



CS 590: Algorithm

Week 4: Divide and Conquer and Computation Complexity 2

In Suk Jang
Department of Computer Science
Stevens Institute of Technology



Outline

- 4.1. Lecture 3 Review
- 4.2. Recursion Analysis Solutions
 - 4.2.1. Substitution Method
 - 4.2.2. Recursion Tree
 - 4.2.3. Master's Theorem
- 4.3. Divide and Conquer
 - 4.3.1. Maximum-Sum Array
 - 4.3.2. Strassen's Theorem



4.1. Lecture 3 Review

4.2. Recursion Analysis Solutions

4.2.1. Substitution Method

4.2.2. Recursion Tree

4.2.3. Master's Theorem

4.3. Divide and Conquer

4.3.1. Maximum-Sum Array

4.3.2. Strassen's Theorem

4.1. Lecture 3 Review



- **Asymptotical Expressions**
 - Notations - O , Θ , Ω , o , ω
 - Use to analyze and express the running time behavior, $T(n)$, of algorithms
 - Do not provide the exact running time value.
 - However, provide the general idea.
- **Insertion Sort**
 - Iteratively locate the key to its appropriate position.
 - Best case: $\Theta(n)$, Average and Worst cases: $\Theta(n^2)$.
- **Merge Sort**
 - Divide-and-conquer based recursion approach.
 - Best, Average, and Worst cases: $\Theta(n \lg n)$.



4.1. Lecture 3 Review

4.2. Recursion Analysis Solutions

4.2.1. Substitution Method

4.2.2. Recursion Tree

4.2.3. Master's Theorem

4.3. Divide and Conquer

4.3.1. Maximum-Sum Array

4.3.2. Strassen's Theorem

4.2. Recursion Analysis Solutions



Recall the divide-and-conquer technique used for the merge-sort algorithm. We

1. **Divide** the problem into a number of subproblems that are smaller instances of same problem.
2. **Conquer** the subproblem by use of recursion. Small enough or trivial subproblems (**base case**) are solved in a straightforward manner.
3. **Combine** the subproblems solutions into the solution for the original problem.



4.2. Recursion Analysis Solutions

- We use the recurrences to characterize the running time of a divide-and-conquer algorithm.
 - Base case – when the sub-arrays cannot be split further.
 - Recurrence running time is characterized as
 - $T(n) = aT\left(\frac{n}{b}\right) + D(n) + C(n)$
 - $aT\left(\frac{n}{b}\right)$ is characterized by the running time of each subarray $T\left(\frac{n}{b}\right)$ where $\frac{1}{b}$ is the ratio of split and the number of subarray problems a .
- Example: The running of MERGE-SORT is described by

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

Split into 2 subarrays and had 2 subproblems
 $a = 2, b = 2$

4.2. Recursion Analysis Solutions



How would the divide-and-conquer running time equation formulate if subproblems are divided into unequal size?

- $T(n) = a_1 T\left(\frac{n}{b_1}\right) + a_2 T\left(\frac{n}{b_2}\right) + D(n) + C(n)$
- Suppose the subproblems have a 2/3-to-1/3 split and the divide and combine steps take linear time
 - $T(n) = T\left(\frac{2n}{3}\right) + T\left(\frac{n}{3}\right) + \Theta(n).$

What if subproblems are not necessarily constrained to being a constant fraction of the original problem size?

- Suppose one of subproblems contains only one element fewer than the original problem.
 - Each recursive call would take constant time + the time for the recursive calls it makes.
 - $T(n) = T(n - 1) + \Theta(n).$

4.2. Recursion Analysis Solutions



There are three methods to solve recurrences by obtaining asymptotic Θ or O bound on the solution.

- **Substitution Method** – **Guess** a bound and use mathematical induction **to prove** it.
- **Recursion-tree Method** – Convert the recurrence **into a tree** whose nodes represent the costs incurred at various levels of the recursion then sum the recurrences.
- **Master Method** – **Provides bounds for recurrences.**
 - Mainly used for divide-and-conquer algorithms.

4.2.1. Substitution Method



The substitution method solves recurrences in two procedures:

1. Guess the form of the solution.
 2. Use mathematical induction to find the constants and show that the solution works.
- First, we substitute the guessed solution for function when applying the inductive hypothesis to smaller values.
 - If proving the initial guessing is failed, we guess the form of the answer and re-test.
 - Repeat until we find the best answer.
 - Keep in mind that the substitution method can establish either upper or lower bounds on a recurrence.



4.2.1. Substitution Method

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

- A guessing function for $T\left(\frac{n}{2}\right)$: $T(k) = k \lg k + k \ \forall k < n$

- Use induction to verify our guess:

- Base case: $n = 1 \Rightarrow k \lg k + k = 1 \cdot \cancel{\lg 1} + 1 = 1$

$$\log_2 1 = 0$$

- Inductive step: substitute $\frac{n}{2}$ into k

$$T(n) = 2T\left(\frac{n}{2}\right) + n = 2(k \lg k + k) + n$$

$$= 2\left(\frac{n}{2} \lg \frac{n}{2} + \frac{n}{2}\right) + n = n \lg \frac{n}{2} + n + n$$

$$\log\left(\frac{a}{b}\right) = \log a - \log b$$

$$= n(\lg n - \cancel{\lg 2}) + 2n = n \lg n + n$$

$$\log_2 2 = 1$$

\therefore our guess is correct.



4.2.1. Substitution Method

When we use have the asymptotic notation, we

- Re-express the function using a constant term.
- Then show the satisfaction of upper and lower boundaries separately.
 - Note that the use of constants might differ in both cases.
- Example: $T(n) = 2T(n/2) + \Theta(n)$
 - For the upper bound,

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

- Using the constant term

$$T(n) \leq 2T\left(\frac{n}{2}\right) + cn$$

for constant $c > 0$.



