Course Name: Analysis and Design of Algorithms

Topics to be covered

- Recurrences

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.

$T(n) = \Theta(1)$ if $n \leq c$

$T(n) = a\, T(n/b) + D(n) + C(n)$ 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.

Example – Exact Function

Recurrence: $T(n) = 1$ if $n = 1$

$T(n) = 2T(n/2) + n$ if $n > 1$

- **Guess:** $T(n) = n \lg n + n$.
- **Induction:**
  - **Basis:** $n = 1 \Rightarrow n \lg n + n = 1 = T(n)$.
  - **Hypothesis:** $T(k) = k \lg k + k$ for all $k < n$.
  - **Inductive Step:** $T(n) = 2\,T(n/2) + n$

$$= 2\,((n/2)\lg(n/2) + (n/2)) + n$$

$$= n\,(\lg(n/2)) + 2n$$

$$= n \lg n - n + 2n$$

$$= n \lg n + n$$

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

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

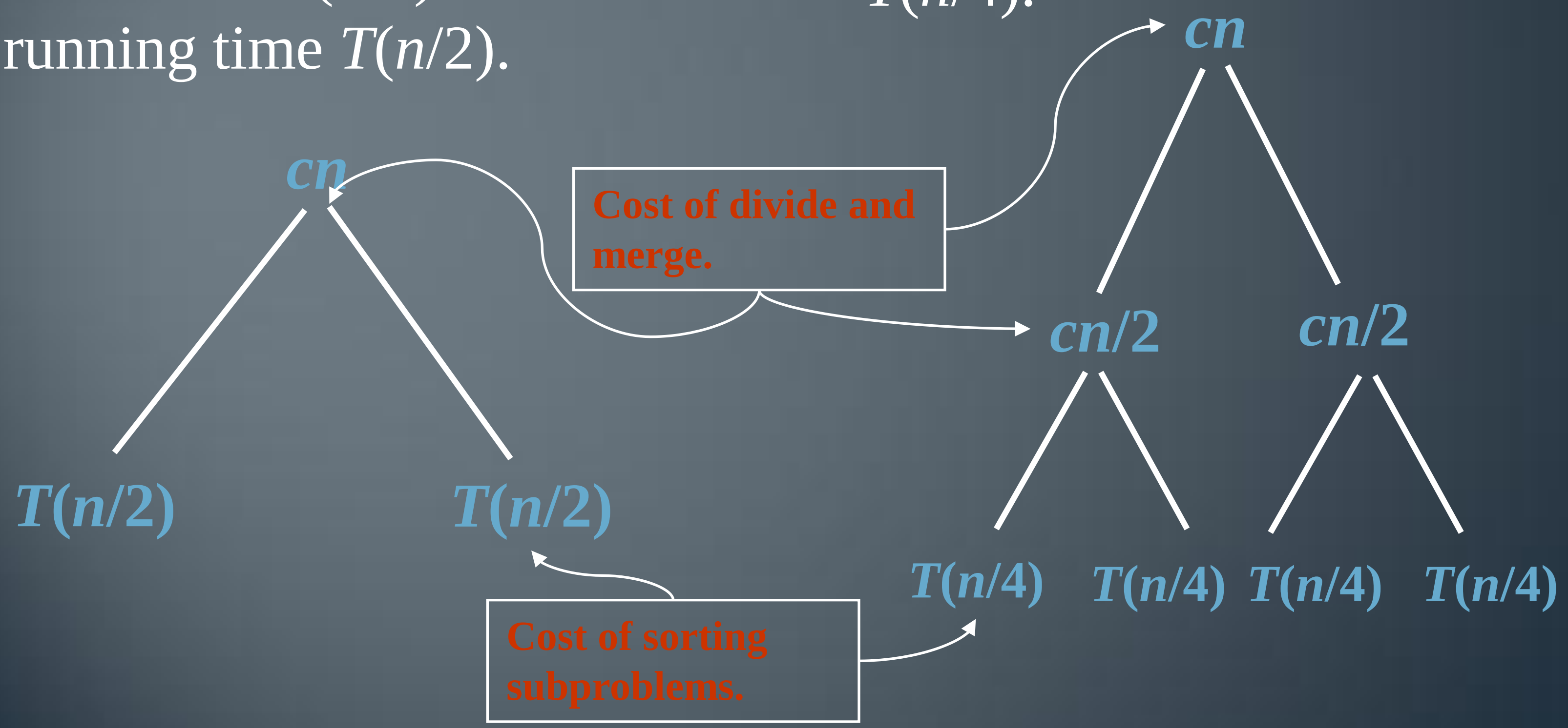
$$T(n) = 2T(n/2) + \boldsymbol{cn} \qquad \text{if } n > 1$$

