Divide and Conquer Algorithms Recurrence Relations

Dr. Navjot Singh

Design and Analysis of Algorithms

Divide and Conquer

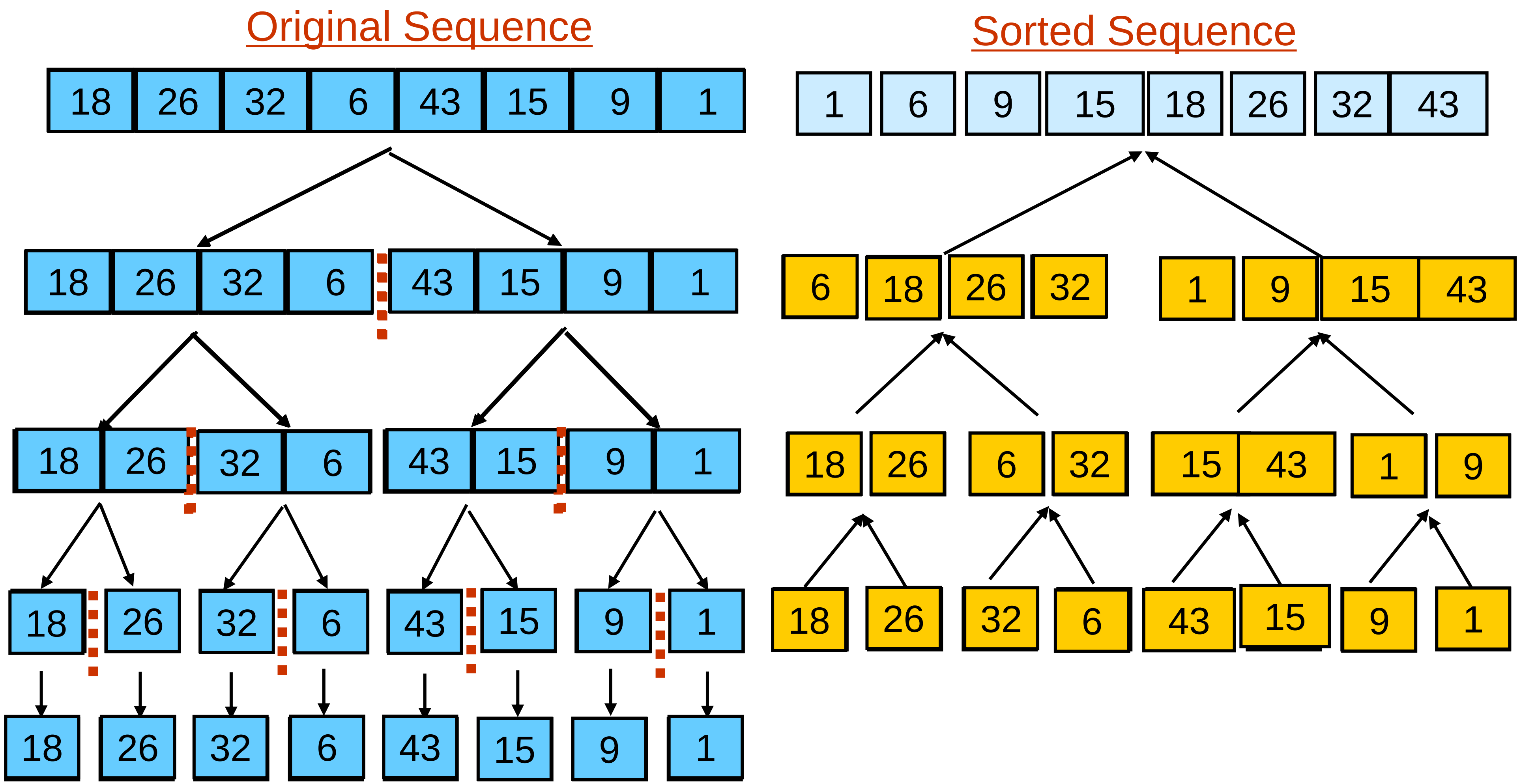
- Recursive in structure
  - ***Divide*** the problem into sub-problems that are similar to the original but smaller in size
  - ***Conquer*** the sub-problems by solving them recursively. If they are small enough, just solve them in a straightforward manner.
  - ***Combine*** the solutions to create a solution to the original problem

An Example: Merge Sort

***Sorting Problem*:** Sort a sequence of *n* elements into non-decreasing order.

- ***Divide*:** Divide the *n*-element sequence to be sorted into two subsequences of *n/2* elements each
- ***Conquer:*** Sort the two subsequences recursively using merge sort.
- ***Combine*:** Merge the two sorted subsequences to produce the sorted answer.

Merge Sort – Example



Merge-Sort (A, p, r)

INPUT: a sequence of *n* numbers stored in array A

OUTPUT: an ordered sequence of *n* numbers

```
MergeSort (A, p, r)   // sort A[p..r] by divide & conquer
    if p < r
        then q ← ⌊(p+r)/2⌋
            MergeSort (A, p, q)
            MergeSort (A, q+1, r)
            Merge (A, p, q, r) // merges A[p..q] with A[q+1..r]
```

Initial Call: MergeSort(A, 1, n)

Procedure Merge

Merge(*A*, *p*, *q*, *r*)

