Algorithms
Chapter 4 Divide-and-Conquer

Associate Professor: Ching-Chi Lin
林清池 副教授
chingchi.lin@gmail.com

Department of Computer Science and Engineering
National Taiwan Ocean University

Outline

- **The substitution method**
- The recursion-tree method
- The master method
- The maximum-subarray problem
- Strassen’s algorithm for matrix multiplication

The purpose of this chapter

- When an algorithm contains a recursive call to itself, its running time can often be described by a recurrence.
- A **recurrence** is an equation or inequality that describes a function in terms of its value on small inputs.
  - For example: the worst-case running time of Merge-Sort

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

  - The solution is $T(n) = \Theta(n \lg n)$.
- Three methods for solving recurrences
  - the substitution method
  - the recursion-tree method
  - the master method

Technicalities$_{1/2}$

- The running time $T(n)$ is only defined when n is an integer, since the size of the input is always an integer for most algorithms.
  - For example: the running time of Merge-Sort is really

$$T(n)=\begin{cases}\theta(1) & \text{if } n=1,\\ T(\lceil n/2\rceil)+T(\lfloor n/2\rfloor)+\theta(n) & \text{if } n>1.\end{cases}$$

- Typically, we ignore the boundary conditions.
- Since the running time of an algorithm on a constant-sized input is a constant, we have $T(n) = \Theta(1)$ for sufficiently small n.
- Thus, we can rewrite the recurrence as

$$T(n) = 2T(n/2)+ \Theta(n).$$

Technicalities$_{2/2}$

- When we state and solve recurrences, we often omit floors, ceilings, and boundary conditions.
- We forge ahead without these details and later determine whether or not they matter.
- They usually don't, but it is important to know when they do.

The substitution method$_{1/3}$

- The substitution method entails two steps:
  - Guess the form of the solution
  - Use mathematical induction to find the constants and show that the solution works
- This method is powerful, but it obviously can be applied only in cases when it is easy to guess the form of the answer

The substitution method$_{2/3}$

- For example: determine an upper bound on the recurrence

$$T(n) = 2T(\lfloor n/2 \rfloor) + n, \text{ where } T(1) = 1.$$

  - guess that the solution is $T(n) = O(n \lg n)$
  - prove that there exist positive constants $c > 0$ and n_0 such that $T(n) \leq cn \lg n$ for all $n \geq n_0$
  - **Basis step**:
    - when $n = 1$, $T(1) \leq c_1 1 \lg 1 = 0$, which is at odds with $T(1) = 1$
    - since the recurrence does not depend directly on $T(1)$, we can replace $T(1)$ by $T(2) = 4$ and $T(3) = 5$ as the base cases
    - $T(2) \leq c_1 2 \lg 2$ and $T(3) \leq c_1 3 \lg 3$ for any choice of $c_1 \geq 2$
    - thus, we can choose $c_1 = 2$ and $n_0 = 2$

The substitution method$_{3/3}$

- **Induction step**:
  - assume $T(\lfloor n/2 \rfloor) \le c_2 \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$ for $\lfloor n/2 \rfloor$
  - then, $T(n) = 2T(\lfloor n/2 \rfloor) + n$

$$
\begin{aligned}
&\le 2(c_2 \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n \\
&\le c_2 n \lg(n/2) + n \\
&= c_2 n \lg n - c_2 n \lg 2 + n \\
&= c_2 n \lg n - c_2 n + n \\
&\le c_2 n \lg n
\end{aligned}
$$

  - the last step holds as long as $c_2 \ge 1$

- There exist positive constants $c = \max\{2, 1\}$ and $n_0 = 2$ such that $T(n) \le cn \lg n$ for all $n \ge n_0$

Making a good guess

- **Experience**: $T(n)=2T(\lfloor n/2\rfloor+17)+n$
  - when n is large, the difference between $T(\lfloor n/2\rfloor)$ and $T(\lfloor n/2\rfloor+17)$ is not that large: both cut n nearly evenly in half.
  - we make the guess that $T(n) = O(n\lg n)$
- **Loose upper and lower bounds**: $T(n)=2T(\lfloor n/2\rfloor)+n$
  - prove a lower bound of $T(n) = \Omega(n)$ and an upper bound of $T(n) = O(n^2)$
  - gradually lower the upper bound and raise the lower bound until we converge on the tight solution of $T(n) = \Theta(n\lg n)$
- **Recursion trees**:
  - will be introduced later

