is $\Theta(n)$

c. $T(n) = 16T(n/4) + n^2$

Answer: $\mathbf{\Theta(n^2 \log n)}$

Proof by Master Theorem

To use the master theorem case 2, we must show that $f(n) = \Theta(n^{\log_b a})$. Since $a = 16$, $b = 4$, and $f(n) = n^2$, we must show that $n^2 = \Theta(n^{\log_4 16})$. But since $n^{\log_4 16} = n^2$, it is clear that $n^2 = \Theta(n^2)$. Therefore by the master theorem case 2, $T(N) = \Theta(f(n) * \log n) = \Theta(n^2 \log n)$.

Justification by Unrolling / Substitution

First, we unroll with different inputs:

$$\begin{aligned} T(n) &= 16T(n/4) + n^2 \\ T(n/4) &= 16T(n/4^2) + (n/4)^2 \\ T(n/4^2) &= 16T(n/4^3) + (n/4^2)^2 \\ &\dots \end{aligned}$$

We substitute recursively to unroll $T(n)$:

$$\begin{aligned} T(n) &= n^2 + 16T(n/4) \\ &= n^2 + 16(n/4)^2 + 16^2T(n/4^2) \\ &= n^2 + 16(n/4)^2 + 16^2(n/4^2)^2 + 16^3T(n/4^3) \\ &= n^2 + n^2 + n^2 + 16^3T(n/4^3) \\ &\dots \\ &= k(n^2) + 16^kT(n/4^k) \end{aligned}$$

We set $n/4^k$ to 1 compute what we should set k to unroll this until $T(1)$:

$$\begin{aligned} n/4^k &= 1 \\ \log_4 n &= k \end{aligned}$$

Substituting into our formula for $T(n)$:

$$\begin{aligned} T(n) &= \log_4 n * n^2 + 16^{\log_4 n}T(n/4^{\log_4 n}) \\ &= \log_4 n * n^2 + \left(4^{\log_4 n}\right)^2 T(1) \\ &= \log_4 n * n^2 + n^2 * T(1) \end{aligned}$$

We can therefore see that the algorithmic complexity is $\Theta(n^2 \log n)$.

d. $T(n) = 7T(n/3) + n^2$

Answer: $\mathbf{\Theta(n^2)}$

Proof by Master Theorem

To use the master theorem case 3, first, we show that $f(n) = \Omega(n^{\log_b(a+\epsilon)})$ for some $\epsilon > 0$. Since $a = 7$, $b = 3$, and $f(n) = n^2$, we must show that $n = \Omega(n^{\log_3(7+\epsilon)})$ for some $\epsilon > 0$. To determine acceptable values for ϵ:

$$\begin{aligned} n^{\log_3(7+\epsilon)} &\geq n^2 \\ \log_3(7+\epsilon) &\geq 2 \\ \epsilon &\geq 2 \end{aligned}$$

So, for any value $\epsilon \geq 2$, we see that $n = \Omega(n^{\log_2(1+\epsilon)})$. The last step is to prove that $af(n/b) \leq cf(n)$ for some constant $c < 1$. If we substitute a, b, and $f(n)$, we get

$$\begin{aligned} af(n/b) &\leq cf(n) \\ 7(n/3)^2 &\leq cn \\ 7/9 &\leq c \end{aligned}$$

This is true for for $7/9 \leq c$. Therefore by the master theorem, $T(N) = \Theta(f(n)) = \Theta(n^2)$

Justification by Unrolling / Substitution

First, we unroll with different inputs:

$$\begin{aligned} T(n) &= n^2 + 7T(n/3) \\ T(n/3) &= (n/3)^2 + 7T(n/3^2) \\ T(n/3^2) &= (n/3^2)^2 + 7T(n/3^3) \\ &\ldots \end{aligned}$$