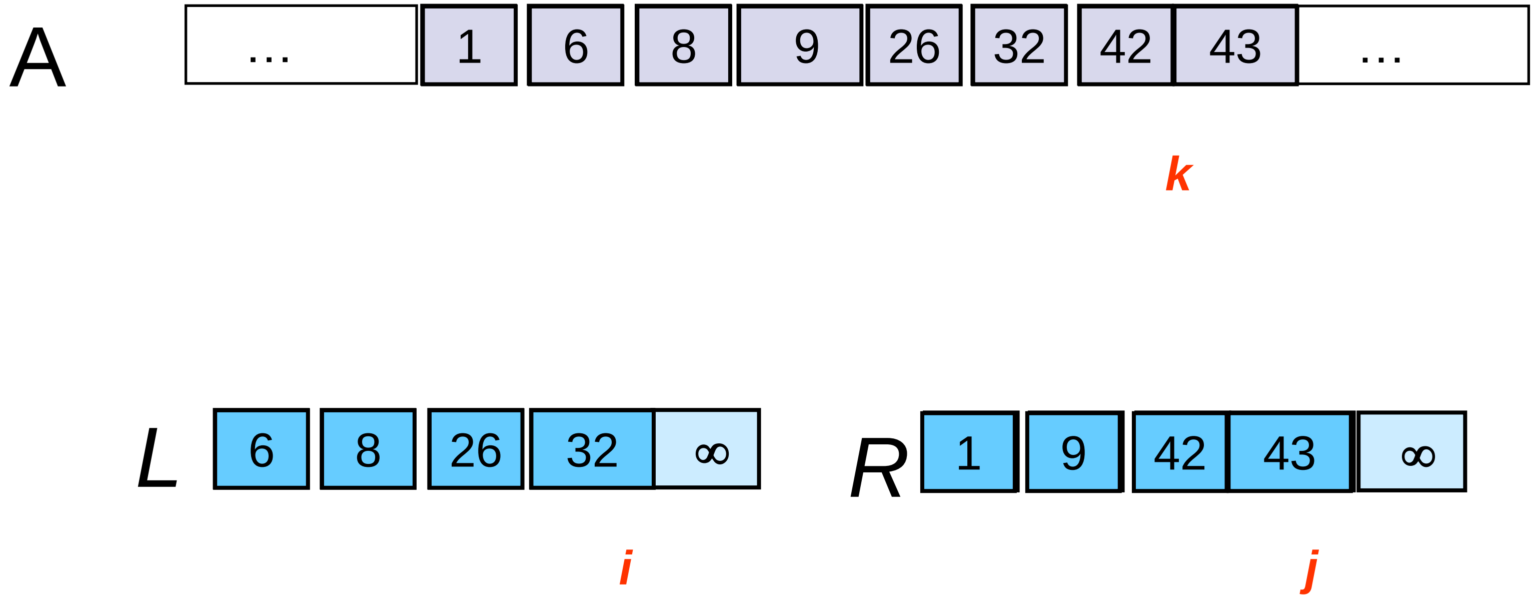
```
n_1 ← q – p + 1
n_2 ← r – q
    for i ← 1 to n_1
        do L[i] ← A[p + i – 1]
    for j ← 1 to n_2
        do R[j] ← A[q + j]
    L[n_1+1] ← ∞
    R[n_2+1] ← ∞
    i ← 1
    j ← 1
    for k ←p to r
        do if L[i] ≤ R[j]
            then A[k] ← L[i]
                 i ← i + 1
            else A[k] ← R[j]
                 j ← j + 1
```

Input: Array containing sorted subarrays *A*[*p*..*q*] and *A*[*q*+1..*r*].

Output: Merged sorted subarray in *A*[*p*..*r*].

Sentinels, to avoid having to check if either subarray is fully copied at each step.

Merge – Example



Correctness of Merge

Merge(*A*, *p*, *q*, *r*)

$n_1 \leftarrow q - p + 1$
$n_2 \leftarrow r - q$
for $i \leftarrow 1$ **to** n_1
    do $L[i] \leftarrow A[p + i - 1]$
for $j \leftarrow 1$ **to** n_2
    do $R[j] \leftarrow A[q + j]$
$L[n_1+1] \leftarrow \infty$
$R[n_2+1] \leftarrow \infty$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow p$ **to** r
    do if $L[i] \leq R[j]$
        then $A[k] \leftarrow L[i]$
            $i \leftarrow i + 1$
        else $A[k] \leftarrow R[j]$
            $j \leftarrow j + 1$

Loop Invariant for the *for* loop

At the start of each iteration of the for loop:

Subarray $A[p..k - 1]$ contains the $k - p$ smallest elements of L and R in sorted order.
$L[i]$ and $R[j]$ are the smallest elements of
L and R that have not been copied back into A.

Initialization:

Before the first iteration:

- $A[p..k - 1]$ is empty.
- $i = j = 1$.
- $L[1]$ and $R[1]$ are the smallest elements of L and R not copied to A.

Correctness of Merge

Merge(*A*, *p*, *q*, *r*)

```
1   n1 ← q – p + 1
2   n2 ← r – q
3       for i ← 1 to n1
4           do L[i] ← A[p + i – 1]
5       for j ← 1 to n2
6           do R[j] ← A[q + j]
7       L[n1+1] ← ∞
8       R[n2+1] ← ∞
9       i ← 1
10      j ← 1
11      for k ←p to r
12          do if L[i] ≤ R[j]
13              then A[k] ← L[i]
14                      i ← i + 1
15              else A[k] ← R[j]
16                      j ← j + 1
```

Maintenance:

Case 1: $L[i] \le R[j]$

- By LI, A contains $p - k$ smallest elements of L and R in sorted order.
- By LI, $L[i]$ and $R[j]$ are the smallest elements of L and R not yet copied into A.
- Line 13 results in A containing $p - k + 1$ smallest elements (again in sorted order).

Incrementing i and k reestablishes the LI for the next iteration.

Similarly for $L[i] > R[j]$.

Termination:

- On termination, $k = r + 1$.
- By LI, A contains $r - p + 1$ smallest elements of L and R in sorted order.
- L and R together contain $r - p + 3$ elements. All but the two sentinels have been copied back into A.

Analysis of Merge Sort

- Running time $T(n)$ of Merge Sort:
- Divide: computing the middle takes $\Theta(1)$
- Conquer: solving 2 subproblems takes $2T(n/2)$
- Combine: merging n elements takes $\Theta(n)$
- Total:

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

$\Rightarrow T(n) = \Theta(n \lg n)$

Recurrence Relations

Recurrence Relations

- Equation or an inequality that characterizes a function by its values on smaller inputs.
- **Solution Methods**
  - Substitution Method.
  - Recursion-tree Method.
  - Master Method.
- Recurrence relations **arise when we analyze the running time of iterative or recursive algorithms**.
  - **Ex:** Divide and Conquer.

$$T(n) = \Theta(1) \qquad \text{if } n \leq c$$
$$T(n) = a\,T(n/b) + D(n) + C(n) \qquad \text{otherwise}$$

Substitution Method

- **Guess** the form of the solution, then **use mathematical induction** to show it correct.
  - Substitute guessed answer for the function when the inductive hypothesis is applied to smaller values – hence, the name.
- Works well when the solution is easy to guess.
- No general way to guess the correct solution.

Substitution method

Guess the form of the solution

Then prove it's correct by induction

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

Halves the input then constant amount of work

Similar to binary search:

Guess: $O(\log_2 n)$

Proof?

$$T(n) = T(n/2) + d = O(\log_2 n)?$$

Ideas?