Subtleties

- Consider the recurrence:

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

  - **guess** that the solution is $O(n)$, i.e., $T(n) \le cn$
  - then, $T(n) \le c\lfloor n/2 \rfloor + (c\lceil n/2 \rceil + 1$
    $= cn + 1$
  - c does not exist
  - **new guess** $T(n) \le cn - b$
  - then, $T(n) \le (c\lfloor n/2 \rfloor - b) + (c\lceil n/2 \rceil - b) + 1$
    $= cn - 2b + 1$
    $\le cn - b \qquad$ for $b \ge 1$
  - also, the constant c must be chosen large enough to handle the boundary conditions

Avoiding pitfalls

- It is easy to err in the use of asymptotic notation.
- For example: $T(n) = 2T(\lfloor n/2 \rfloor) + n$
  - guess that the solution is $O(n)$, i.e., $T(n) \leq cn$
  - then, $T(n) = 2T(\lfloor n/2 \rfloor) + n$

$$\begin{aligned} &\leq 2(c\lfloor n/2 \rfloor) + n \\ &\leq cn + n \\ &= O(n) \quad \Leftarrow \textbf{wrong} \end{aligned}$$

  - the error is that we haven't proved the **exact form** of the inductive hypothesis, that is, that $T(n) \leq cn$

Outline

- The substitution method
- **The recursion-tree method**
- The master method
- The maximum-subarray problem
- Strassen’s algorithm for matrix multiplication

The recursion-tree method

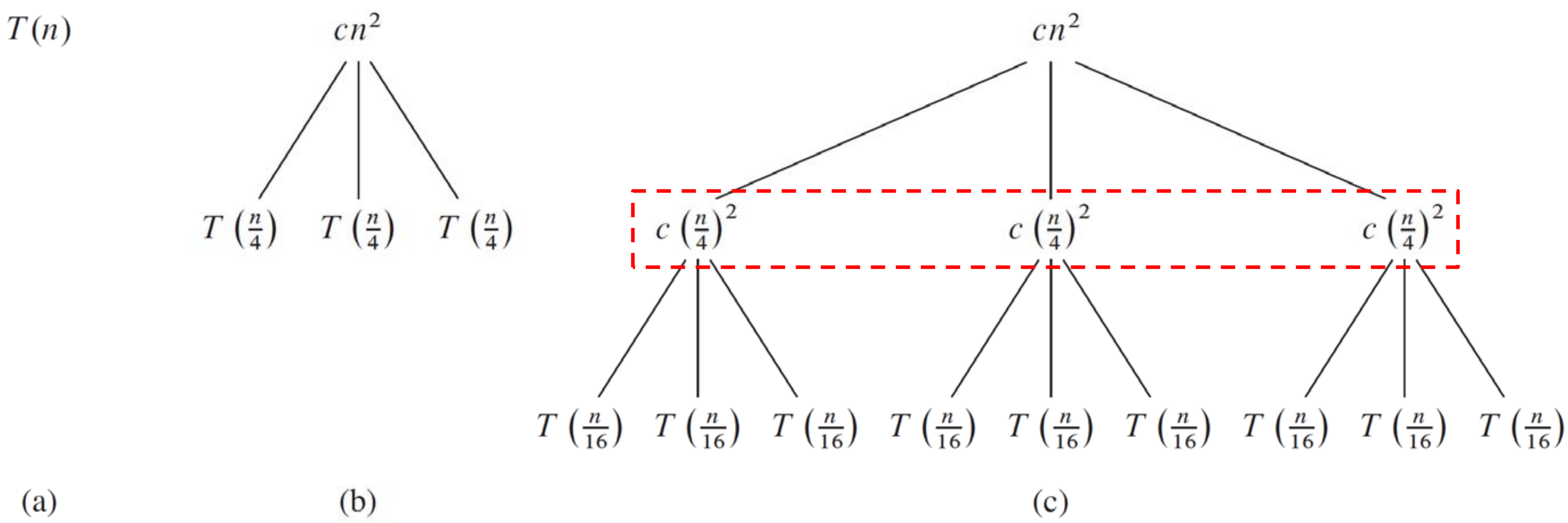
- A recursion tree is best used to generate a good guess, which is then verified by the substitution method.
- Tolerating a small amount of “sloppiness”, we could use recursion-tree to generate a good guess.
- One can also use a recursion tree as a direct proof of a solution to a recurrence.
- Ideas:
  - in a **recursion tree**, each node represents the cost of a single subproblem
  - sum the costs within each level to obtain a set of per-level costs
  - sum all the per-level costs to determine the total cost

