CSC 225 - Summer 2019

Recurrence Relations II

Bill Bird

Department of Computer Science
University of Victoria

June 11, 2019

Recurrence Exercises (1)

Solve the following recurrence relations to find a closed form.

1. (Assume that n is a power of 2).

$$\begin{aligned} T_1(n) &= 1 && \text{if } n = 1 \\ &= T(n/2) + 1 && \text{if } n > 1 \end{aligned}$$

2. (Assume that n is a power of 3).

$$\begin{aligned} T_2(n) &= 1 && \text{if } n = 1 \\ &= 3T(n/3) + n \log_3 n && \text{if } n > 1 \end{aligned}$$

3. (Assume that n is a power of 4).

$$\begin{aligned} T_3(n) &= 1 && \text{if } n = 1 \\ &= 5T(n/4) + n^2 && \text{if } n > 1 \end{aligned}$$

Recurrence Exercises - Solutions (1)

$$\begin{aligned} T_1(n) &= 1 && \text{if } n = 1 \\ &= T(n/2) + 1 && \text{if } n \geq 2 \end{aligned}$$

The closed form is

$$T_1(n) = 1 + \log_2 n$$

which is in $\Theta(\log_2 n)$.

Recurrence Exercises - Solutions (2)

$$\begin{aligned} T_2(n) &= 1 && \text{if } n = 1 \\ &= 3T(n/3) + n\log_3 n && \text{if } n \geq 2 \end{aligned}$$

The closed form is

$$T_2(n) = n + \frac{1}{2}n(\log_3 n(\log_3 n + 1))$$

which is in $\Theta(n(\log_3 n)^2)$

Recurrence Exercises - Solutions (3)

$$\begin{aligned} T_3(n) &= 1 && \text{if } n = 1 \\ &= 5T(n/4) + n^2 && \text{if } n \geq 2 \end{aligned}$$

The unsimplified closed form is

$$T_3(n) = 5^{\log_4 n} + \sum_{j=0}^{\log_4 n - 1} 5^j 4^{2(k-j)}$$

which can be simplified to

$$\begin{aligned} T_3(n) &= 5^{\log_4 n} + 4^{2\log_4 n} \sum_{j=0}^{\log_4 n - 1} \left(\frac{5}{16}\right)^j \\ &\leq n^{\log_4 5} + cn^2 \\ &\leq (c+1)n^2 \end{aligned}$$

(you will not be expected to simplify recurrences this way on exams)

Merge Sort Analysis Again (1)

```
procedure MergeSort(A)
    n ← Length(A)
    if n = 1 then
        //An array of size 1 is already sorted.
        return
    end if
    n1 ← ⌊n/2⌋
    n2 ← n − n1
    Split A into two arrays A1 (with size n1) and A2 (with size n2).
    MergeSort(A1)
    MergeSort(A2)
    Merge the sorted A1 and A2 together into A.
end procedure
```

Most of the recursive algorithms we have encountered so far can be analysed by deriving a recurrence relation, solving the recurrence to obtain a closed form, then proving the closed form to be correct.

Merge Sort Analysis Again (2)

```
procedure MergeSort(A)
    n ← Length(A)
    if n = 1 then
        //An array of size 1 is already sorted.
        return
    end if
    n_1 ← ⌊n/2⌋
    n_2 ← n − n_1
    Split A into two arrays A_1 (with size n_1) and A_2 (with size n_2).
    MergeSort(A_1)
    MergeSort(A_2)
    Merge the sorted A_1 and A_2 together into A.
end procedure
```

However, depending on the exact choice of operations counted, different recurrences can be derived for the same algorithm.

Merge Sort Analysis Again (3)

```
procedure MergeSort(A)
    n ← Length(A)
    if n = 1 then
        //An array of size 1 is already sorted.
        return
    end if
    n_1 ← ⌊n/2⌋
    n_2 ← n − n_1
    Split A into two arrays A_1 (with size n_1) and A_2 (with size n_2).
    MergeSort(A_1)
    MergeSort(A_2)
    Merge the sorted A_1 and A_2 together into A.
end procedure
```

It turns out that any reasonable assumptions will lead to the correct asymptotic running time.