We substitute recursively to unroll $T(n)$:

$$\begin{aligned} T(n) &= n^2 + 7T(n/3) \\ &= n^2 + 7(n/3)^2 + 7^2T(n/3^2) \\ &= n^2 + 7(n/3)^2 + 7^2(n/3^2)^2 + 7^3T(n/3^3) \\ &\ldots \\ &= n^2 \sum_{i=0}^{k-1} (7/9)^i + 7^k * T(n/3^k) \end{aligned}$$

We set $n/3^{k+1}$ to 1 compute what we should set k to unroll this until $T(1)$:

$$\begin{aligned} n/3^k &= 1 \\ n &= 3^k \\ k &= \log_3(n) \end{aligned}$$

Substituting k into our formula for $T(n)$:

$$\begin{aligned}
T(n) &= n^2 \sum_{i=0}^{k-1} (7/9)^i + 7^k * T(n/3^k) \\
&= n^2 \sum_{i=0}^{\log_3(n)-1} (7/9)^i + 7^{\log_3(n)} * T(n/3^{\log_3(n)}) \\
&= n^2 \sum_{i=0}^{\log_3(n)-1} (7/9)^i + 7^{\log_3(n)} * T(1) \\
&= n^2 \sum_{i=0}^{\log_3(n)-1} (7/9)^i + n^{\log_3 7} * T(1)
\end{aligned}$$

Let $S = \sum_{i=0}^{\log_3(n)-2} (7/9)^i$. We care only about asymptotic behavior, so let n approach infinity. If we multiply the infinite sequence by of S by $(7/9)$, then we can do the following:

$$\begin{aligned}
S &= (7/9)^0 + (7/9)^1 + (7/9)^2 ... \\
S(7/9) &= (7/9)^1 + (7/9)^2 + (7/9)^3 ... \\
S - S(7/9) &= (7/10)^0 = 1 \\
S(2/9)) &= 1 \\
S &= 9/2 = 4.5
\end{aligned}$$

So, the sequence S approaches the constant 4.5 as n approaches infinity. Substituting back into our previous formula for $T(n)$:

$$T(n) = 4.5n^2 + n^{\log_3 7} * T(1)$$

At this point, we care only about the sizes of the respective exponents, since the larger exponent will determine the algorithmic complexity. Since $\log_3 7 \approx 1.77$, which is smaller than 2, the final complexity is $\Theta(n^2)$.

e. $T(n) = 7T(n/2) + n^2$

Answer: $\mathbf{\Theta(n^{\log_2 7})}$

Proof by Master Theorem

To use the master theorem case 1, we must show that $f(n) = O(n^{\log_b(a-\epsilon)})$ for some $\epsilon > 0$. Since $a = 7$, $b = 2$, and $f(n) = n^2$, we must show that $n^2 = O(n^{\log_2(7-\epsilon)})$ for some $\epsilon > 0$. To determine acceptable values for ϵ:

$$\begin{aligned}
n^{\log_2(7-\epsilon)} &\geq n^2 \\
\log_2(7-\epsilon) &\geq 2 \\
3 &\geq \epsilon
\end{aligned}$$

So, for any value $\epsilon \leq 3$, we see that $n^2 = O(n^{\log_2(1-\epsilon)})$. Therefore by the master theorem, $T(N) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 7})$

Justification by Unrolling / Substitution

First, we unroll with different inputs:

$$
\begin{aligned}
T(n) &= n^2 + 7T(n/2) \\
T(n/2) &= (n/2)^2 + 7T(n/2^2) \\
T(n/2^2) &= (n/2^2)^2 + 7T(n/2^3) \\
&...
\end{aligned}
$$

We substitute recursively to unroll $T(n)$:

$$
\begin{aligned}
T(n) &= n^2 + 7T(n/2) \\
&= n^2 + 7(n/2)^2 + 7^2T(n/2^2) \\
&= n^2 + 7(n/2)^2 + 7^2(n/2^2)^2 + 7^3T(n/2^3) \\
...& \\
&= n^2 \sum_{i=0}^{k-1} (7/4)^i + 7^k * T(n/2^k)
\end{aligned}
$$