4.2.1. Substitution Method

- Let the guessing function be $T(n) \leq dk \lg k$ for constant $d > 0$.
- Since c is a part of recurrence, d might depend on c .
- We define the condition of d can be expressed in terms of c .

$$T(n) \leq 2T\left(\frac{n}{2}\right) + cn = 2T(dk \lg k) + cn$$

Substitute $\frac{n}{2}$ into $k \forall k < n$.

$$= 2\left(d \left\lceil \frac{n}{2} \lg \frac{n}{2} \right\rceil\right) + cn$$

$$\uparrow d \left\lceil n \lg \frac{n}{2} \right\rceil = dn(\lg n - \lg 2)$$

- We want $-dn + cn \leq 0$!
- so $dn \lg n - dn + cn \leq dn \lg n$
- $\rightarrow d \geq c$.

$$\begin{aligned} &= dn \lg n - dn + cn \\ &\leq dn \lg n \text{ i. f. f. } d \geq c \\ &= O(n \lg n) \end{aligned}$$

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

4.2.1. Substitution Method



For the lower bound:

- We guess $T(n) \geq dk \lg k$ for constant $d > 0$.
- Substitution process:

$$\begin{aligned} T(n) &\geq 2T\left(\frac{n}{2}\right) + \Omega(n) = 2T\left(\frac{n}{2}\right) + cn \\ &= 2\left(d\frac{n}{2}\lg\frac{n}{2}\right) + cn \\ &= dn \lg n - dn + cn \\ &\geq dn \lg n \text{ i. f. f. } -dn + cn \geq 0 \\ &\geq dn \lg n = \Omega(n \lg n) \end{aligned}$$

$$\therefore T(n) = \Omega(n \lg n)$$




4.2.1. Substitution Method

Example: $T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$.

- Upper bound: $T(n) \leq 8T\left(\frac{n}{2}\right) + O(n^2) = 8T\left(\frac{n}{2}\right) + cn^2$
- Guessing Function: $T(k) \leq dk^3$

Even though $c, d > 0$,
the condition of d in terms c is
not determinable.

$$\begin{aligned} T(n) &\leq 8d \left(\frac{n}{2}\right)^3 + cn^2 \\ &= 8d \left(\frac{n^3}{8}\right) + cn^2 = dn^3 + cn^2 \\ &\not\leq dn^3 \end{aligned}$$


- **Need to make a new guessing function.**
 - \Rightarrow Subtract the lower-order terms off from our previous guess.
 - \Rightarrow How about $T(k) \leq dk^3 - d'k^2$?



4.2.1. Substitution Method

Let the new guess be $T(k) \leq dk^3 - d'k^2$ where $d, d' > 0$.

$$T(n) \leq 8 \left(d \left(\frac{n}{2} \right)^3 - d' \left(\frac{n}{2} \right)^2 \right) + cn^2$$

$$= 8d \left(\frac{n^3}{8} \right) - 8d' \left(\frac{n^2}{4} \right) + cn^2$$

Make $2d'n^2 = -d'n^2 - d'n^2$

$$= dn^3 - 2d'n^2 + cn^2$$

$$= dn^3 - d'n^2 - d'n^2 + cn^2$$

$$\underbrace{(-d' + c)n^2}_{d' \geq c} \leq 0$$

$$\leq dn^3 - d'n^2 = O(dn^3 - d'n^2) \text{ i.f.f } d' \geq c.$$



4.2.1. Substitution Method

- Be careful when using asymptotic notation.
- Consider a recurrence function $T(n) = 4T\left(\frac{n}{4}\right) + n$ and let the guessing function be $T(k) = O(k)$.

$$\begin{aligned}T(n) &= 4T\left(\frac{n}{4}\right) + n \leq 4\left(c\left(\frac{n}{4}\right)\right) + n \\&= cn + n = (c + 1)n = dn \\ \therefore T(n) &= O(n) \text{ if } d = c + 1 > 0\end{aligned}$$

- Is this acceptable? Why or why not?
- How can we prove that $T(n) \leq dn$?
 - Is c in the recurrence?

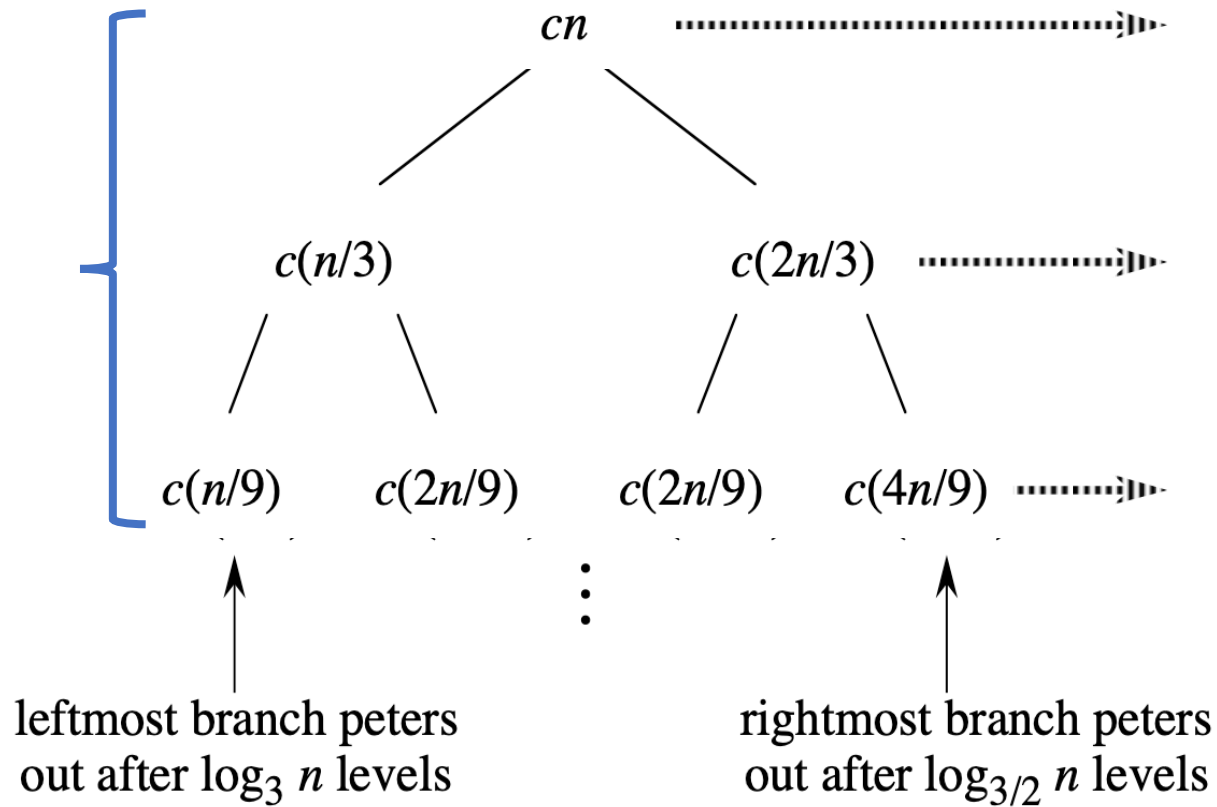
- The constant c or d does not describe the original function but a constant in the guessing function.
- The condition on any constants in the guessing function must be defined.