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

Recursion Tree for Merge Sort

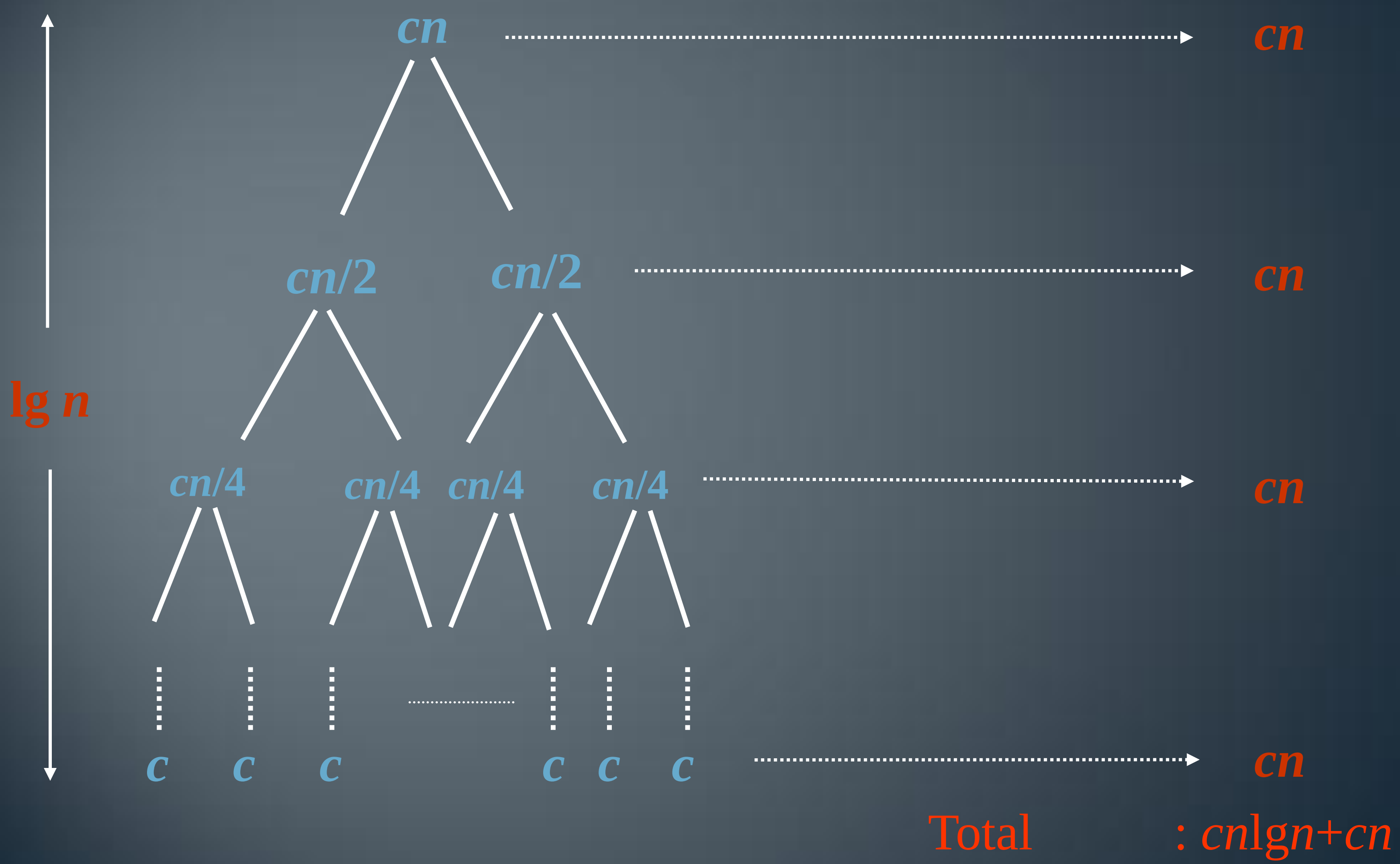
For the original problem, we have a cost of cn, plus two subproblems each of size $(n/2)$ and running time $T(n/2)$.