We set $n/2^k$ to 1 compute what we should set k to unroll this until $T(1)$:

$$
\begin{aligned}
n/2^k &= 1 \\
n &= 2^k \\
k &= \log_2(n)
\end{aligned}
$$

Substituting k into our formula for $T(n)$:

$$
\begin{aligned}
T(n) &= n^2 \sum_{i=0}^{\log_2(n)-1} (7/4)^i + 7^{\log_2(n)} * T(n/2^{\log_2(n)}) \\
&= n^2 \sum_{i=0}^{\log_2(n)-1} (7/4)^i + n^{\log_2(7)} * T(1)
\end{aligned}
$$

Let $S = \sum_{i=0}^{\log_2(n)-1} (7/4)^i$. We begin with $\log_2(n) - 1$ to observe the pattern from $\log_2(n) - 1$ to 0:

$$
\begin{aligned}
S &= (7/4)^{\log_2(n)-1} + (7/4)^{\log_2(n)-2} + (7/4)^{\log_2(n)-3} + ... \\
&= (7/4)^{\log_2(n/2)} + (7/4)^{\log_2(n/4)} + (7/4)^{\log_2(n/8)} + ... \\
&= (4/7)(7/4)^{\log_2(n)} + (4/7)^2(7/4)^{\log_2(n)} + (4/7)^3(7/4)^{\log_2(n)} + ... \\
&= (7/4)^{\log_2(n)}((4/7) + (4/7)^2 + (4/7)^3 + ...) \\
&= n^{\log_2(7/4)}((4/7) + (4/7)^2 + (4/7)^3 + ...) \\
&= C_1 * n^{\log_2(7/4)}
\end{aligned}
$$

The last step is justified since the sum $((4/7) + (4/7)^2 + (4/7)^3 + ...)$ must converge to a constant as n approaches infinity. If we substitute S back into our above equation for $T(n)$:

$$\begin{aligned} T(n) &= Sn^2 + n^{\log_2(7)} * T(1) \\ &= (C_1 * n^{\log_2(7/4)})n^2 + n^{\log_2(7)} * T(1) \\ &= C_1(n^{(2+\log_2(7/4))}) + n^{\log_2(7)} * T(1) \\ &= C_1(n^{(\log_2 4+\log_2(7/4))}) + n^{\log_2(7)} * T(1) \\ &= C_1(n^{\log_2 7}) + n^{\log_2 7} * T(1) \end{aligned}$$

In other words, **both** the left-side quantity and right-side quantity equal some constant times $n^{\log_2 7}$, so the algorithmic complexity is $\Theta(n^{\log_2 7})$.

f. $T(n) = 2T(n/4) + \sqrt{n}$

Answer: $\mathbf{\Theta(\sqrt{n}\log n)}$

Proof by Master Theorem

To use the master theorem case 2, we must show that $f(n) = \Theta(n^{\log_b a})$. Since $a = 2$, $b = 4$, and $f(n) = \sqrt{n}$, we must show that $\sqrt{n} = \Theta(n^{\log_4 2})$. But since $n^{\log_4 2} = \sqrt{n}$, it is clear that $\sqrt{n} = \Theta(\sqrt{n})$. Therefore by the master theorem case 2, $T(N) = \Theta(f(n) * \log n) = \Theta(\sqrt{n}\log n)$.

Justification by Unrolling / Substitution

First, we unroll with different inputs:

$$\begin{aligned} T(n) &= \sqrt{n} + 2T(n/4) \\ T(n/4) &= \sqrt{n/4} + 2T(n/4^2) \\ T(n/4^2) &= \sqrt{n/4^2} + 2T(n/4^2) \\ &\ldots \end{aligned}$$