4.2.2. Recursion Tree

- We often use a recursion tree to **generate** or **derive** our guess function.
- Then, we **verify** our guess by using the substitution method.
- Consider an example: $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + \Theta(n)$
 - Upper bound: $T(n) \leq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$
 - Lower bound: $T(n) \geq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$
- How can we draw the recursion tree for $T(n)$?

4.2.2. Recursion Tree



- Each level contributes $\leq cn$.
- $T(n) = cn \times \lg n = \Theta(n \lg n)$?
- What is the actual height?
 - $h(T) = \log_{\frac{3}{2}} n$
 - So, $T(n) = \Theta\left(n \log_{\frac{3}{2}} n\right)$?
 - Not $O\left(n \log_{\frac{3}{2}} n\right)$?



4.2.2. Recursion Tree

- The lower bound can be guessed from the most left of the tree.
- Lower bound guess: $\geq dn \log_3 n = \Omega(n \log_3 n)$ for $d > 0$.
- Considering the lower and upper boundaries,
 - $T_L(n) = \Omega(n \log_3 n)$ & $T_U(n) = O\left(n \log_{\frac{3}{2}} n\right)$
 - Let a guessing function be $n \lg n$ to cover the both boundaries:
 - $T_L = T_U = \Theta(n \lg n)$.

4.2.2. Recursion Tree

Upper bound guess: $T(k) \leq dk \lg k$.

$$T(n) \leq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$$

$$= d\left(\frac{n}{3}\right) \lg\left(\frac{n}{3}\right) + d\left(\frac{2n}{3}\right) \lg\left(\frac{2n}{3}\right) + cn$$

$$= \frac{2dn}{3}$$

$$= \left(d\left(\frac{n}{3}\right) (\lg n - \lg 3) \right) + \left(d\left(\frac{2n}{3}\right) (\lg 2 + \lg n - \lg 3) \right) + cn$$

$$= \frac{dn}{3} \lg n + \frac{2dn}{3} \lg n = dn \lg n$$

$$-\frac{dn}{3} \lg 3 - \frac{2dn}{3} \lg 3 = -dn \lg 3$$

$$= dn \lg n - dn \left(\lg 3 - \frac{2}{3} \right) + cn$$

$$\leq dn \lg n \quad \text{i. f. f. } d \geq c \left(\lg 3 - \frac{2}{3} \right)^{-1}$$

$$-dn \left(\lg 3 - \frac{2}{3} \right) + cn \leq 0$$

$$\rightarrow d \geq c \left(\lg 3 - \frac{2}{3} \right)^{-1}$$

$$\therefore T(n) = O(n \lg n)$$



4.2.2. Recursion Tree

Lower boundary guess: $T(k) \leq dk \lg k$.

$$\begin{aligned} T(n) &\geq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + \Omega(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn \\ &= d\left(\frac{n}{3}\right) \lg\left(\frac{n}{3}\right) + d\left(\frac{2n}{3}\right) \lg\left(\frac{2n}{3}\right) + cn \\ &= \left(d\left(\frac{n}{3}\right) (\lg n - \lg 3)\right) + \left(d\left(\frac{2n}{3}\right) (\lg 2 + \lg n - \lg 3)\right) + cn \\ &= dn \lg n - \underbrace{dn \left(\lg 3 - \frac{2}{3}\right)}_{\text{purple bracket}} + cn \\ &\leq dn \lg n \quad \text{i.f.f. } 0 < d \leq c \left(\lg 3 - \frac{2}{3}\right)^{-1} \\ &\quad \therefore T(n) = \Omega(n \lg n) \end{aligned}$$

$$\begin{aligned} -dn \left(\lg 3 - \frac{2}{3}\right) + cn &\geq 0 \\ \rightarrow 0 < d &\leq c \left(\lg 3 - \frac{2}{3}\right)^{-1} \end{aligned}$$