Proof?

$$T(n) = T(n/2) + d = O(\log_2 n)?$$

Proof by induction!
-Assume it's true for smaller T(k), i.e. k < n
-prove that it's then true for current T(n)

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

Assume $T(k) = O(\log_2 k)$ for all $k < n$

Show that $T(n) = O(\log_2 n)$

From our assumption, $T(n/2) = O(\log_2 n)$:

$$O(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n \text{ such that} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

From the definition of big-*O*: $T(n/2) \le c \log_2(n/2)$

How do we now prove T(n) = O(log n)?

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

To prove that $T(n) = O(\log_2 n)$ identify the appropriate constants:

$$O(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n \text{ such that} \\ 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \end{array} \right\}$$

i.e. some constant c' such that $T(n) \leq c' \log_2 n$

$$\begin{aligned} T(n) &= T(n/2) + d \\ &\leq c\log_2(n/2) + d \qquad \text{from our inductive hypothesis} \\ &\leq c\log_2 n - c\log_2 2 + d \\ &\leq c\log_2 n - c + d \qquad \text{residual} \end{aligned}$$

Key question: does a constant exist such that:
$T(n) \leq c' log_2 n$

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

To prove that $T(n) = O(\log_2 n)$ identify the appropriate constants:

$$O(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n \text{ such that} \\ 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \end{array} \right\}$$

i.e. some constant c_2 such that $T(n) \leq c_2 \log_2 n$

Key question: does a constant exist such that:

$T(n) \leq c' log_2 n$

$$\mathrm{T(n)} \leq c log_2 n \underbrace{- c + d}$$

if $c \geq d$, then, yes!

(if not, just let c' = d)

Base case?

For an inductive proof we need to show two things:

- Assuming it's true for $k < n$ show it's true for n
- *Show that it holds for some base case*

What is the base case in our situation?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \text{ is small} \\ T(n/2) + d & \text{otherwise} \end{cases}$$

$$T(n) = T(n-1) + n$$

Guess the solution?

At each iteration, does a linear amount of work (i.e. iterate over the data) and reduces the size by one at each step

$O(n^2)$

Assume $T(k) = O(k^2)$ for all $k < n$

- again, this implies that $T(n-1) \leq c(n-1)^2$

Show that $T(n) = O(n^2)$, i.e. $T(n) \leq c'n^2$

$$T(n) = T(n-1) + n$$

$$\leq c(n-1)^2 + n$$ from our inductive hypothesis

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

$$= cn^2 - 2cn + c + n$$ residual

if $-2cn + c + n \leq 0$

then let c’ = c and there exists a constant such that $T(n) \leq c'n^2$

$$
\begin{aligned}
T(n) &= T(n-1) + n \\
&\le c(n-1)^2 + n \quad \text{from our inductive hypothesis} \\
&= c(n^2 - 2n + 1) + n \\
&= cn^2 - 2cn + c + n \quad \text{residual}
\end{aligned}
$$

$$
\begin{aligned}
-2cn + c + n &\le 0 \\
-2cn + c &\le -n \\
c(-2n + 1) &\le -n \\
c &\ge \frac{n}{2n-1} \\
c &\ge \frac{1}{2 - 1/n}
\end{aligned}
$$

which holds for any $c \ge 1$ for $n \ge 1$

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

Guess the solution?

Recurses into 2 sub-problems that are half the size and performs some operation on all the elements

$O(n \log n)$

What if we guess wrong, e.g. $O(n^2)$?

Assume $T(k) = O(k^2)$ for all $k < n$

- again, this implies that $T(n/2) \leq c(n/2)^2$

Show that $T(n) = O(n^2)$

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

$$\leq 2c(n/2)^2 + n$$ from our inductive hypothesis

$$= 2cn^2/4 + n$$

$$= 1/2cn^2 + n$$

$$= cn^2 - (1/2cn^2 - n)$$ residual

if

$$-(1/2cn^2 - n) \leq 0$$

$$-1/2cn^2 + n \leq 0$$

overkill?

$$cn \geq 2$$

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

What if we guess wrong, e.g. O(*n*)?

Assume *T(k) = O(k)* for all *k < n*

- again, this implies that *T(n/2) ≤ c(n/2)*

Show that *T(n) = O(n)*

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &\leq 2cn/2 + n \\ &= cn + n \\ &\leq cn \end{aligned}$$

factor of *n* so we can just roll it in?

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

What if we guess wrong, e.g. O(*n*)?

Assume *T(k) = O(k)* for all *k < n*

- again, this implies that *T(n/2) ≤ c(n/2)*

Show that *T(n) = O(n)*

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &\leq 2cn/2 + n \\ &= cn + n \\ &\leq cn \end{aligned}$$

Must prove the exact form!

cn+n ≤ cn ??

~~factor of *n* so we can just roll it in?~~

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

Prove $T(n) = O(n \log_2 n)$

Assume $T(k) = O(k \log_2 k)$ for all $k < n$

- again, this implies that $T(k) = ck \log_2 k$

Show that $T(n) = O(n \log_2 n)$

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &\le 2cn/2 \log(n/2) + n \\ &\le cn(\log_2 n - \log_2 2) + n \\ &\le cn \log_2 n - cn + n \\ &\le cn \log_2 n \end{aligned}$$

residual

if $cn \ge n$, $c > 1$

Changing variables

$$T(n) = 2T(\sqrt{n}) + \log n$$

Guesses?

We can do a variable change: let $m = \log_2 n$
(or $n = 2^m$)

$$T(2^m) = 2T(2^{m/2}) + m$$

Now, let $S(m)=T(2^m)$

$$S(m) = 2S(m/2) + m$$

Changing variables

$$S(m) = 2S(m/2) + m$$

Guess? $S(m) = O(m \log m)$

$$T(n) = T(2^m) = S(m) = O(m \log m)$$

substituting m=log n

$$T(n) = O(\log n \log \log n)$$

Recursion-tree Method

- Making a good guess is sometimes difficult with the substitution method.
- Use **recursion trees** to devise good guesses.
- Recursion Trees
  - Show successive expansions of recurrences using trees.
  - Keep track of the time spent on the subproblems of a divide and conquer algorithm.
  - Help organize the algebraic bookkeeping necessary to solve a recurrence.