We substitute recursively to unroll $T(n)$:

$$\begin{aligned} T(n) &= \sqrt{n} + 2T(n/4) \\ &= \sqrt{n} + 2\sqrt{n/4} + 2^2T(n/4^2) \\ &= \sqrt{n} + 2\sqrt{n/4} + 2^2\sqrt{n/4^2} + 2^3T(n/4^3) \\ &= \sqrt{n} + \sqrt{n} + \sqrt{n} + 2^3T(n/4^3) \\ &\ldots \\ &= k\sqrt{n} + 2^kT(n/4^k) \end{aligned}$$

We set $n/4^k$ to 1 compute what we should set k to unroll this until $T(1)$:

$$\begin{aligned} n/4^k &= 1 \\ k &= \log_4 n \end{aligned}$$

Substituting k into our formula for $T(n)$:

$$\begin{aligned} T(n) &= (\log_4 n)\sqrt{n} + 2^{\log_4 n}T(n/4^{\log_4 n}) \\ &= (\log_4 n)\sqrt{n} + \sqrt{n}T(1) \end{aligned}$$

Thus, the algorithmic complexity is $\Theta(\sqrt{n}\log n)$.

g. $T(n) = T(n-2) + n^2$

Answer: $\mathbf{\Theta(n^3)}$

Proof by Substitution

To determine a correct guess, first we unroll the recurrence with different inputs:

$$\begin{aligned} T(n) &= n^2 + T(n-2) \\ T(n-2) &= (n-2)^2 + T(n-4) \\ T(n-4) &= (n-4)^2 + T(n-6) \\ &... \end{aligned}$$

We substitute recursively to unroll $T(n)$:

$$\begin{aligned} T(n) &= n^2 + T(n-2) \\ &= n^2 + (n-2)^2 + T(n-4) \\ &= n^2 + (n-2)^2 + (n-4)^2 + T(n-6) \\ &... \\ &= \sum_{i=0}^{k-1} (n-2i)^2 + T(n-2k) \end{aligned}$$

We set $n - 2k$ to 1 compute what we should set k to unroll $T(n)$ until $T(1)$:

$$\begin{aligned} n - 2k &= 1 \\ k &= \frac{n-1}{2} \end{aligned}$$

Substituting back into our formula:

$$\begin{aligned} T(n) &= \sum_{i=0}^{k-1} (n-2i)^2 + T(n-2k) \\ &= \sum_{i=0}^{k-1} (n-2i)^2 + T(1) \end{aligned}$$

Let $S = \sum_{i=0}^{k-1} (n-2i)^2$. Since the above formula adds only $T(1)$ to S, the algorithmic complexity of $T(n)$ will be the same as S.

If we unroll S, we can get the following sum of sums:

$$\begin{aligned} S &= \sum_{i=0}^{k-1} (n-2i)^2 \\ &= n^2 + (n-2)^2 + (n-4)^2 + (n-6)^2 ... \\ &= n^2 + (n^2 - 4n + 4) + (n^2 - 8n + 16) + (n^2 - 12n + 36) ... \\ &= (n^2 + n^2 + n^2 + n^2 + ...) + (-4n + -8n + -12n + ...) + (4 + 16 + 36 + ...) \\ &= (n^2 + n^2 + n^2 + n^2 + ...) + -4n(1 + 2 + 3 + ...) + 4(1 + 4 + 9 + ...) \\ &= k(n^2) + -4n \sum_{i=1}^{k-1} i + 4 \sum_{i=1}^{k-1} i^2 \end{aligned}$$

The two sums $\sum_{i=1}^{k-1} i$ and $\sum_{i=1}^{k-1} i^2$ have closed form solutions:

$$\sum_{i=1}^{k-1} i = \frac{k(k-1)}{2}$$
$$\sum_{i=1}^{k-1} i^2 = \frac{k(k-1)(2k-1)}{6}$$

Substituting the closed form back into our formulas yields:

$$\begin{aligned} S &= k(n^2) + -4n(\frac{k(k-1)}{2}) + 4(\frac{k(k-1)(2k-1)}{6}) \\ &= k(n^2) - 2nk^2 + 2nk + \frac{4k^3}{3} - 2k^2 + \frac{2k}{3} \end{aligned}$$