4.2.2. Recursion Tree

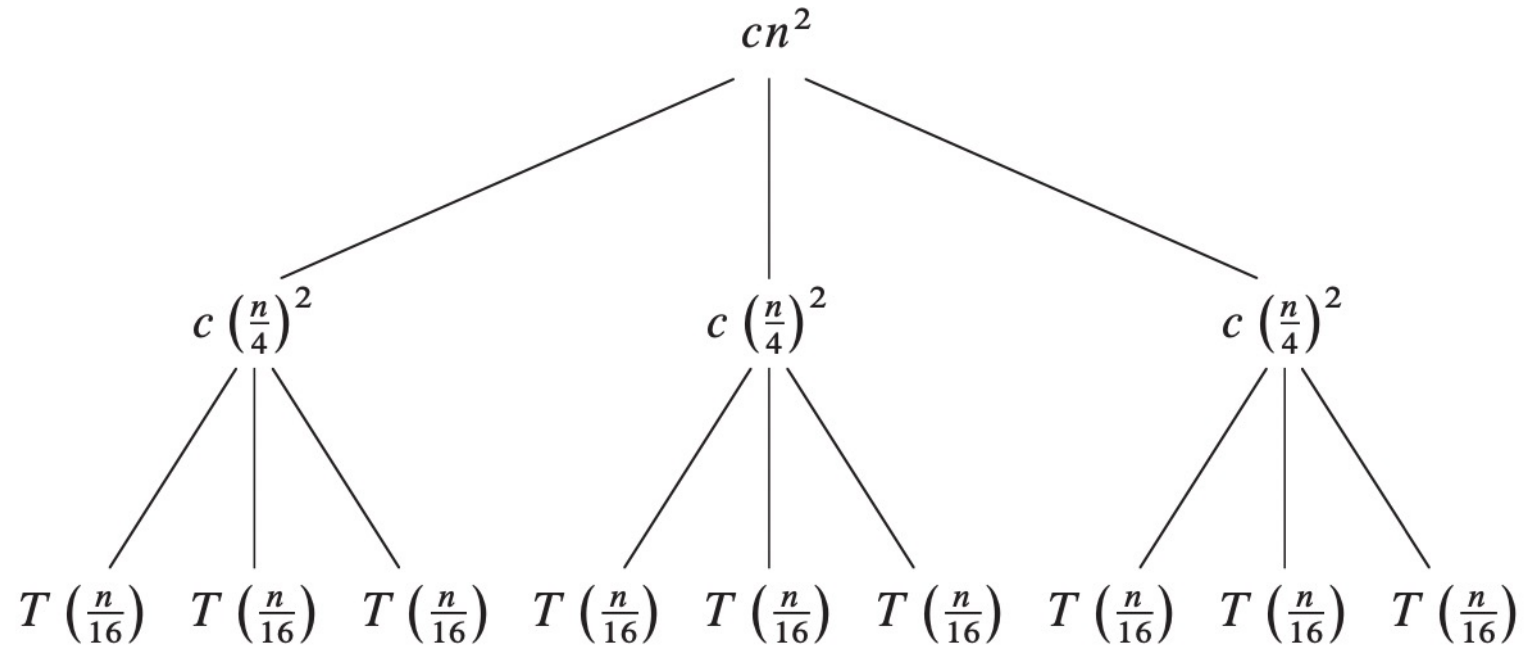


- The lower and upper boundary conditions are verified and confirmed.
- Therefore,

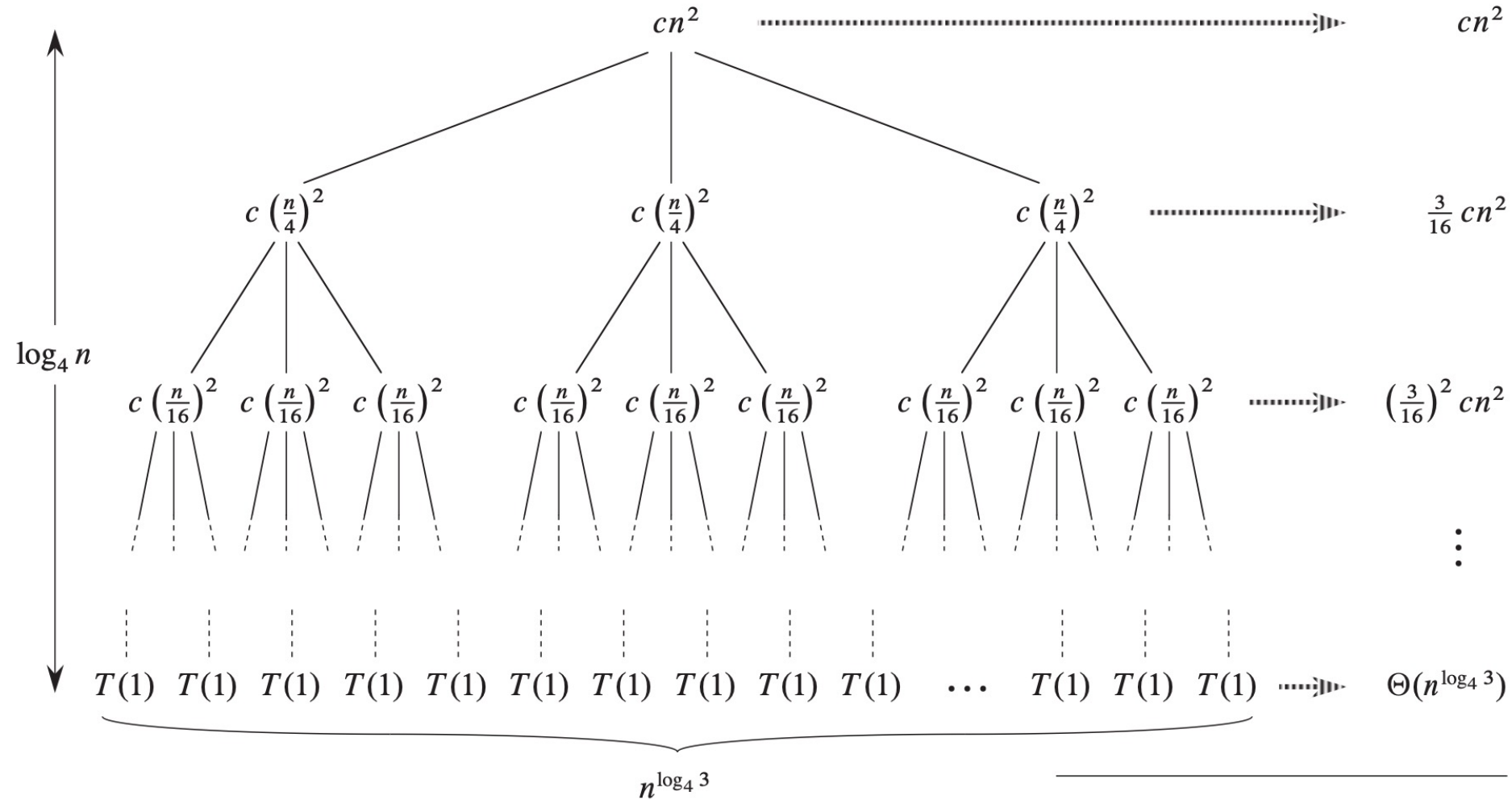
$$T(n) = \Omega(n \lg n) = O(n \lg n) = \Theta(n \lg n).$$

4.2.2. Recursion Tree

Consider a recurrence as $T(n) = 3T\left(\frac{n}{4}\right) + \Theta(n^2)$.