Recursion Tree – Example

- Running time of Merge Sort:

$$T(n) = \Theta(1) \qquad \text{if } n = 1$$

$$T(n) = 2T(n/2) + \Theta(n) \qquad \text{if } n > 1$$

- Rewrite the recurrence as

$$T(n) = \mathbf{c} \qquad \text{if } n = 1$$

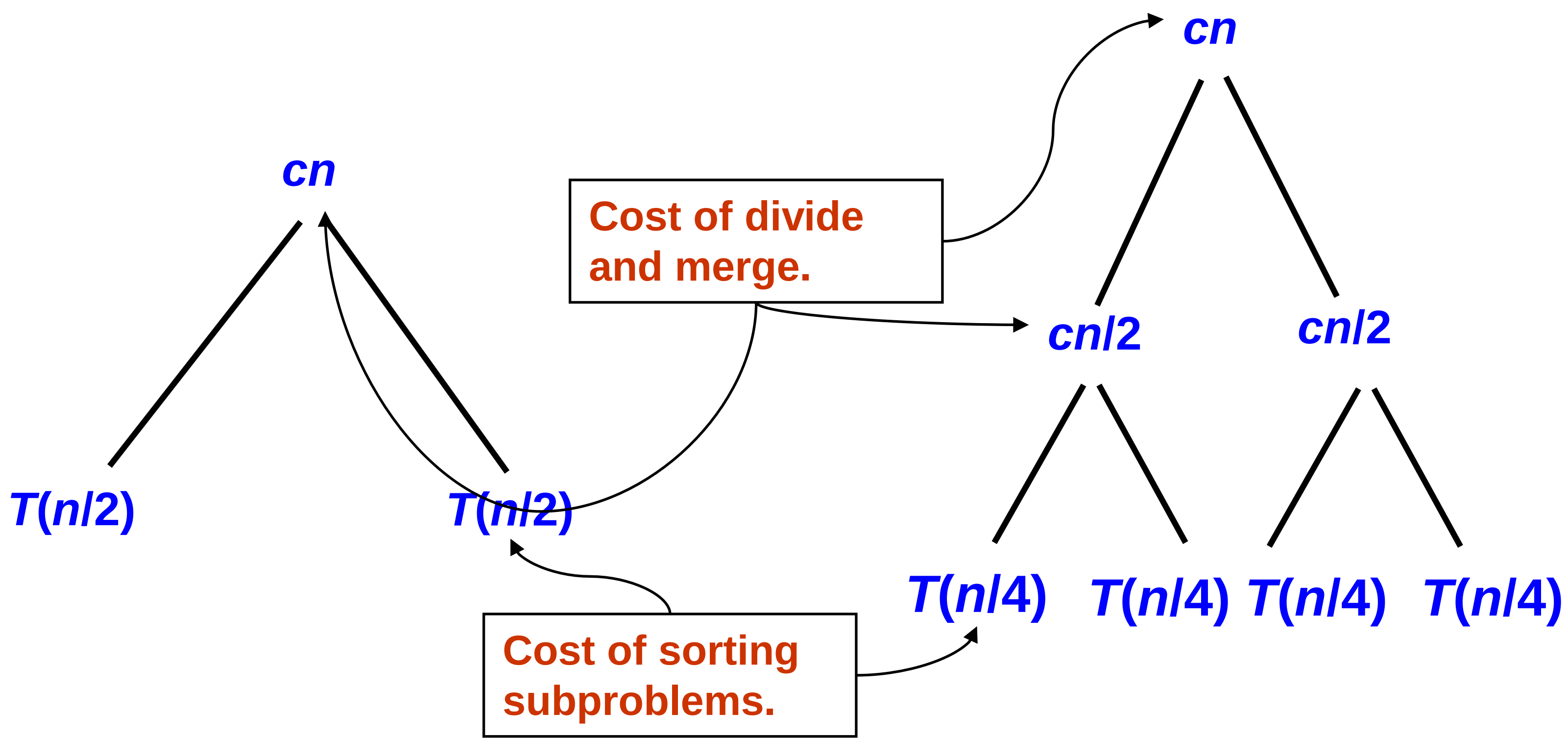
$$T(n) = 2T(n/2) + \mathbf{cn} \qquad \text{if } n > 1$$

$\mathbf{c > 0}$: Running time for the base case and time per array element for the divide and combine steps.

Recursion Tree for Merge Sort

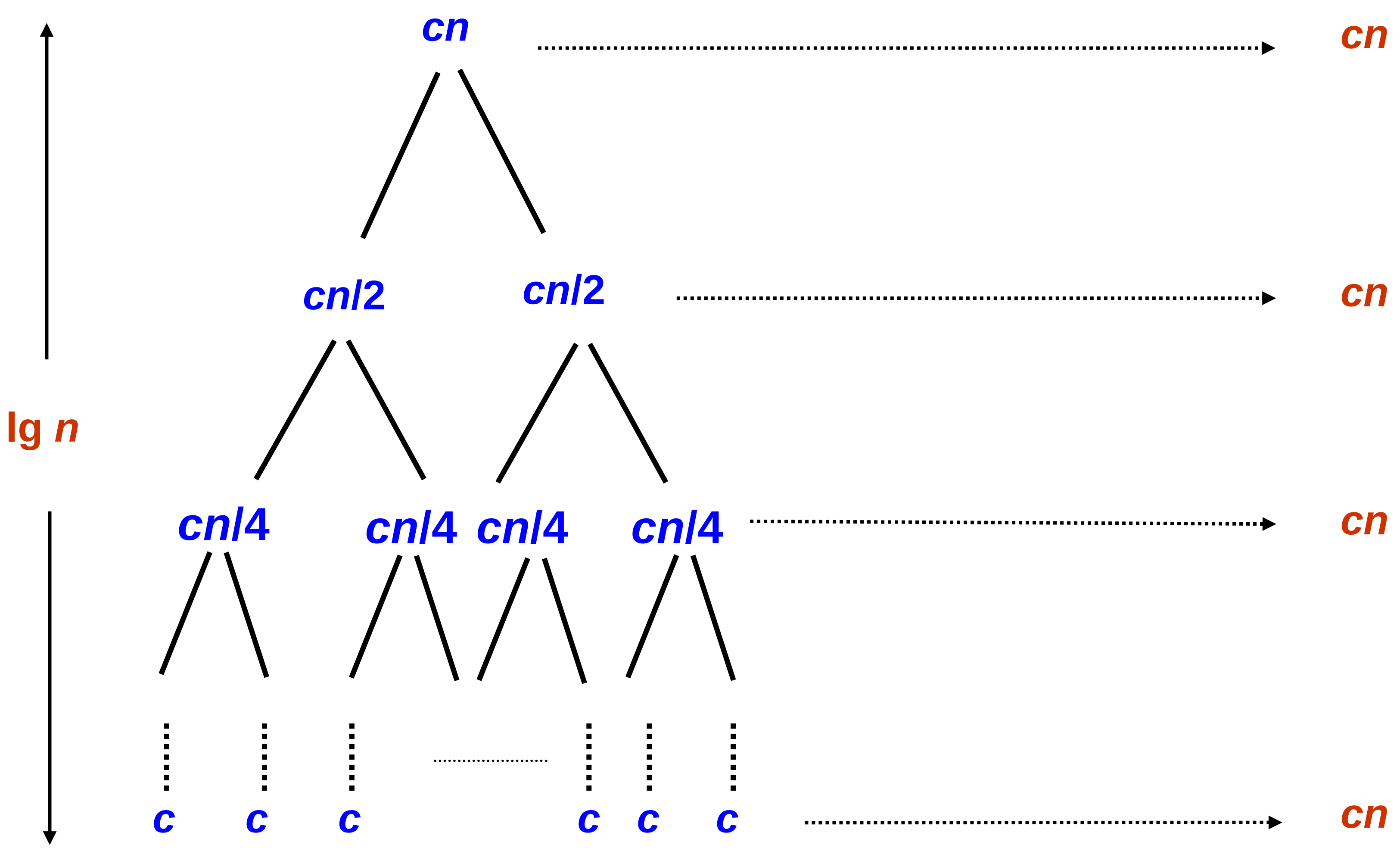
For the original problem, we have a cost of *cn*, plus two subproblems each of size (*n*/2) and running time *T*(*n*/2).