An example

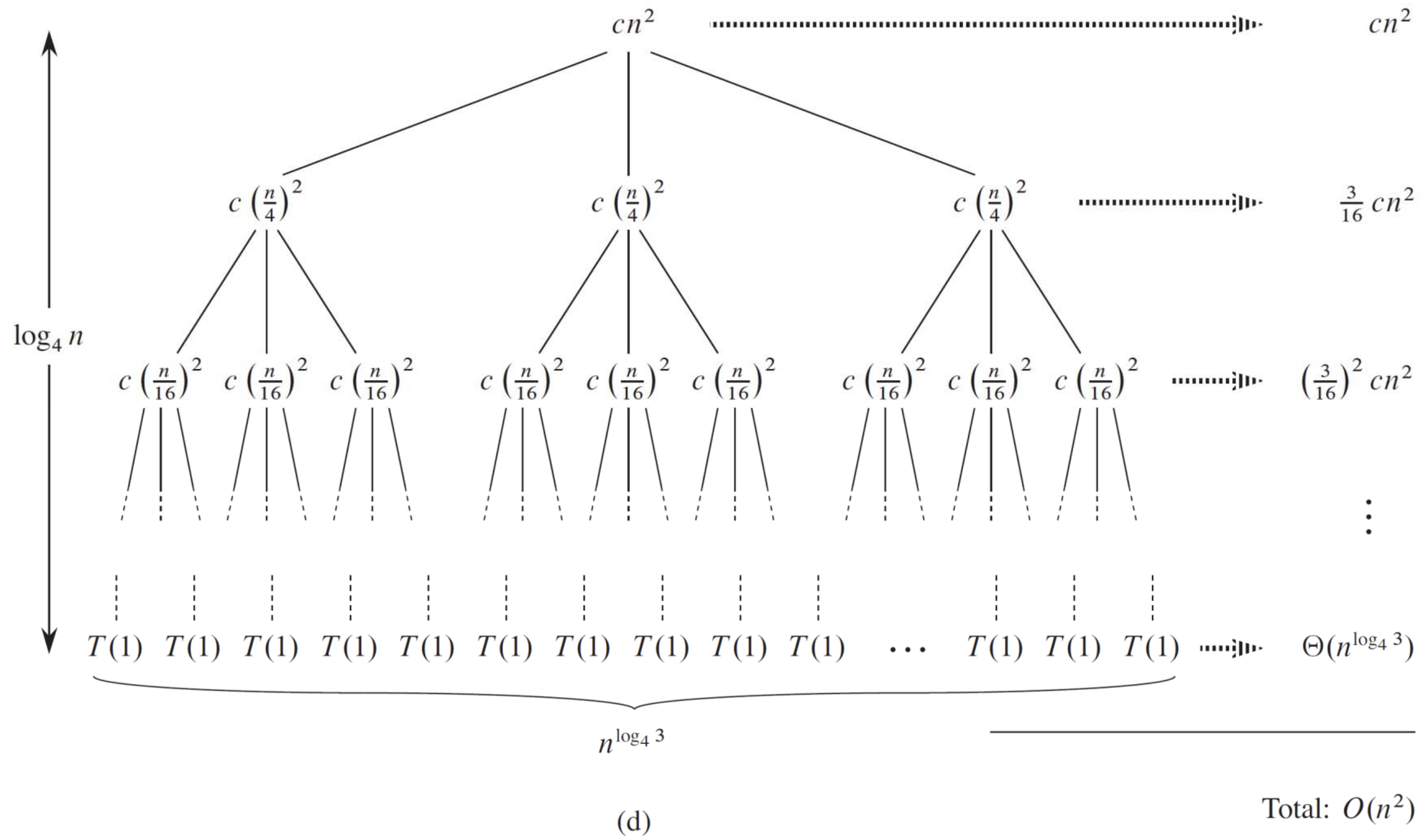
- For example: $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$
- Tolerating the sloppiness:
  - ignore the floor in the recurrence
  - assume n is an exact power of 4
- Rewrite the recurrence as $T(n) = 3T(n/4) + cn^2$