4.2.2. Recursion Tree



(d)

Total: $O(n^2)$

4.2.2. Recursion Tree

- For each depth i , the cost is $T_i(n) = \left(\frac{3}{16}\right)^i cn^2$.
- The expansion of the total cost is then

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

$$\sum_{i=0}^N x^i = \frac{x^N - 1}{x - 1}$$

$$\begin{aligned} &= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) < \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\ &= \frac{1}{1 - (3/16)} cn^2 + \Theta(n^{\log_4 3}) = \frac{16}{3} cn^2 + \Theta(n^{\log_4 3}) \\ &= \Theta(n^2) \end{aligned}$$

$2 > \log_4 3$

4.2.2. Recursion Tree

$$T(n) \leq 3 \left(d \frac{n^2}{4^2} \right) + cn^2 = \frac{3d}{16} n^2 + cn^2 \quad T(n) \geq 3 \left(d \frac{n^2}{4^2} \right) + cn^2 = \frac{3d}{16} n^2 + cn^2$$

$$\leq dn^2 \text{ i. f. f. } d \geq \frac{16}{3}$$

$$\geq dn^2 \text{ i. f. f. } 0 < d \leq \frac{16}{3}$$



4.2.3. Master's Methods

Used for divide-and-conquer algorithm analysis:

- The divide-and-conquer will provide bounds for recurrences of the form of
 - $T(n) = aT\left(\frac{n}{b}\right) + f(n)$ where $f(n)$ is a given function.
 - $a, b, f(n) > 0$ and a & b are constants (they are called regularities).
 - n/b to mean either $\left\lfloor \frac{n}{b} \right\rfloor$ or $\left\lceil \frac{n}{b} \right\rceil$.
- The solutions via the master's methods will be constructed based on the three cases:
 1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
 2. If $f(n) = \Theta(n^{\log_b a} \lg^k n)$, then $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$ for $k \geq 0$.
 3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n , then $T(n) = \Theta(f(n))$.



4.2.3. Master's Methods

1st case:

- MT1 says if $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
 - This means $f(n)$ is **polynomially smaller** than $n^{\log_b a}$.
 - Therefore, $f(n)$ must be asymptotically smaller than $n^{\log_b a}$.

Example: $T(n) = 5T\left(\frac{n}{2}\right) + \Theta(n^2)$

- $a = 5, b = 2, f(n) = n^2$.
- $f(n) = n^2 = O(n^{\log_b a - \epsilon}) = O(n^{\log_2 5 - \epsilon})$
- The comparison of $n^{\log_2 5 - \epsilon}$ vs. n^2 shows that $\epsilon > 0$.
 - $\lg 5 - \epsilon = \lg 4 \rightarrow \epsilon > 0$.
- $T(n) = \Theta(n^{\lg 5}) + \Theta\left(O(n^{\lg 5 - \epsilon})\right) = \Theta(n^{\lg 5})$

4.2.3. Master's Methods



2nd Case:

MT2. If $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.

- This means $f(n)$ is within a poly-log (a polynomial in the logarithm of n) factor of $n^{\log_b a}$ but not smaller.
- The cost is $n^{\log_b a} \lg^k n$ for each of the $\Theta(\lg n)$ levels and translates to $\Theta(n^{\log_b a} \lg^{k+1} n)$.

Example: $T(n) = 27T\left(\frac{n}{3}\right) + \Theta(n^3 \lg n)$

$$a = 27, b = 3, \log_3 27 = 3$$

$$\bullet \quad f(n) = \Theta(n^3 \lg n) = \Theta(n^{\log_3 27} \lg^1 n)$$

$$k = 1$$

$$\bullet \quad T(n) = \Theta(n^3) + \Theta(n^3 \lg^2 n) = \Theta(n^3 \lg^2 n).$$



4.2.3. Master's Methods

3rd Case:

If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n , then $T(n) = \Theta(f(n))$.

- This means that $f(n)$ is **polynomially greater** than $n^{\log_b a}$.
- The cost is dominated **by the root**. It translates to $T(n) = \Theta(f(n))$.
- Note: MT3 regularity (a, b) condition is generally not a problem. It always holds whenever $f(n) = n^k$ and $f(n) = \Omega(n^{\log_b a + \epsilon})$ for constant $\epsilon > 0$.



4.2.3. Master's Methods

3rd Case:

Example: $T(n) = 5T\left(\frac{n}{2}\right) + \Theta(n^3)$

1. Need to show $\epsilon > 0$ when $n^{\log_b a + \epsilon} \geq f(n)$.
2. Need to show $af\left(\frac{n}{b}\right) < cf(n)$ for $c < 1$.

