Divide-and-Conquer Algorithms and Recurrence Relations

Niloufar Shafiei

Divide-and-conquer algorithms

Divide-and-conquer algorithms:

1. Dividing the problem into smaller sub-problems
2. Solving those sub-problems
3. Combining the solutions for those smaller sub-problems to solve the original problem

Recurrences are used to analyze the computational complexity of divide-and-conquer algorithms.

Example:

Binary search is a divide-and-conquer algorithm.

Divide-and-conquer recurrence

- Assume a divide-and-conquer algorithm divides a problem of size n into a sub-problems.
- Assume each sub-problem is of size n/b.
- Assume f(n) extra operations are required to combine the solutions of sub-problems into a solution of the original problem.
- Let T(n) be the number of operations required to solve the problem of size n.

$$T(n) = a\,T(n/b) + f(n)$$

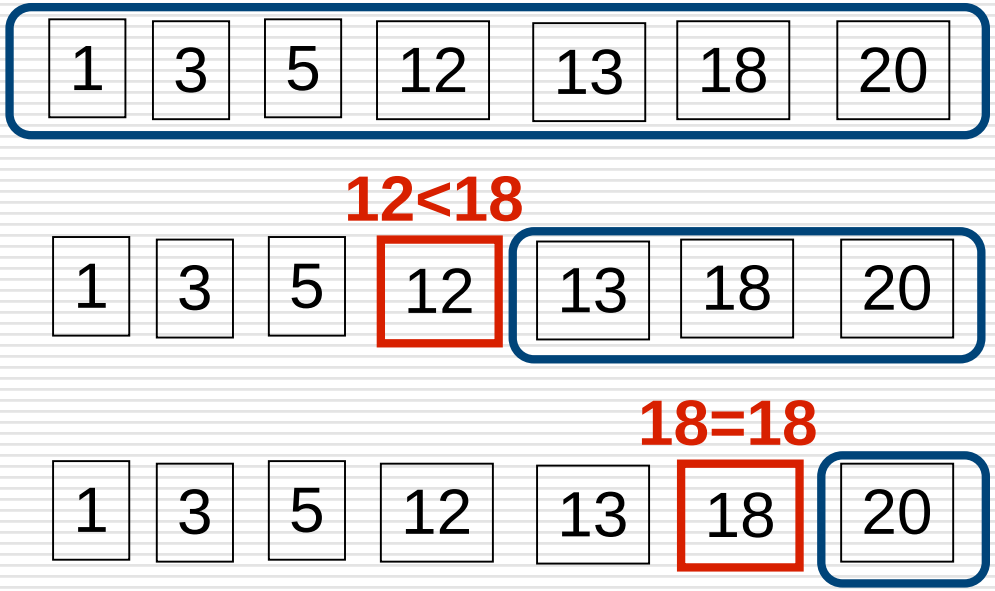
In order to make the recurrence well defined T(n/b) term will actually be either $T(\lceil n/b \rceil)$ or $T(\lfloor n/b \rfloor)$.

The recurrence will also have to have initial conditions. (e.g. T(1) or T(0))

Example

Binary search:

Search for **18**



$T(n) = T(\lfloor n/2 \rfloor) + 2$

$T(1) = 2$

Determine which half of the list to choose and determine whether any terms of the list remain

Example

Finding the maximum and minimum of a sequence:

Consider the sequence $a_1, a_2, \ldots, a_n$.

If n=1, then a_1 is the maximum and the minimum.

If n>1, split the sequence into two sequences, either where both have the same number of elements or where one of the sequences has one more element than the other.

The problem is reduced to finding the maximum and minimum of each of the two smaller sequences.

The solution to the original problem results from the comparison of the separate maxima and minima of the two smaller sequences.

Find a recurrence T(n) that be the number of operations required to solve the problem of size n.

Solution:

The problem of size n can be reduced into two problems of size $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$.