The construction of a recursion tree$_{1/2}$

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



The construction of a recursion tree$_{2/2}$



Determine the cost of the tree$_{1/2}$

- The subproblem size for a node at depth i is $n/4^i$.
- Thus, the tree has $\log_4 n + 1$ levels (0, 1, 2,..., $\log_4 n$).
- Each node at depth i, has a cost of $c(n/4^i)^2$ for $0 \le i \le \log_4 n$.
- So, the total cost over all nodes at depth i is $3^i * c(n/4^i)^2 = (3/16)^i cn^2$.
- The last level, at $\log_4 n$, has $3^{\log_4 n} = n^{\log_4 3}$ nodes.
- The cost of the entire tree:

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

$$= \sum_{i=0}^{\log_4 n} (\frac{3}{16})^i cn^2$$

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

Determine the cost of the tree$_{2/2}$

- Take advantage of small amounts of sloppiness, we have

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n} \left(\frac{3}{16}\right)^i cn^2 \\
&< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 = \frac{1}{1-(3/16)} cn^2 \\
&= \frac{16}{13} cn^2 = O(n^2)
\end{aligned}
$$

- Thus, we have derived a guess of $T(n)=O(n^2)$.

Verify the correctness of our guess

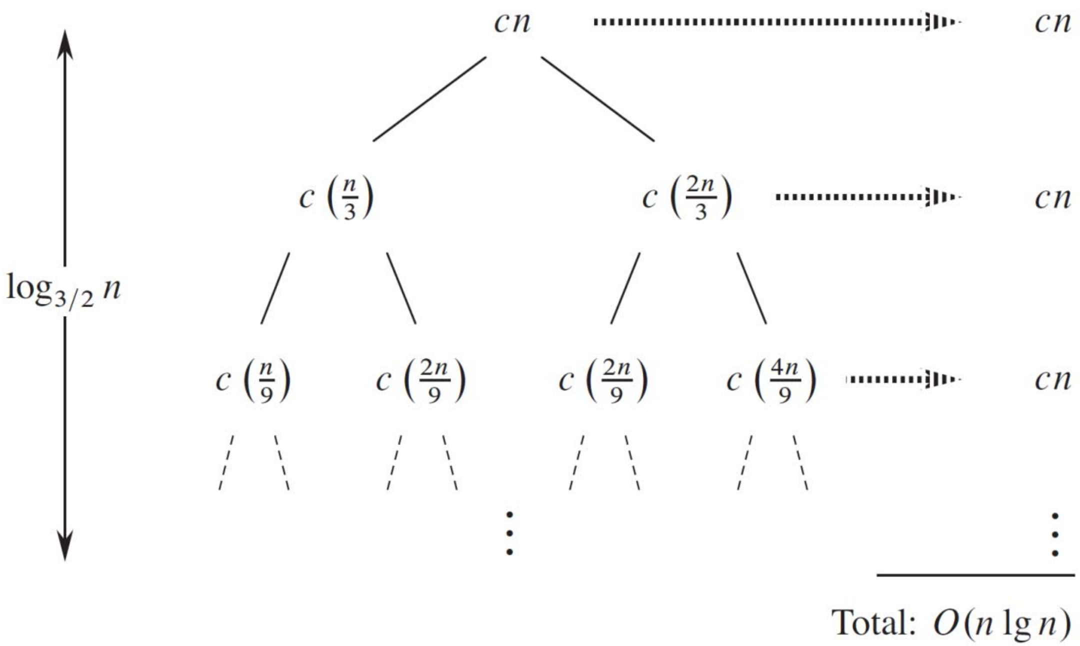
- Now we can use the substitution method to verify that our guess is correct.
- We want to show that $T(\mathrm{n}) \le dn^2$ for some constant $d > 0$.
- Using the same constant $c > 0$ as before, we have