- $n^{\log_2 5 + \epsilon} \geq n^3 \rightarrow \lg 5 + \epsilon \geq 3 \therefore \epsilon > 0$
- $af\left(\frac{n}{b}\right) = 5\left(\frac{n}{2}\right)^3 = \frac{5n^3}{8}$
- $cf(n) = cn^3$
 - $\frac{5n^3}{8} \leq cn^3 \rightarrow c \geq \frac{5}{8} < 1$.
- $\therefore T(n) = \Theta(n^{\lg 5}) + \Theta(n^3) = \Theta(n^3)$

4.2.3. Master's Methods



- $T(n) = 2T\left(\frac{n}{2}\right) + n^{0.5}$
- $T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n}$
- $T(n) = 4T\left(\frac{n}{2}\right) + n^2\sqrt{n}$



4.2.3. Master's Methods

$$T(n) = 2T\left(\frac{n}{2}\right) + n^{0.5}$$

- MT1: $n^{\log_2 2}$ is polynomially bigger than $n^{0.5}$.
- $n^{1-\epsilon} \geq n^{0.5} \Rightarrow 0 < \epsilon \leq 0.5$
- $T(n) = \Theta(n) + \Theta\left(O(n^{0.5})\right) = \Theta(n)$

4.2.3. Master's Methods



$$T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n}$$

- $n^{\log_4 2 - \epsilon} = n^{\log_4 2 + \epsilon} = n^{0.5} \Rightarrow MT2$
- $\Theta(n^{0.5}) = \Theta(n^{0.5} \lg^0 n) \Rightarrow k \geq 0.$
- $T(n) = \Theta(n^{0.5}) + \Theta(n^{0.5} \lg^1 n) = \Theta(n^{0.5} \lg n)$

4.2.3. Master's Methods



$$T(n) = 4T\left(\frac{n}{2}\right) + n^2\sqrt{n}$$

- $n^2 \leq n^{2.5} \Rightarrow MT3$
- $n^{2+\epsilon} \leq n^{2.5} \Rightarrow \epsilon \geq 0.5.$
- $4\left(\frac{n}{2}\right)^{2.5} \leq c(n^{2.5})$
 - $\frac{4}{2^{2.5}} = 2^{-0.5} \leq c < 1.$
- $T(n) = \Theta(n^2) + \Theta(n^{2.5}) = \Theta(n^{2.5}).$



4.2.3. Master's Methods

- Note: There is a gap between MT1 and MT2, when $f(n) < n^{\log_b a}$ but **not polynomially smaller** (a similar gap exist between MT2 and MT3).
- The regularity conditions will be failed.
- Then, the master method cannot solve the recurrence.

Example: $T(n) = 27 \left(\frac{n}{3} \right) + \Theta \left(\frac{n^3}{\lg n} \right)$

- n^3 vs. $\frac{n^3}{\lg n} = n^3 \lg^{-1} n \neq \Theta(n^3 \lg^k n)$ for any $k \geq 0$.

Example: $T(n) = 27T \left(\frac{n}{3} \right) - n^3$



4.1. Lecture 3 Review

4.2. Recursion Analysis Solutions

4.2.1. Substitution Method

4.2.2. Recursion Tree

4.2.3. Master's Theorem

4.3. Divide and Conquer

4.3.1. Maximum-Sum Array

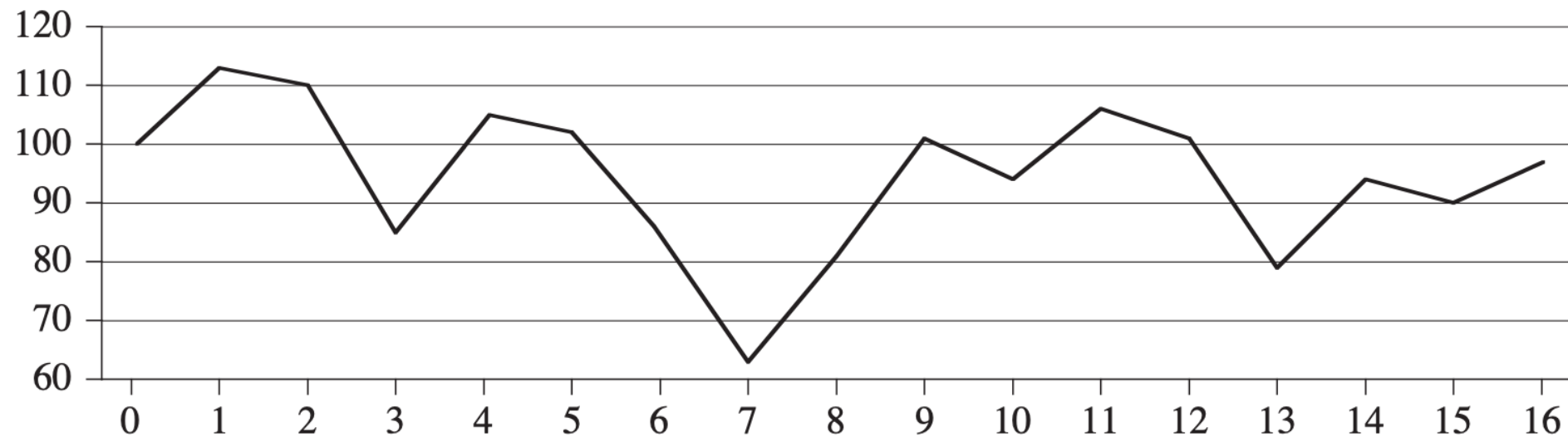
4.3.2. Strassen's Theorem



4.3.1. Maximum-subarray Problem

Consider a stock trade.

- We want to buy at the lowest and sell at the highest for the optimized profit.
- How can we find when it would occur and the maximum profit?
- Any cases of having the lowest price after the highest price?



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7



4.3.1. Maximum-subarray Problem

- The most naïve approach is pairing all possible buying and selling dates.
- Suppose there are n -many days.
 - The pairs of dates can be expressed as $\binom{n}{2}$.
 - $\binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2} = \Theta(n^2)$.
 - If each pair of dates is in constant time duration, this problem approaches $\Omega(n^2)$ time.



4.3.1. Maximum-subarray Problem

- Alternatively, we can consider the daily changes instead.
- Let $\Delta p_i = p_i - p_{i-1}$ where i is the current i -th day from the starting date.
- If we treat the change as an array, we need to find the nonempty, contiguous subarray whose values have the largest sum.
- Let A be an array with changes for 16 days, we will have 15 elements of Δp_i .
 $A = [13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]$
 - The maximum subarray of $A(18, 20, -7, 12)$.
- How can we solve the maximum-subarray problem using the divide-and-conquer technique?