Each of the size *n*/2 problems has a cost of *cn*/2 plus two subproblems, each costing *T*(*n*/4).



Recursion Tree for Merge Sort

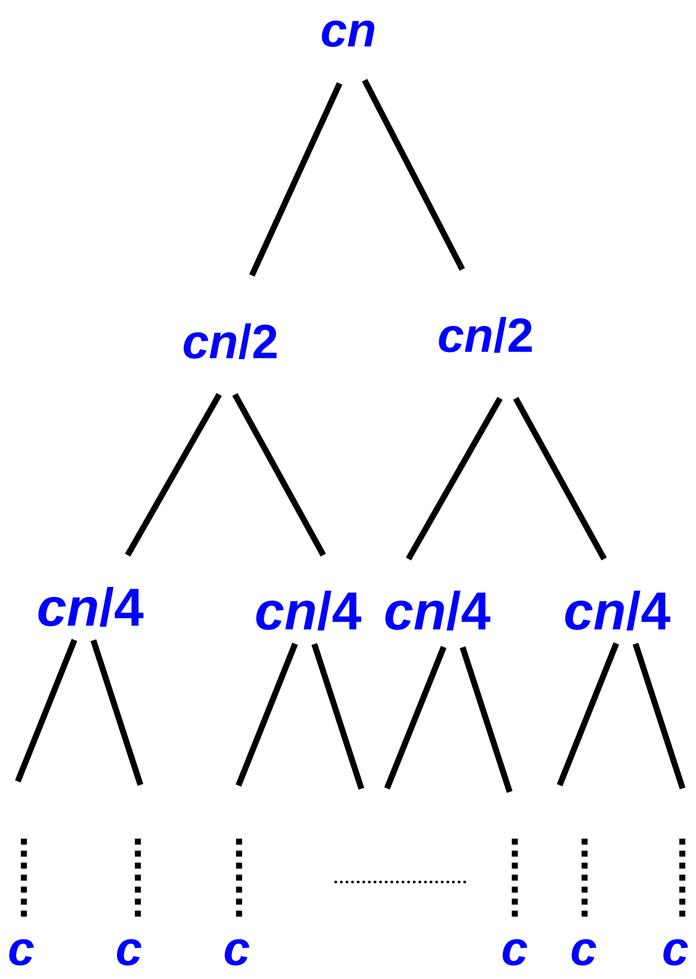
Continue expanding until the problem size reduces to 1.



Total : $cn\lg n + cn$

Recursion Tree for Merge Sort

Continue expanding until the problem size reduces to 1.



- Each level has total cost ***cn***.
- Each time we go down one level, the number of subproblems doubles, but the cost per subproblem halves $\Rightarrow$ *cost per level remains the same*.
- There are lg n + 1 levels, height is lg n. (Assuming n is a power of 2.)
- Can be proved by induction.
- Total cost = sum of costs at each level

= (lg n + 1)cn = cnlg$n$ + cn = Θ(n lg$n$).

Other Examples

- Use the recursion-tree method to determine a guess for the recurrences
  - $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$.
  - $T(n) = T(n/3) + T(2n/3) + O(n)$.