Merge Sort Analysis Again (4)

```
procedure MergeSort(A)
    n ← Length(A)
    if n = 1 then
        //An array of size 1 is already sorted.
        return
    end if
    n1 ← ⌊n/2⌋
    n2 ← n − n1
    Split A into two arrays A1 (with size n1) and A2 (with size n2).
    MergeSort(A1)
    MergeSort(A2)
    Merge the sorted A1 and A2 together into A.
end procedure
```

Claim: For any constants $b, c, d > 0$, the recurrence

$$\begin{aligned} T(n) &= b && \text{if } n = 1 \\ &= 2T(n/2) + cn + d && \text{if } n \geq 2 \end{aligned}$$

is asymptotically equivalent to the running time for Merge Sort.

Merge Sort Analysis Again (5)

$$
\begin{aligned}
T(n) &= b && \text{if } n = 1 \\
&= 2T(n/2) + cn + d && \text{if } n \geq 2
\end{aligned}
$$

Step 1:

$$T(n) = 2T(n/2) + cn + d$$

Step 2:

$$
\begin{aligned}
T(n) &= 2\left[2T(n/4) + \frac{cn}{2} + d\right] + cn + d \\
&= 4T(n/4) + \frac{2cn}{2} + 2d + cn + d \\
&= 4T(n/4) + 2cn + 3d
\end{aligned}
$$

Merge Sort Analysis Again (6)

$$
\begin{aligned}
T(n) &= b && \text{if } n = 1 \\
&= 2T(n/2) + cn + d && \text{if } n \geq 2
\end{aligned}
$$

(end of step 2)

$$T(n) = 4T(n/4) + 2cn + 3d$$

Step 3:

$$
\begin{aligned}
T(n) &= 4\left[2T(n/8) + \frac{cn}{4} + d\right] + 2cn + 3d \\
&= 8T(n/8) + 4\frac{cn}{4} + 4d + 2n + 3d \\
&= 8T(n/8) + 3cn + 7d
\end{aligned}
$$

Merge Sort Analysis Again (7)

$$\begin{aligned} T(n) &= b && \text{if } n = 1 \\ &= 2T(n/2) + cn + d && \text{if } n \geq 2 \end{aligned}$$

(end of step 3)

$$T(n) = 8T(n/8) + 3cn + 7d$$

Step i:

$$T(n) = 2^i T(n/2^i) + i \cdot cn + (2^i - 1)d$$

Choose $i = \log_2 n$:

$$\begin{aligned} T(n) &= 2^{\log_2 n} T\left(n/2^{\log_2 n}\right) + cn \log_2 n + (2^{\log_2 n} - 1)d \\ &= nT(1) + cn \log_2 n + d(n-1) \\ &= bn + cn \log_2 n + dn - d \in \Theta(n \log_2 n) \end{aligned}$$

Merge Sort Analysis Again (8)

```
procedure MergeSort(A)
    n ← Length(A)
    if n = 1 then
        //An array of size 1 is already sorted.
        return
    end if
    n1 ← ⌊n/2⌋
    n2 ← n − n1
    Split A into two arrays A1 (with size n1) and A2 (with size n2).
    MergeSort(A1)
    MergeSort(A2)
    Merge the sorted A1 and A2 together into A.
end procedure
```

There are many recurrences whose asymptotic behavior is $n \log_2 n$. More significantly, the exact closed form of a recurrence is not entirely relevant for asymptotic purposes.

Merge Sort Analysis Again (9)

```
procedure MergeSort(A)
    n ← Length(A)
    if n = 1 then
        //An array of size 1 is already sorted.
        return
    end if
    n1 ← ⌊n/2⌋
    n2 ← n − n1
    Split A into two arrays A1 (with size n1) and A2 (with size n2).
    MergeSort(A1)
    MergeSort(A2)
    Merge the sorted A1 and A2 together into A.
end procedure
```

For certain classes of recurrence relations, there is a general method to find a Big-Theta expression without finding the closed form.

Anatomy of a Recurrence (1)

Recurrences resulting from analysis of divide and conquer algorithms, such as Merge Sort, normally can be written in the form

$$\begin{aligned} T(n) &= c && \text{if } n = 1 \\ &= aT(n/b) + g(n) && \text{if } n > 1 \end{aligned}$$

where $g(n)$ is some function of n.

For example, with $a = 2, b = 2, c = 1$ and $g(n) = n + 1$, we obtain the recurrence

$$\begin{aligned} T(n) &= 1 && \text{if } n = 1 \\ &= 2T(n/2) + n + 1 && \text{if } n \geq 2 \end{aligned}$$

which was used in the analysis of Merge Sort earlier in the course.