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



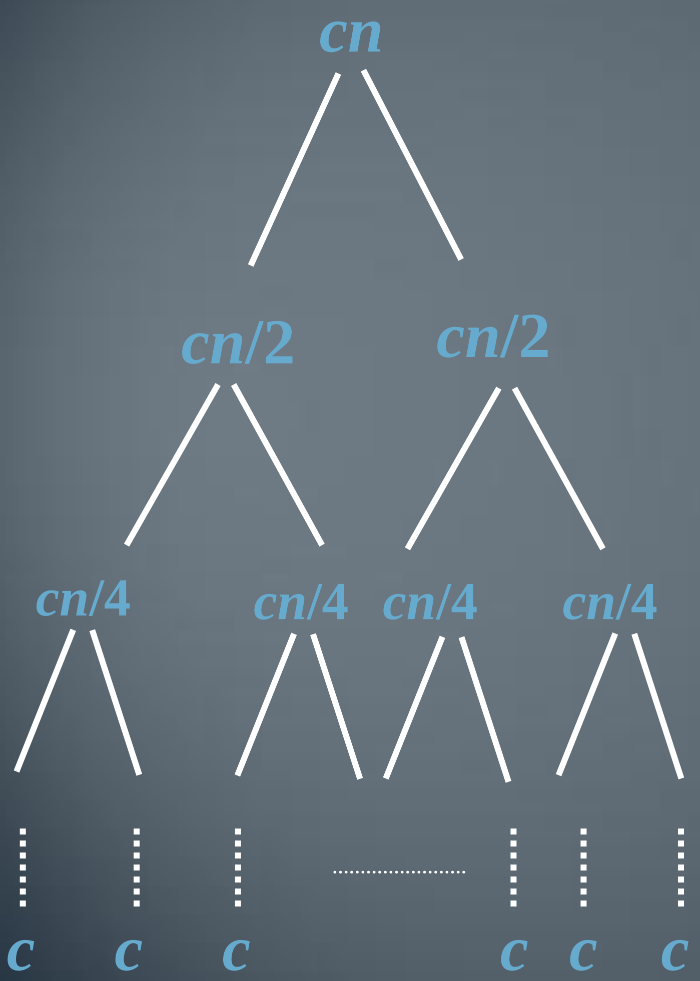
Recursion Tree for Merge Sort

Continue expanding until the problem size reduces to 1.



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 = $(\lg n + 1)cn = cn\lg n + cn = \Theta(n \lg n)$.

Other Examples

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

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 4.1

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

RECURRENCE RELATIONS EXAMPLE

EXAMPLE 1: <u>QUICK SORT</u>

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

$T(1)= O(1)$

- In the above case the presence of function of T on both sides of the equation signifies the presence of recurrence relation
- (*SUBSTITUTION MEATHOD used*) The equations are simplified to produce the final result:

……cntd

Cntd….

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

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

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

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

$$= 2^3\ T(n/2^3) + \underline{n + n + n}$$

$$= \boldsymbol{n \log n}$$

Cntd…

EXAMPLE 2: BINARY SEARCH

T(n)=O(1) + T(n/2)

T(1)=1

Above is another example of recurrence relation and the way to solve it is by Substitution.

$T(n)=T(n/2) +1$

$= T(n/2^2)+1+1$

$= T(n/2^3)+1+1+1$

$= \log n$

T(n)= O(logn)