And finally, substituting $\frac{n-1}{2}$ for k:

$$\begin{aligned} S &= k(n^2) - 2nk^2 + 2nk + \frac{4k^3}{3} - 2k^2 + \frac{2k}{3} \\ &= (\frac{n-1}{2})(n^2) - 2n(\frac{n-1}{2})^2 + 2n(\frac{n-1}{2}) + \frac{4(\frac{n-1}{2})^3}{3} - 2(\frac{n-1}{2})^2 + \frac{2(\frac{n-1}{2})}{3} \\ &= \frac{n^3}{6} + \frac{3n^2}{2} + \frac{n}{3} - 1 \end{aligned}$$

Therefore, $S = \Theta(n^3)$ and therefore $T(n) = \Theta(n^3)$.

Problem 4-2

Throughout this book, we assume that parameter passing during procedure calls takes constant time, even if an N-element array is being passed. This assumption is valid in most systems because a pointer to the array is passed, not the array itself. This problem examines the implications of three parameter-passing strategies:

1. An array is passed by pointer. Time = $\Theta(1)$.
2. An array is passed by copying. Time = $\Theta(N)$, where N is the size of the array.
3. An array is passed by copying only the subrange that might be accessed by the called procedure. Time = $\Theta(q - p + 1)$ if the subarray A[p...q] is passed.

Questions:

a. Consider the recursive binary search algorithm for finding a number in a sorted array (see Exercise 2.3-5). Give recurrences for the worst-case running times of binary search when arrays are passed using each of the three methods above, and give good upper bounds on the solutions of the recurrences. Let N be the size of the original problem and n be the size of a subproblem.

Answer:

The recurrence relation for worst-case binary search is the following:

$$T(N) = T(N/2) + X$$
$$T(1) = \Theta(1)$$

where X is the time to execute binary search if we exclude the time on the recursive portion. Normally, we would expect X to be $\Theta(1)$. However, in this problem, we are told that the time it takes to perform the act of calling a function itself is not constant. In that case, we would have to rewrite this recurrence for each case.

1. Recurrence relation: $\mathbf{T(N) = T(N/2) + \Theta(1)}$
   Running time: $\mathbf{\Theta(\log N)}$
   Proof.
   By the master theorem Case 2, we can see that the running time of the above recurrence is $\Theta(1)$. With Case 2, we need only show that $f(N) = \Theta(N^{\log_b a})$, where $a \geq 1$ and $b > 1$. In this recurrence, $a = 1$, $b = 2$, and $f(N) = 1$. In that case, the expression $N^{\log_b a}$ equals $N^{\log_2 1} = N^0 = 1$. And of course, $1 = \Theta(1)$, so by the master theorem $T(N) = \Theta(N^{\log_b a} \log N) = \Theta(\log N)$.
2. Recurrence relation: $\mathbf{T(N) = T(N/2) + \Theta(N)}$
   Running time: $\mathbf{\Theta(N)}$
   Proof.
   To use the master theorem case 3, first, we show that $f(n) = \Omega(n^{\log_b(a+\epsilon)})$ for some $\epsilon > 0$. Since $a = 1$, $b = 2$, we must show that $N = \Omega(N^{\log_2(1+\epsilon)})$ for some $\epsilon > 0$. If we set $\epsilon = 3$, then by inspection we can see that $N = \Omega(N^{\log_2(1+3)}) = \Omega(N^2)$, which is obviously true. The last step is to prove that $af(N/b) \leq cf(N)$ for some constant $c < 1$. If we substitute a, b, and $f(N)$, we get

$$\begin{aligned} af(N/b) &\leq cf(n) \\ 1(N/2) &\leq cN \\ 1/2 &\leq c \end{aligned}$$