$$T(n) = 3T(n/4) + n^2$$

cost

cn^2 $\qquad$ cn^2

$T(n/4)$ $\qquad$ $T(n/4)$ $\qquad$ $T(n/4)$

$$T(n) = 3T(n/4) + n^2$$

cost

cn^2 $\qquad$ cn^2

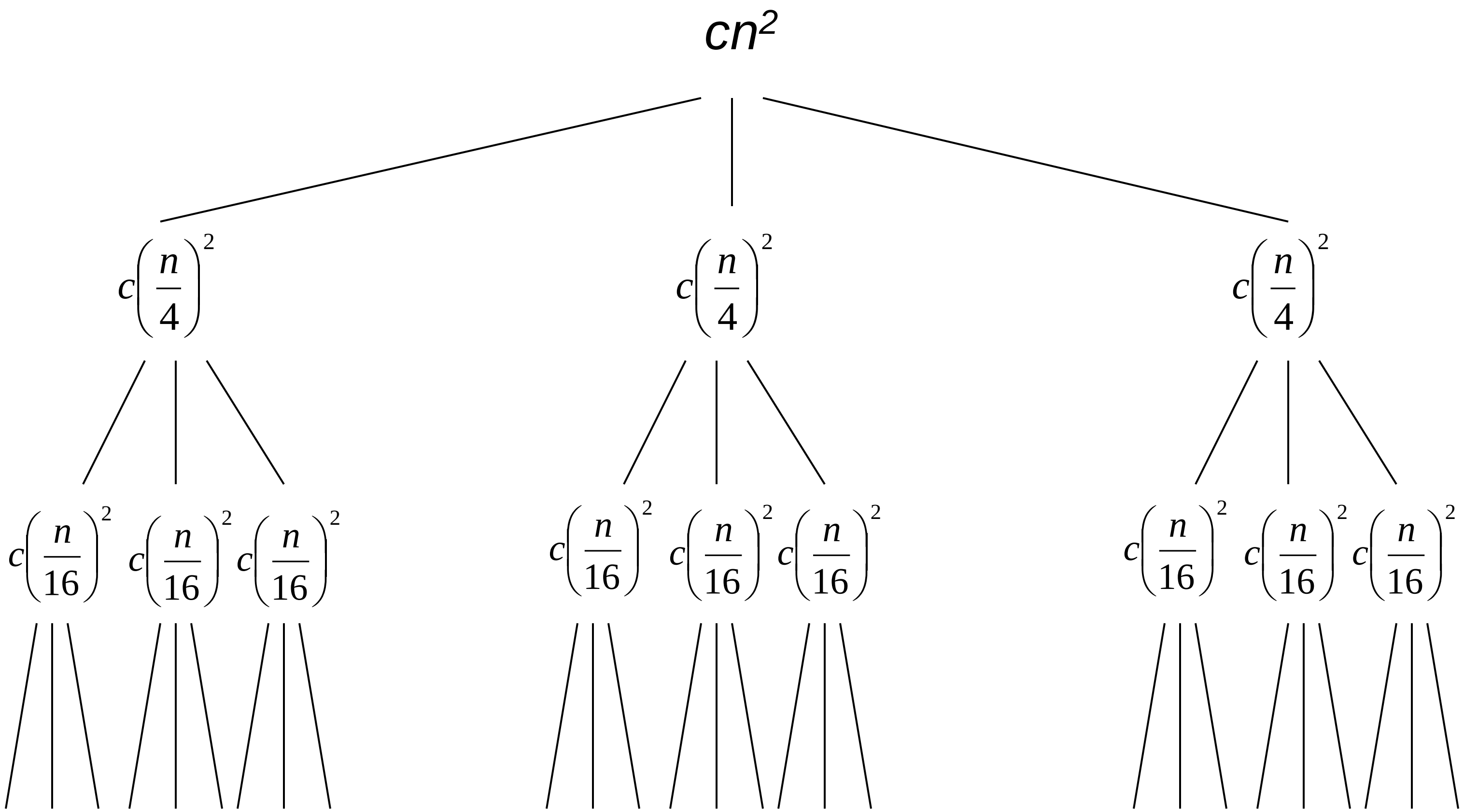
$$c\left(\frac{n}{4}\right)^2 \qquad c\left(\frac{n}{4}\right)^2 \qquad c\left(\frac{n}{4}\right)^2$$

$3/16cn^2$

$$T\left(\frac{n}{16}\right) T\left(\frac{n}{16}\right) T\left(\frac{n}{16}\right) \qquad T\left(\frac{n}{16}\right) T\left(\frac{n}{16}\right) T\left(\frac{n}{16}\right) \qquad T\left(\frac{n}{16}\right) T\left(\frac{n}{16}\right) T\left(\frac{n}{16}\right)$$

$$T(n) = 3T(n/4) + n^2$$

cost



cn^2

$3/16cn^2$

$(3/16)^2cn^2$

What is the cost at each level? $\left(\frac{3}{16}\right)^d cn^2$

What is the depth of the tree?

At each level, the size of the data is divided by 4

$$\frac{n}{4^d} = 1$$

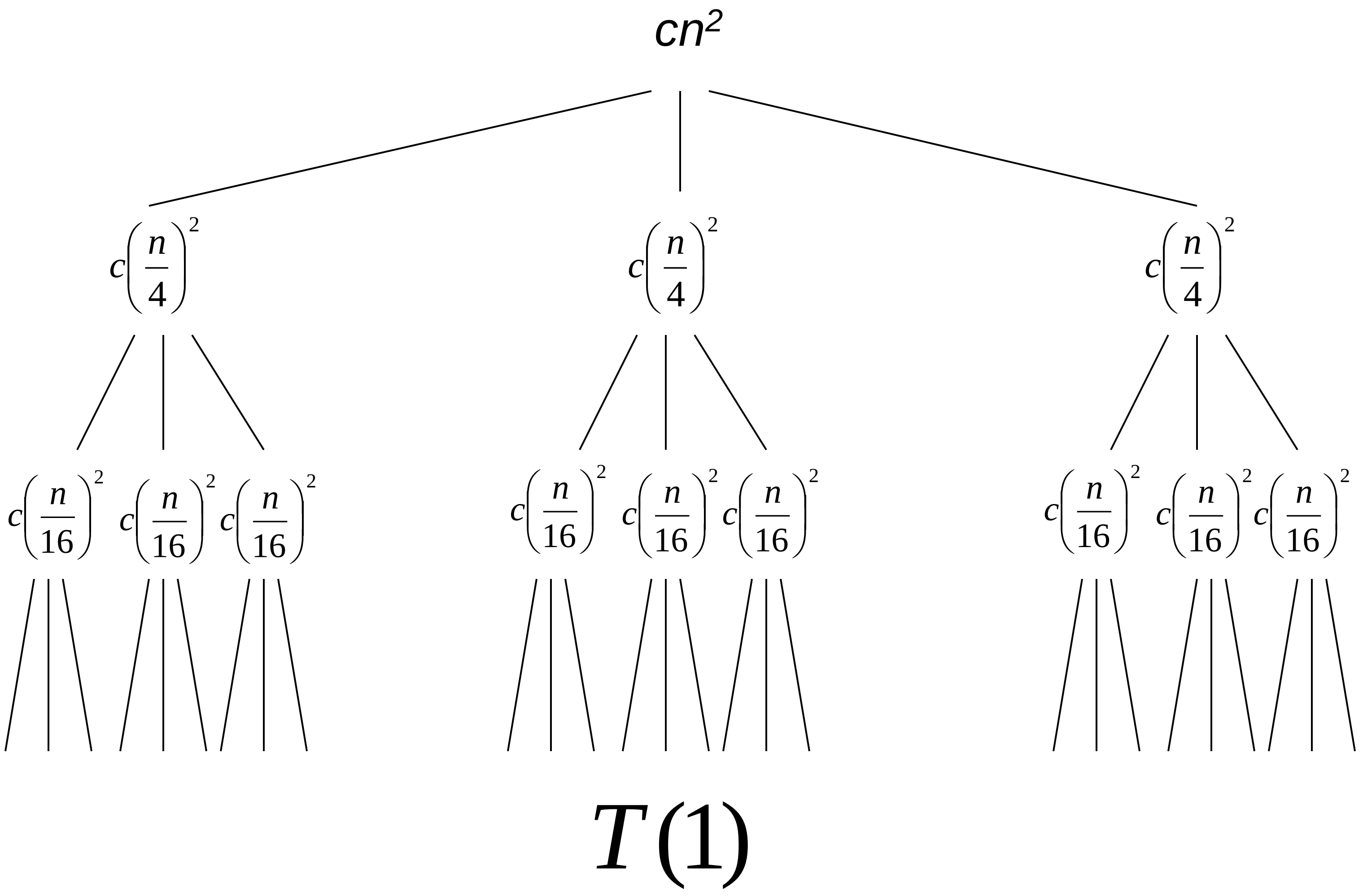
$$\log\left(\frac{n}{4^d}\right) = 0$$

$$\log n - \log 4^d = 0$$

$$d \log 4 = \log n$$

$$d = \log_4 n$$

$$T(n) = 3T(n/4) + n^2$$



How many leaves are there?