$$\begin{aligned} T(n) &= 3T(\lfloor n/4 \rfloor) + \theta(n^2) \\ &\le 3T(\lfloor n/4 \rfloor) + cn^2 \\ &\le 3d\lfloor n/4 \rfloor^2 + cn^2 \\ &\le 3d(n/4)^2 + cn^2 \\ &= 3/16dn^2 + cn^2 \\ &\le dn^2, \end{aligned}$$

where the last step holds for $d \ge (16/13)c$.

Another example

- Another example: $T(n) = T(n/3)+T(2n/3)+O(n)$.
- The recursion-tree:



Determine the cost of the tree

- In the figure, we get a value of *cn* for every level.
- The height of tree is $\log_{3/2} n$.
- Intuitively, we expect the solution to the recurrence to be at most $O(cn \log_{3/2} n) = O(n \lg n)$.
- The recursion tree has fewer than $2^{\log_{3/2} n} = n^{\log_{3/2} 2}$ leaves.
- The total cost of all leaves would then be $\theta(n^{\log_{3/2} 2})$, which is $\omega(n \lg n)$.
- Also, not all levels contribute a cost of exactly *cn*.
- Thus, we derived a guess of $T(n) = O(n \lg n)$.

Verify the correctness of our guess

- We can verify the guess by the substitution method.

- We have
$$
\begin{aligned}
T(n) &= T(n/3)+T(2n/3)+O(n) \\
&\le T(n/3)+T(2n/3)+cn \\
&\le d(n/3)\lg(n/3)+d(2n/3)\lg(2n/3)+cn \\
&= (d(n/3)\lg n-d(n/3)\lg 3) \\
&\quad +(d(2n/3)\lg n-d(2n/3)\lg(3/2))+cn \\
&= dn\lg n-d((n/3)\lg 3+(2n/3)\lg(3/2))+cn \\
&= dn\lg n-d((n/3)\lg 3+(2n/3)\lg 3-(2n/3)\lg 2)+cn \\
&= dn\lg n-dn(\lg 3-2/3)+cn \\
&\le dn\lg n
\end{aligned}
$$

for $d \ge c/(\lg 3-(2/3))$.

Outline

- The substitution method
- The recursion-tree method
- **The master method**
- The maximum-subarray problem
- Strassen's algorithm for matrix multiplication

The master method$_{1/2}$

- The master method provides a "cookbook" method for solving recurrences of the form

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

  - $a \geq 1$ and $b > 1$ are constants
  - $f(n)$ is an asymptotically positive function

- It requires memorization of three cases, but then the solution of many recurrences can be determined quite easily.

The master method$_{2/2}$

- The recurrence $T(n) = aT(n/b) + f(n)$ describes the running time of an algorithm that
  - divides a problem of size n into a subproblems, each of size n/b
  - each of subproblems is solved recursively in time $T(n/b)$
  - the cost of dividing and combining the results is $f(n)$
- For example, the recurrence arising from the MERGE-SORT procedure has $a = 2$, $b = 2$, and $f(n) = \Theta(n)$.
- Normally, we omit the floor and ceiling functions when writing divide-and-conquer recurrences of this form.

Master theorem

- **Master theorem:** Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

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

  where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.
  Then, $T(n)$ can be bounded asymptotically as follows.

  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 $af(n/b) \leq cf(n)$ for some constant $c<1$ and all sufficiently large n, then $T(n) = \theta(f(n))$.