   So, c can be any constant greater than $1/2$. Therefore by the master theorem, $T(N) = \Theta(f(n)) = \Theta(N)$
3. Recurrence relation: $\mathbf{T(N) = T(N/2) + \Theta(\frac{1}{2}N)}$
   Running time: $\mathbf{\Theta(N)}$
   Proof.
   To use the master theorem case 3, first, we show that $f(n) = \Omega(n^{\log_b(a+\epsilon)})$ for some $\epsilon > 0$. Since $a = 1$, $b = 2$, and $f(N) = \frac{1}{2}N$, we must show that $\frac{1}{2}N = \Omega(N^{\log_2(1+\epsilon)})$ for some $\epsilon > 0$. If we set $\epsilon = 3$, then by inspection we can see that $\frac{1}{2}N = \Omega(N^{\log_2(1+3)}) = \Omega(N^2)$, which is obviously true. The last step is to prove that $af(N/b) \leq cf(N)$ for some constant $c < 1$. If we substitute a, b, and $f(N)$, we get

$$\begin{aligned} af(N/b) &\leq cf(n) \\ 1(N/4) &\leq cN \\ 1/4 &\leq c \end{aligned}$$

   So, c can be any constant greater than $1/4$. Therefore by the master theorem, $T(N) = \Theta(f(n)) = \Theta(N)$

b. Redo part (a) for the MERGE-SORT algorithm from Section 2.3.1.

Answer:

The recurrence relation for worst-case merge sort is the following:

$$\begin{aligned} T(N) &= 2T(N/2) + \Theta(N) + \Theta(X) \\ T(1) &= \Theta(1) \end{aligned}$$

where X is the time to execute all the operations except for the merging step (represented by $\Theta(N)$). Since merging is always linear, the effect of the three different ways of calling a function will not affect the algorithmic complexity.

1. Recurrence relation: $\mathbf{T(N) = 2T(N/2) + \Theta(N+1)}$

   Running time: $\mathbf{\Theta(N \log N)}$

   Proof.

   By the master theorem Case 2, we can see that the running time of the above recurrence is $\Theta(N \log N)$. With Case 2, we need only show that $f(N) = \Theta(N^{\log_b a})$, where $a \geq 1$ and $b > 1$. In this recurrence, $a = 2$, $b = 2$, and $f(N) = N + 1$. In that case, the expression $N^{\log_b a}$ equals $N^{\log_2 2} = N^1 = N$. And of course, $N + 1 = \Theta(N)$, so by the master theorem $T(N) = \Theta(N^{\log_b a} \log N) = \Theta(N \log N)$.

2. Recurrence relation: $\mathbf{T(N) = 2T(N/2) + \Theta(2N)}$

   Running time: $\mathbf{\Theta(N \log N)}$

   Proof.

   By the master theorem Case 2, we can see that the running time of the above recurrence is $\Theta(N \log N)$. With Case 2, we need only show that $f(N) = \Theta(N^{\log_b a})$, where $a \geq 1$ and $b > 1$. In this recurrence, $a = 2$, $b = 2$, and $f(N) = 2N$. In that case, the expression $N^{\log_b a}$ equals $N^{\log_2 2} = N^1 = N$. And of course, $2N = \Theta(N)$, so by the master theorem $T(N) = \Theta(N^{\log_b a} \log N) = \Theta(N \log N)$.

3. Recurrence relation: $\mathbf{T(N) = 2T(N/2) + \Theta(\frac{3}{2}N)}$

   Running time: $\mathbf{\Theta(N \log N)}$

   Proof.

   By the master theorem Case 2, we can see that the running time of the above recurrence is $\Theta(N \log N)$. With Case 2, we need only show that $f(N) = \Theta(N^{\log_b a})$, where $a \geq 1$ and $b > 1$. In this recurrence, $a = 2$, $b = 2$, and $f(N) = \frac{3}{2}N$. In that case, the expression $N^{\log_b a}$ equals $N^{\log_2 2} = N^1 = N$. And of course, $\frac{3}{2}N = \Theta(N)$, so by the master theorem $T(N) = \Theta(N^{\log_b a} \log N) = \Theta(N \log N)$.