4.3.1. Maximum-subarray Problem

- Suppose we want to find a maximum subarray of the subarray $A[low \dots high]$.
- Find the midpoint, mid , of the subarray and subarrays are $A[low \dots mid]$ and $A[mid + 1 \dots high]$.
- Any contiguous subarray $A[i \dots j]$ of $A[low \dots high]$ must lie in exactly one of the followings:
 - entirely in the left subarray of $A[low \dots mid]$ so $low \leq i \leq j \leq mid$
 - entirely in the right subarray of $A[mid + 1 \dots high]$ so $mid < i \leq j \leq high$
 - crossing the midpoint, $low \leq i \leq mid < j \leq high$

4.3.1. Maximum-subarray Problem



- We must choose a subarray that cross the midpoint.
- Two subarrays: $A[i \dots mid]$ & $A[mid + 1 \dots j]$
- Find maximum subarray in two subarrays and combine them.
 - A function takes as input the array A and the indices $low, mid, \& high$.
 - it returns a tuple containing the indices demarcating a maximum subarray that crosses the midpoint along with the sum of the values in the maximum subarray.



4.3.1. Maximum-subarray Problem

Find-Max-Crossing-Subarray(*A*,*low*,*mid*,*high*)

```
1  leftsum =  $-\infty$ 
2  sum = 0
3  For i = mid downto low
4      sum = sum + A[i]
5      If sum > leftsum
6          leftsum = sum
7          maxleft = i
8  rightsum =  $-\infty$ 
9  sum = 0
10 For j = mid + 1 to high
11     sum = sum + A[j]
12     If sum > rightsum
13         rightsum = sum
14         maxright = j
15 return maxleft, maxright, leftsum + rightsum
```

- L1,2,8,9, and 15 are constants.
- The total iterations in for-loops are n -many iterations.
 - All instructions are constants.
 - $\therefore T(n) = \Theta(n)$.

4.3.1. Maximum-subarray Problem

[13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]



Before the for-loop of i,

- leftsum = $-\infty$ and sum = 0.
- So the first leftsum and sum can be updated with the A[mid].
- A[mid] does not change before the for-loop.

i=8: A[8]=18, sum = 18, leftsum = 18, maxleft = 18

i=7: A[7]=-23, sum = -5, leftsum = 18, maxleft = 18

i=6: A[6]=-16, sum = -21, leftsum = 18, maxleft = 18

i=5: A[5]=-3, sum = -24, leftsum = 18, maxleft = 18

i=4: A[4]=20, sum = -4, leftsum = 18, maxleft = 18

i=3: A[3]=-25, sum = -29, leftsum = 18, maxleft = 18

i=2: A[2]=-3, sum = -32, leftsum = 18, maxleft = 18

i=1: A[1]=13, sum = -19, leftsum = 18, maxleft = 18

i=0: Terminates, sum = -19, leftsum = 18, maxleft = 18



4.3.1. Maximum-subarray Problem

[13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]

j=9: A[9]=20, sum = 20, rightsum = 20, maxright = 9

j=10: A[10]=-7, sum = 13, rightsum = 20, maxright = 9

j=11: A[11]=12, sum = 26, rightsum = 26, maxright = 10

j=12: A[12]=-5, sum = 21, rightsum = 26, maxright = 10

j=13: A[13]=-22, sum = -1, rightsum = 26, maxright = 10

j=14: A[14]=15, sum = 14, rightsum = 26, maxright = 10

j=15: A[15]=-4, sum = 10, rightsum = 26, maxright = 10

j=16: A[16]=7, sum = 17, rightsum = 26, maxright = 10

```
8   For  $j = mid + 1$  to  $high$ 
9        $sum = sum + A[j]$ 
10      If  $sum > rightsum$ 
11           $rightsum = sum$ 
12           $maxright = j$ 
13  return  $maxleft, maxright, leftsum + rightsum$ 
```

4.3.1. Maximum-subarray Problem



FIND-MAXIMUM-SUBARRAY(*A*,*low*,*high*)

```
1  if high == low
2      return (low, high, A[low])  //base case: only one element
3  else mid =  $\lfloor (\textit{low} + \textit{high}) / 2 \rfloor$ 
4      (leftlow, lefthigh, leftsum) = FIND-MAXIMUM-SUBARRAY(A,low,mid)
5      (rightlow, righthigh, rightsum) = FIND-MAXIMUM-SUBARRAY(A,mid+1,high)
6      (crosslow, crosshigh, crosssum) = FIND-MAX-CROSSING-SUBARRAY(A,low,mid,high)
7      if leftsum  $\geq$  rightsum & leftsum  $\geq$  crosssum
8          return(leftlow, lefthigh, leftsum)
9      elseif rightsum  $\geq$  leftsum & rightsum  $\geq$  crosssum
10         return(rightlow, righthigh, rightsum)
11     else return(crosslow, crosshigh, crosssum)
```



4.3.1. Maximum-subarray Problem

- When $n = 1$, Line 2 takes constant time so $T(1) = \Theta(1)$
- When $n > 1$, Line 1 and 3 takes constant time.
- Line 4 & 5: A subarray of $n/2$ elements so $T(n/2)$
- Line 6: Recalling FIND-MAX-CROSSING-SUBARRAY takes $\Theta(n)$.
- Line 7-end: Takes constant time $\Theta(1)$.
- To this end,

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



4.3.1. Maximum-subarray Problem

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

- $n^{\lg 2} = n = n \lg^0 n$.
- This is MT case 2.
- $T(n) = \Theta(n \lg n)$.