Intuition behind the master method

- Intuitively, the solution to the recurrence is determined by comparing the two functions $f(n)$ and $n^{\log_b a}$.
  - Case 1: if $n^{\log_b a}$ is asymptotically larger than $f(n)$ by a factor of n^{ε} for some constant $\varepsilon > 0$, then the solution is $T(n) = \theta(n^{\log_b a})$.
  - Case 2: if $n^{\log_b a}$ is asymptotically equal to $f(n)$, then the solution is $T(n) = \theta(n^{\log_b a} \lg n)$.
  - Case 3: if $n^{\log_b a}$ is asymptotically smaller then $f(n)$ by a factor of n^{ε}, and the function $f(n)$ satisfies the "regularity“ condition that $af(n/b) \le cf(n)$, then the solution is $T(n) = \theta(f(n))$.
- The three cases do not cover all the possibilities for $f(n)$.

Using the master method$_{1/3}$

- Example 1: $T(n) = 9T(n/3) + n$
  - For this recurrence, we have $a = 9$, $b = 3$, $f(n) = n$.
  - Thus, $n^{\log_b a} = n^{\log_3 9} = \theta(n^2)$.
  - Since $f(n) = O(n^{\log_3 9-\varepsilon})$, where $\varepsilon = 1$, we can apply case 1.
  - The solution is $T(n) = \Theta(n^2)$.

- Example 2: $T(n) = T(2n/3) + 1$
  - For this recurrence, we have $a = 1$, $b = 3/2$, $f(n) = 1$.
  - Thus, $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$.
  - Since $f(n) = O(n^{\log_b a}) = \theta(1)$, we can apply case 2.
  - The solution is $T(n) = \Theta(\lg n)$.

Using the master method$_{3/3}$

- Example 4: $T(n) = 2T(n/2) + n\lg n$
  - For this recurrence, we have $a = 2$, $b = 2$, $f(n) = n\lg n$.
  - The function $f(n) = n\lg n$ is asymptotically larger than $n^{\log_b a} = n^{\log_2 2} = n$.
  - But, it is not polynomially larger since the ratio $f(n)/n^{\log_b a} = (n\lg n)/n = \lg n$ is asymptotically less than n^{ε} for any positive constant ε.
  - Consequently, the recurrence falls into the gap between case 2 and case 3.

- If $g(n)$ is asymptotically larger than $f(n)$ by a factor of n^{ε} for some constant $\varepsilon>0$, then we said $g(n)$ is **polynomially larger** than $f(n)$.

Using the master method$_{2/3}$

- Example 3: $T(n) = 3T(n/4) + n \lg n$
  - For this recurrence, we have $a = 3$, $b = 4$, $f(n) = n \lg n$.
  - Thus, $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$.
  - For sufficiently large n, $af(n/b) = 3(n/4)\lg(n/4) \leq (3/4)n \lg n = cf(n)$ for $c = 3/4$.
  - Since $f(n) = \Omega(n^{\log_4 3+\varepsilon})$ with $\varepsilon \approx 0.2$ and the regularity condition holds for $f(n)$ case 3 applies.
  - The solution is $T(n) = \Theta(n \lg n)$.

Outline

- The substitution method
- The recursion-tree method
- The master method
- **The maximum-subarray problem**
- Strassen's algorithm for matrix multiplication

The maximum-subarray problem

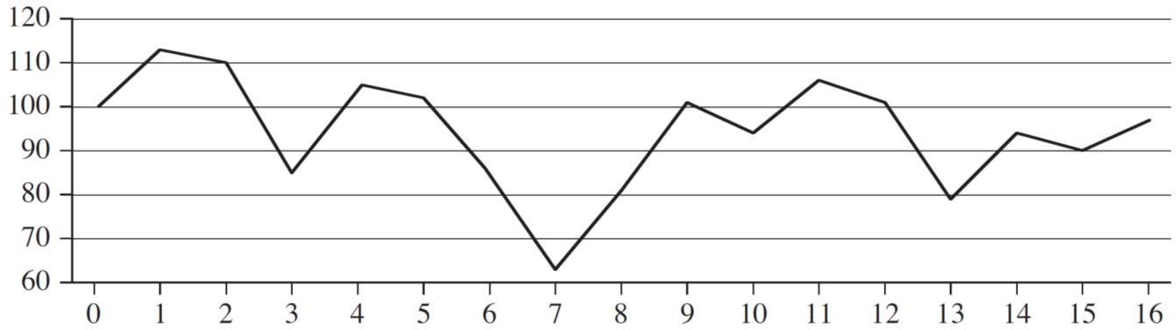
- **Input:** An array $A[1 \ldots n]$ of numbers.
  - Assume that some of the numbers are negative, because this problem is trivial when all numbers are nonnegative.
- **Output:** Indices i and j such that $A[i \ldots j]$ has the greatest sum of any contiguous subarray of array A.
- For example:
  - The subarray $A[8 \ldots 11]$, with sum 43, has the greatest sum of any contiguous subarray of array A.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