How many leaves?

How many leaves are there in a complete ternary tree of depth *d*?

$$3^d = 3^{\log_4 n}$$

Total cost

$$T(n) = cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \ldots + \left(\frac{3}{16}\right)^{d-1} cn^2 + \Theta(3^{\log_4 n})$$

$$= cn^2 \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i + \Theta(3^{\log_4 n})$$

$$< cn^2 \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i + \Theta(3^{\log_4 n})$$

$$= \frac{1}{1-(3/16)} cn^2 + \Theta(3^{\log_4 n})$$

$$= \frac{16}{13} cn^2 + \Theta(3^{\log_4 n})$$?

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

let x = 3/16

Total cost

$$T(n) = \frac{16}{13}cn^2 + \Theta(3^{\log_4 n})$$

$$3^{\log_4 n} = 4^{\log_4 3^{\log_4 n}}$$

$$= 4^{\log_4 n \log_4 3}$$

$$= 4^{\log_4 n^{\log_4 3}}$$

$$= n^{\log_4 3}$$

$$T(n) = \frac{16}{13}cn^2 + \Theta(n^{\log_4 3})$$

$$T(n) = O(n^2)$$

Verify solution using substitution

$$T(n) = 3T(n/4) + n^2$$

Assume $T(k) = O(k^2)$ for all $k < n$

Show that $T(n) = O(n^2)$

Given that $T(n/4) = O((n/4)^2)$, then

$$O(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n \text{ such that} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

$T(n/4) \le c(n/4)^2$

$$T(n) = 3T(n/4) + n^2$$

To prove that Show that $T(n) = O(n^2)$ we need to identify the appropriate constants:

$$O(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n \text{ such that} \\ 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \end{array} \right\}$$

i.e. some constant c such that $T(n) \leq cn^2$

$$T(n) = 3T(n/4) + n^2$$

$$\leq 3c(n/4)^2 + n^2$$

$$= cn^2 3/16 + n^2$$

$$= cn^2 - cn^2 * \frac{13}{16} + n^2$$

residual

a constant exists if, if $-cn^2 * \frac{13}{16} + n^2 \leq 0$

$$T(n) = 3T(n/4) + n^2$$

To prove that Show that $T(n) = O(n^2)$ we need to identify the appropriate constants:

$$O(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n \text{ such that} \\ 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \end{array} \right\}$$

i.e. some constant c such that $T(n) \leq cn^2$

$$-cn^2 * \frac{13}{16} + n^2 \leq 0$$

$$cn^2 * \frac{13}{16} \geq n^2$$

$$c \geq \frac{16}{13}$$

Recursion Trees – Caution Note

- Recursion trees only generate guesses.
  - Verify guesses using substitution method.
- A small amount of “sloppiness” can be tolerated. Why?
- If careful when drawing out a recursion tree and summing the costs, can be used as direct proof.

The Master Method

- Based on the Master theorem.
- "Cookbook" approach for solving recurrences of the form

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

  - $a \geq 1$, $b > 1$ are constants.
  - $f(n)$ is asymptotically positive.
  - n/b may not be an integer, but we ignore floors and ceilings. Why?
- Requires memorization of three cases.

The Master Theorem

Theorem

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and

Let $T(n)$ be defined on nonnegative integers by the recurrence

$T(n) = aT(n/b) + f(n)$, where we can replace n/b by $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.

$T(n)$ can be bounded asymptotically in three cases:

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$,

   and if, for some constant $c < 1$ and all sufficiently large n,

   we have $a \cdot f(n/b) \leq c\, f(n)$, then $T(n) = \Theta(f(n))$.

   (regularity condition)

The Master Theorem: Examples

- $T(n) = 16T(n/4)+n$
  - $a = 16$, $b = 4$, $n^{\log_b a} = n^{\log_4 16} = n^2$.
  - $f(n) = n = O(n^{\log_b a-\varepsilon}) = O(n^{2-\varepsilon})$, where $\varepsilon = 1$ $\Rightarrow$ **Case 1.**
  - Hence, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$.

- $T(n) = T(3n/7) + 1$
  - $a = 1$, $b=7/3$, and $n^{\log_b a} = n^{\log_{7/3} 1} = n^0 = 1$
  - $f(n) = 1 = \Theta(n^{\log_b a})$ $\Rightarrow$ **Case 2**.
  - Therefore, $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(\lg n)$

The Master Theorem: Examples

- $T(n) = 3T(n/4) + n \lg n$
  - $a = 3$, $b=4$, thus $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$
  - $f(n) = n \lg n = \Omega(n^{\log_4 3 + \varepsilon})$ where $\varepsilon \approx 0.2$
  - Is $3n/4 \lg 3n/4 \le c\, n \lg n$ *for* $c<1$ *? – It holds for* $c=3/4$ $\Rightarrow$ **Case 3**.
  - Therefore, $T(n) = \Theta(f(n)) = \Theta(n \lg n)$.

- $T(n) = 2T(n/2) + n \lg n$
  - $a = 2$, $b=2$, $f(n) = n \lg n$, and $n^{\log_b a} = n^{\log_2 2} = n$
  - ***f*(*n*)** is asymptotically larger than $n^{\log_b a}$, but **not polynomially larger**. The ratio $\lg n$ is asymptotically less than n^{ε} for any positive ε. Thus, the Master Theorem ***doesn't*** apply here.

$$T(n) = T(n/2) + 2^n$$

if $f(n) = O(n^{\log_b a - \varepsilon})$ for $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$

if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$

if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for $\varepsilon > 0$ and $af(n/b) \le cf(n)$ for $c < 1$

then $T(n) = \Theta(f(n))$

a = 1
b = 2
f(n) = *2ⁿ*

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

is $2^n = O(n^{0-\varepsilon})$?

is $2^n = \Theta(n^0)$?

is $2^n = \Omega(n^{0+\varepsilon})$?