Anatomy of a Recurrence (2)

$$\begin{aligned} T(n) &= c && \text{if } n = 1 \\ &= aT(n/b) + g(n) && \text{if } n > 1 \end{aligned}$$

Consider the result of repeated substitution on the abstract form above.

Step 1:

$$T(n) = aT(n/b) + g(n)$$

Step 2:

$$\begin{aligned} T(n) &= a\left[aT(n/b^2) + g(n/b)\right] + g(n) \\ &= a^2 T(n/b^2) + ag(n/b) + g(n) \end{aligned}$$

Step 3:

$$\begin{aligned} T(n) &= a^2\left[aT(n/b^3) + g(n/b^2)\right] + ag(n/b) + g(n) \\ &= a^3 T(n/b^3) + a^2 g(n/b^2) + ag(n/b) + g(n) \end{aligned}$$

Anatomy of a Recurrence (3)

$$\begin{aligned} T(n) &= c && \text{if } n = 1 \\ &= aT(n/b) + g(n) && \text{if } n > 1 \end{aligned}$$

Consider the result of repeated substitution on the abstract form above.

Step 3:

$$T(n) = a^3 T(n/b^3) + a^2 g(n/b^2) + ag(n/b) + g(n)$$

Step i:

$$T(n) = a^i T(n/b^i) + \sum_{k=0}^{i-1} a^k g(n/b^k)$$

Set $i = \log_b n$:

$$T(n) = a^{\log_b n} T(1) + \sum_{k=0}^{\log_b n - 1} a^k g(n/b^k)$$

Anatomy of a Recurrence (4)

$$\begin{aligned} T(n) &= c && \text{if } n = 1 \\ &= aT(n/b) + g(n) && \text{if } n > 1 \end{aligned}$$

Consider the result of repeated substitution on the abstract form above.

Set $i = \log_b n$:

$$T(n) = a^{\log_b n} T(1) + \sum_{k=0}^{\log_b n - 1} a^k g(n/b^k)$$

$$= ca^{\log_b n} + \sum_{k=0}^{\log_b n - 1} a^k g(n/b^k)$$

If our goal is only to find a simple Big-Theta expression for the asymptotic behavior of $T(n)$, it may not be necessary to find the exact closed form.

Anatomy of a Recurrence (5)

$$T(n) = ca^{\log_b n} + \sum_{k=0}^{\log_b n - 1} a^k g(n/b^k)$$

We can simplify the $a^{\log_b n}$ term by using the identity

$$p^{\log_q x} = x^{\log_q p}$$

to obtain

$$T(n) = cn^{\log_b a} + \sum_{k=0}^{\log_b n - 1} a^k g(n/b^k)$$

Anatomy of a Recurrence (6)

$$T(n) = \underbrace{cn^{\log_b a}}_{\text{Term X}} + \underbrace{\sum_{k=0}^{\log_b n - 1} a^k g(n/b^k)}_{\text{Term Y}}$$

Intuitively, there are three cases:

1. **Case 1**: Term X is significantly larger than term Y asymptotically. In this case, it is clear that

$$T(n) \in \Theta\left(n^{\log_b a}\right)$$

2. **Case 2**: Terms X and Y are basically equal asymptotically.
3. **Case 3**: Term X is significantly smaller than term Y asymptotically.

Anatomy of a Recurrence (7)

$$T(n) = \underbrace{cn^{\log_b a}}_{\text{Term X}} + \underbrace{\sum_{k=0}^{\log_b n - 1} a^k g(n/b^k)}_{\text{Term Y}}$$

Using a theorem called the 'Master Theorem', we can obtain a Big-Theta expression for $T(n)$ without solving it by determining which of the three cases on the previous slide applies.

Master Theorem (1)

Theorem: (The Master Theorem)
Let T be a recurrence of the form

$$\begin{aligned} T(n) &= c && \text{if } n = 1 \\ &= aT(n/b) + g(n) && \text{if } n > 1 \end{aligned}$$

Case 1: If $g(n) \in O\left(n^{\log_b a - \varepsilon}\right)$ for some constant $\varepsilon > 0$, then

$$T(n) \in \Theta(n^{\log_b a})$$

Case 2: If $g(n) \in \Theta\left(n^{\log_b a}(\log_b n)^k\right)$ for some constant $k \geq 0$, then

$$T(n) \in \Theta\left(n^{\log_b a}(\log_b n)^{k+1}\right)$$

Case 3: If $g(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right)$ for some constant $\varepsilon > 0$, and there exists some constant $\delta < 1$ such that $ag(n/b) < \delta g(n)$ for large n, then

$$T(n) \in \Theta\left(g(n)\right)$$

Master Theorem (2)

The master theorem provides an easy[1] method for asymptotically bounding divide and conquer recurrence relations.