4.3.2. Strassen's algorithm for matrix multiplication



If $A = (a_{ij})$ and $B = (b_{ij})$ are square $n \times n$ matrices, then the product $C = A \cdot B$ by

$$c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$$

- We must compute n^2 matrix and each is the sum of n values.
- We assume that each matrix has an attribute rows, giving the number of rows in the matrix.

4.3.2. Strassen's algorithm for matrix multiplication



SQUARE-MATRIX-MULTIPLY(A,B)

```
1   $n = A.rows$ 
2  Let C be an new  $n \times n$  matrix
3  for  $i = 1$  to  $n$ 
4      for  $j = 1$  to  $n$ 
5           $c_{ij} = 0$ 
6          for  $k = 1$  to  $n$ 
7               $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ 
8  return C
```

- We have 3 loops that runs exactly n iterations and each execution of line 7 takes constant time.
- This means it takes $\Theta(n^3)$.



4.3.2. Strassen's algorithm for matrix multiplication

- How to use divide-and-conquer method?
- Assume that n is an exact power of 2 in each of the $n \times n$ matrices.
- Then we can divide the matrices into four $\frac{n}{2} \times \frac{n}{2}$ matrices.
- As long as $n \geq 2$, the dimension $n/2$ is an integer.



4.3.2. Strassen's algorithm for matrix multiplication

SQUARE-MATRIX-MULTIPLY-RECURSIVE: SQMR(A,B)

```
1   $n = A.rows$ 
2  Let C be an new  $n \times n$  matrix
3  if  $i == 1$ 
4       $c_{11} = a_{11} \cdot b_{11}$ 
5  else partition A,B, and C
6       $C_{11} = SQMR(A_{11}, B_{11}) + SQMR(A_{12}, B_{21})$ 
7       $C_{12} = SQMR(A_{11}, B_{12}) + SQMR(A_{12}, B_{22})$ 
8       $C_{21} = SQMR(A_{21}, B_{11}) + SQMR(A_{22}, B_{21})$ 
9       $C_{22} = SQMR(A_{21}, B_{12}) + SQMR(A_{22}, B_{22})$ 
10 return C
```

- L5: we can partition the matrices without copying entries by using index calculations.
- We identify a submatrix by a range of row indices and a range of column indices of the original matrix. Then it takes $\Theta(1)$ time.
- If we copy entries, then we would spend $\Theta(n^2)$.



4.3.2. Strassen's algorithm for matrix multiplication

The total time for the recursive case is the sum of the partitioning time, the time for all the recursive calls, and the time add the matrices resulting from the recursive calls.

$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1 \\ 8T(n/2) + \Theta(n^2), & \text{if } n > 1 \end{cases}$$

Using MT1, the solution $T(n) = \Theta(n^3)$.

Thus this approach is no faster than the straightforward procedure.



4.3.2. Strassen's algorithm for matrix multiplication

Steps:

1. Divide the input matrices A and B , and output matrix C into $\frac{n}{2} \times \frac{n}{2}$ submatrices. It takes $\Theta(1)$ time.
2. Create 10 matrices $S = S_1, S_2, \dots, S_{10}$, each of which is $\frac{n}{2} \times \frac{n}{2}$ and is the sum or difference of two matrices created in step 1. This process takes $\Theta(n^2)$ time.
3. Recursively compute seven matrix products P_1, P_2, \dots, P_7 . Each matrix P_i is $\frac{n}{2} \times \frac{n}{2}$.
4. Compute the desired submatrices $C_{11}, C_{12}, C_{21}, C_{22}$ of the result matrix C by adding and subtracting various combinations of the P_i matrices. It takes $\Theta(n^2)$ time.

The total running time is then

$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1 \\ 7T(n/2) + \Theta(n^2), & \text{if } n > 1 \end{cases}$$

and the solution $T(n) = \Theta(n^{\lg 7})$ via MT1.

4.3.2. Strassen's algorithm for matrix multiplication



What are 10 S matrices?

1	$S_1 = B_{12} - B_{22}$
2	$S_2 = A_{11} + A_{12}$
3	$S_3 = A_{21} + A_{22}$
4	$S_4 = B_{21} - B_{11}$
5	$S_5 = A_{11} + A_{22}$
6	$S_6 = B_{11} + B_{22}$
7	$S_7 = A_{12} - A_{22}$
8	$S_8 = B_{21} + B_{22}$
9	$S_9 = A_{11} - A_{21}$
10	$S_{10} = B_{11} + B_{12}$

What are 7 P matrices?

1	$P_1 = A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22}$
2	$P_2 = S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22}$
3	$P_3 = S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11}$
4	$P_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11}$
5	$P_5 = S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}$
6	$P_6 = S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}$
7	$P_7 = S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}$



4.3.2. Strassen's algorithm for matrix multiplication

How to combine P_i matrices to get C_{ij} ?

$$1 \quad C_{11} = P_5 + P_4 - P_2 + P_6$$

$$2 \quad C_{12} = P_1 + P_2$$

$$3 \quad C_{21} = P_3 + P_4$$

$$4 \quad C_{22} = P_5 + P_1 - P_3 - P_7$$

$$C_{12} = SQMR(A_{11}, B_{12}) + SQMR(A_{12}, B_{22})$$

$$P_1 = A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22}$$

$$P_2 = S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22}$$

In the original recursion algorithm, $C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$.

In this work, $C_{12} = P_1 + P_2 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22} + A_{11} \cdot B_{22} + A_{12} \cdot B_{22}$
 $= A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$.

This step takes $\Theta(n^2)$ steps.

Overall, $T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2) = \Theta(n^{\lg 7}) < \Theta(n^3)$.



4.4. Conclusion

4.2. Recursion Analysis Solutions

4.2.1. Substitution Method – Make a guessing function and proof the boundaries

4.2.2. Recursion Tree – may solve the problem directly or be used for making a guessing function.

4.2.3. Master's Theorem – A powerful and simple technique. But it is strongly restricted to divide-and-conquer algorithms and regularities.

4.3. Divide and Conquer

4.3.1. Maximum-Sum Array

4.3.2. Strassen's Theorem