Case 3?

is $2^{n/2} \le c2^n$ for $c < 1$?

$$T(n) = T(n/2) + 2^n$$

$$\text{if } f(n) = O(n^{\log_b a - \varepsilon}) \text{ for } \varepsilon > 0, \text{ then } T(n) = \Theta(n^{\log_b a})$$

$$\text{if } f(n) = \Theta(n^{\log_b a}), \text{ then } T(n) = \Theta(n^{\log_b a} \log n)$$

$$\text{if } f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ for } \varepsilon > 0 \text{ and } af(n/b) \le cf(n) \text{ for } c < 1$$

$$\text{then } T(n) = \Theta(f(n))$$

$$\text{is } 2^{n/2} \le c2^n \text{ for } c < 1?$$

Let c = 1/2

$$2^{n/2} \le (1/2)2^n$$

$$2^{n/2} \le 2^{-1}2^n$$

$$2^{n/2} \le 2^{n-1}$$

$T(n) = \Theta(2^n)$

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

$$\text{if } f(n) = O(n^{\log_b a - \varepsilon}) \text{ for } \varepsilon > 0, \text{ then } T(n) = \Theta(n^{\log_b a})$$

$$\text{if } f(n) = \Theta(n^{\log_b a}), \text{ then } T(n) = \Theta(n^{\log_b a} \log n)$$

$$\text{if } f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ for } \varepsilon > 0 \text{ and } af(n/b) \le cf(n) \text{ for } c < 1$$

$$\text{then } T(n) = \Theta(f(n))$$

a = 2

b = 2

f(n) = *n*

$$n^{\log_b a} = n^{\log_2 2} = n^1$$

$$\text{is } n = O(n^{1-\varepsilon})?$$

$$\text{is } n = \Theta(n^1)?$$

$$\text{is } n = \Omega(n^{1+\varepsilon})?$$

Case 2: Θ(*n* log *n*)

$$T(n) = 16T(n/4) + n!$$

$$\text{if } f(n) = O(n^{\log_b a - \varepsilon}) \text{ for } \varepsilon > 0\text{, then } T(n) = \Theta(n^{\log_b a})$$

$$\text{if } f(n) = \Theta(n^{\log_b a})\text{, then } T(n) = \Theta(n^{\log_b a} \log n)$$

$$\text{if } f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ for } \varepsilon > 0 \text{ and } af(n/b) \le cf(n) \text{ for } c < 1$$
$$\text{then } T(n) = \Theta(f(n))$$

a = 16

b = 4

f(n) = *n!*

$$n^{\log_b a} = n^{\log_4 16} = n^2$$

$$\text{is } n! = O(n^{2-\varepsilon})?$$

$$\text{is } n! = \Theta(n^2)?$$

$$\text{is } n! = \Omega(n^{2+\varepsilon})?$$

Case 3?

$$\text{is } 16(n/4)! \le cn! \text{ for } c < 1?$$

$$T(n) = 16T(n/4) + n!$$

$$\text{if } f(n) = O(n^{\log_b a - \varepsilon}) \text{ for } \varepsilon > 0, \text{ then } T(n) = \Theta(n^{\log_b a})$$

$$\text{if } f(n) = \Theta(n^{\log_b a}), \text{ then } T(n) = \Theta(n^{\log_b a} \log n)$$

$$\text{if } f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ for } \varepsilon > 0 \text{ and } af(n/b) \le cf(n) \text{ for } c < 1$$

$$\text{then } T(n) = \Theta(f(n))$$

$$\text{is } 16(n/4)! \le cn! \text{ for } c < 1?$$

Let c = 1/2

$$cn! = 1/2n!$$

$$> (n/2)!$$

therefore,

$$16(n/4)! \le (n/2)! < 1/2n!$$

T(n) = Θ($n!$)

$$T(n) = \sqrt{2}T(n/2) + \log n$$

if $f(n) = O(n^{\log_b a - \varepsilon})$ for $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$

if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$

if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for $\varepsilon > 0$ and $af(n/b) \le cf(n)$ for $c < 1$

then $T(n) = \Theta(f(n))$

a = $\sqrt{2}$

b = 2

f(*n*) = log*n*

$$n^{\log_b a} = n^{\log_2 \sqrt{2}} = n^{\log_2 2^{1/2}} = \sqrt{n}$$

is $\log n = O(n^{1/2-\varepsilon})$?

is $\log n = \Theta(n^{1/2})$?

is $\log n = \Omega(n^{1/2+\varepsilon})$?

Case 1: $\Theta(\sqrt{n})$

$$T(n) = 4T(n/2) + n$$

$$\text{if } f(n) = O(n^{\log_b a - \varepsilon}) \text{ for } \varepsilon > 0, \text{ then } T(n) = \Theta(n^{\log_b a})$$

$$\text{if } f(n) = \Theta(n^{\log_b a}), \text{ then } T(n) = \Theta(n^{\log_b a} \log n)$$

$$\text{if } f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ for } \varepsilon > 0 \text{ and } af(n/b) \le cf(n) \text{ for } c < 1$$
$$\text{then } T(n) = \Theta(f(n))$$

a = 4
b = 2
f(n) = *n*

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

$$\text{is } n = O(n^{2-\varepsilon})?$$

$$\text{is } n = \Theta(n^2)?$$

$$\text{is } n = \Omega(n^{2+\varepsilon})?$$

Case 1: $\Theta(n^2)$

Recurrences

$$T(n) = 2T(n/3) + d \qquad T(n) = 7T(n/7) + n$$

$$\text{if } f(n) = O(n^{\log_b a - \varepsilon}) \text{ for } \varepsilon > 0, \text{ then } T(n) = \Theta(n^{\log_b a})$$

$$\text{if } f(n) = \Theta(n^{\log_b a}), \text{ then } T(n) = \Theta(n^{\log_b a} \log n)$$

$$\text{if } f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ for } \varepsilon > 0 \text{ and } af(n/b) \le cf(n) \text{ for } c < 1$$

$$\text{then } T(n) = \Theta(f(n))$$

$$T(n) = T(n-1) + \log n \qquad T(n) = 8T(n/2) + n^3$$

Acknowledgements

- Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C., Introduction to algorithms. MIT press, 2009
- Dr. David Kauchak, Pomona College
- Prof. David Plaisted, The University of North Carolina at Chapel Hill