If the master theorem is needed on an exam, you will be provided the full statement of the theorem on the previous slide.

If you are asked to **solve** a recurrence, you must provide an exact closed form (so the master theorem is not applicable). If you are asked to find a Big-Theta expression for a recurrence (and not specifically directed to use a particular method), you are free to either solve the recurrence or use the master theorem.

[1] For a certain perverse definition of easy.

Recurrence Exercises Again (1)

Give a simple Big-Theta expression for the asymptotic behavior of the following recurrences.

1. (Assume that n is a power of 2).

$$\begin{aligned} T_1(n) &= 1 && \text{if } n = 1 \\ &= T(n/2) + 1 && \text{if } n > 1 \end{aligned}$$

2. (Assume that n is a power of 3).

$$\begin{aligned} T_2(n) &= 1 && \text{if } n = 1 \\ &= 3T(n/3) + n \log_3 n && \text{if } n > 1 \end{aligned}$$

3. (Assume that n is a power of 4).

$$\begin{aligned} T_3(n) &= 1 && \text{if } n = 1 \\ &= 5T(n/4) + n^2 && \text{if } n > 1 \end{aligned}$$

Recurrence Exercises - Solutions (1)

$$\begin{aligned} T_1(n) &= 1 && \text{if } n = 1 \\ &= T(n/2) + 1 && \text{if } n \geq 2 \end{aligned}$$

Solution with Master Theorem:[1]
For this recurrence, $a = 1, b = 2, c = 1$ and $g(n) = 1$.

$n^{\log_b a} = n^{\log_2 1} = n^0 = 1$, so

$$g(n) \in \Theta\left(n^{\log_b a}(\log_b n)^0\right),$$

which falls under case 2 of the Master Theorem. Therefore,

$$T_1(n) \in \Theta\left(n^{\log_b a}(\log_b n)^1\right) = \Theta(\log_2 n)$$

[1] This recurrence is probably easier to solve directly.

Recurrence Exercises - Solutions (2)

$$T_2(n) = 1 \qquad \text{if } n = 1$$
$$= 3T(n/3) + n\log_3 n \qquad \text{if } n \geq 2$$

Solution with Master Theorem:
For this recurrence, $a = 3, b = 3, c = 1$ and $g(n) = n\log_3 n$.

$n^{\log_b a} = n^{\log_3 3} = n^1 = n$, so

$$g(n) = n\log_3 n \in \Theta\left(n^{\log_b a}(\log_b n)^1\right),$$

which falls under case 2 of the Master Theorem. Therefore,

$$T_2(n) \in \Theta\left(n^{\log_b a}(\log_b n)^2\right) = \Theta(n(\log_3 n)^2)$$

Recurrence Exercises - Solutions (3)

$$\begin{aligned} T_3(n) &= 1 && \text{if } n = 1 \\ &= 5T(n/4) + n^2 && \text{if } n \geq 2 \end{aligned}$$

Solution with Master Theorem:
For this recurrence, $a = 5, b = 4, c = 1$ and $g(n) = n^2$.

$n^{\log_b a} = n^{\log_4 5}$, and since $\log_4 5 < 2$, the value $\varepsilon = 2 - \log_4 5$ is positive, so

$$g(n) = n^2 \in \Omega\left(n^{\log_b a + \varepsilon}\right),$$

which falls under case 3 of the Master Theorem. To apply the Theorem in case 3, we also have to establish that for sufficiently large n,

$$ag(n/b) = 5(n/4)^2 < \delta g(n) = \delta n^2$$

for some $\delta < 1$.

Recurrence Exercises - Solutions (4)

$$\begin{aligned} T_3(n) &= 1 && \text{if } n = 1 \\ &= 5T(n/4) + n^2 && \text{if } n \geq 2 \end{aligned}$$

Taking $\delta = \frac{1}{2}$ is sufficient, since

$$5(n/4)^2 = \frac{5}{4^2}n^2 = \frac{5}{16}n^2 < \frac{1}{2}n^2$$

Therefore, case 3 applies and

$$T_3(n) \in \Theta(g(n)) = \Theta(n^2)$$