Two comparisons are required to find the original solution from those sub-problems.

$$T(1) = 1 \qquad T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 2$$

Example

Merge sort:

The merge sort algorithm splits a list with n elements into two list with $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$ elements. (The list with 1 element is considered sorted.)

It uses less than n comparison to merge two sorted lists of $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$ elements.

Find a recurrence T(n) that represents the number of operations required to solve the problem of size n.

(The merge sort may have less operations than T(n))

Solution:

The problem of size n can be reduced into two problems of size $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$.

n comparisons are required to find the original solution from those sub-problems.

$T(1) = 0$ $\qquad\qquad$ $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n$

Master theorem

There is a theorem that gives asymptotic behavior of any sequence defined by a divide-and-conquer recurrence with $f(n)=c.n^d$ for constants $c>0$ and $d\geq 0$.

This theorem is sometimes called the master theorem.

Master theorem

Theorem:

Assume a sequence is defined by a recurrence equation

$$T(n) = aT(n/b) + cn^d \text{ for } n>1,$$

where $a \geq 1$, $b \geq 2$, $c>0$ and $d \geq 0$ are constants and n/b is either $\lceil n/b \rceil$ or $\lfloor n/b \rfloor$, then one of the following holds.

$$T(n) = \Theta(n^d) \quad \text{if } a < b^d$$
$$T(n) = \Theta(n^d \log n) \quad \text{if } a = b^d$$
$$T(n) = \Theta(n^{\log_b a}) \quad \text{if } a > b^d$$

Example

Assume an algorithm behavior is determined using the following recurrence.Give big-Theta estimate for its complexity.

$T(1) = 1$

$T(n) = T(\lceil n/2 \rceil) + 4$, for $n \geq 2$

Solution:

$a=1$, $b=2$, $c=4$ and $d=0$

So, $a = b^d = 1$.

By Master theorem, $T(n) = \Theta(n^d \log n) = \Theta(\log n)$.

Example

Assume an algorithm behavior is determined using the following recurrence.Give big-Theta estimate for its complexity.

$T(1) = 0$

$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$, for $n \geq 2$

Solution:

$a=2$, $b=2$, $c=1$ and $d=0$

So, $a > b^d = 1$.

By Master theorem, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 2}) = \Theta(n)$.

Example

Assume an algorithm behavior is determined using the following recurrence.Give big-Theta estimate for its complexity.

$x_0 = 0$

$x_n = 7x_{\lfloor n/5 \rfloor} + 9n^2$, for $n \geq 1$

Solution:

a=7, b=5, c=9 and d=2

So, $a < b^d = 25$.

By Master theorem, $T(n) = \Theta(n^d) = \Theta(n^2)$.

Example

Give big-Theta estimate for the merge sort algorithm.

Solution:

By previous example, we know the following recurrence represent the merge sort algorithm.

$T(1) = 0$

$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n$ for $n>1$.

$a=2$, $b=2$, $c=1$ and $d=1$

So, $a = b^d = 2$.

By Master theorem, $T(n) = \Theta(n^d \log n) = \Theta(n \log n)$.

Example

Give big-Theta estimate for following recurrence.

$f(n) = 2f(\sqrt{n}) + 1$ when a perfect square $n > 1$,

$f(2)=1$.

(Hint: assume $m = \log_2 n$.)

Solution:

$m = \log_2 n$ $\qquad$ $2^m = n$

$f(2^m) = 2\, f((2^m)^{1/2}) + 1$ $\qquad$ $f(2^m) = 2\, f(2^{m/2}) + 1$

$g(m) = 2\, g(m/2) + 1$

$a=2$, $b=2$, $c=1$ and $d=0$

So, $a > b^d = 1$.

By Master theorem, $g(m) = \Theta(m^{\log_b a}) = \Theta(m^{\log_2 2}) = \Theta(m)$.

So, $f(n) = \Theta(\log_2 n)$.

Recommended exercises

7, 17,21,22

Eric Ruppert's Notes about Solving Recurrences

(http://www.cse.yorku.ca/course_archive/2007-08/F/1019/A/recurrence.pdf)