maximum subarray

An application

- You have the prices that a stock traded at over a period of *n* consecutive days.
- When should you have bought the stock? When should you have sold the stock?



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

Solving by brute-force algorithm

- **Brute-force algorithm:**
  - Check all $\binom{n}{2} + n = \theta(n^2)$ subarrays.
  - Organize the computation so that each subarray *A*[*i*…*j*] takes *O*(1) time.
  - So that the brute-force solution takes $\theta(n^2)$ time.

A[9…13] = -1

	8	9	10	11	12	13	14	15	16	17
A	…	-6	5	-4	3	1	-2	3	-6	…

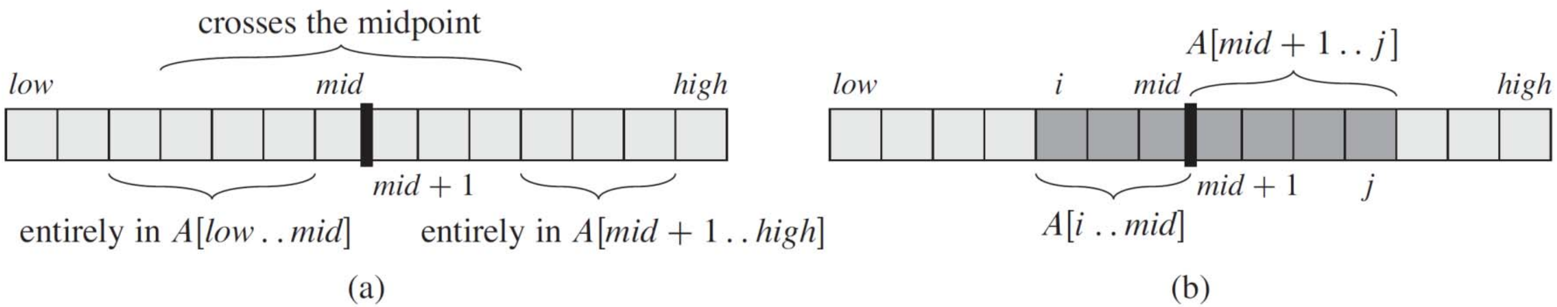
O(1) time

A[9…14] = -3

	8	9	10	11	12	13	14	15	16	17
A	…	-6	5	-4	3	1	-2	3	-6	…

Solving by divide-and-conquer

- **Divide** by splitting into two subarrays $A[low…mid]$ and $A[mid+1…high]$, mid is the midpoint of $A[low…high]$.
- **Conquer** by recursively finding a maximum subarrays of the two subarrays $A[low…mid]$ and $A[mid+1…high]$.
- **Combine** by finding a maximum subarray that crosses the midpoint, and using the best solution out of the three.

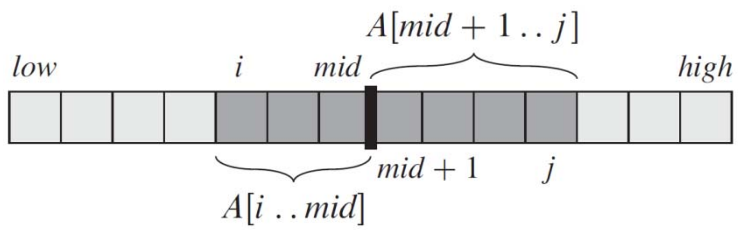


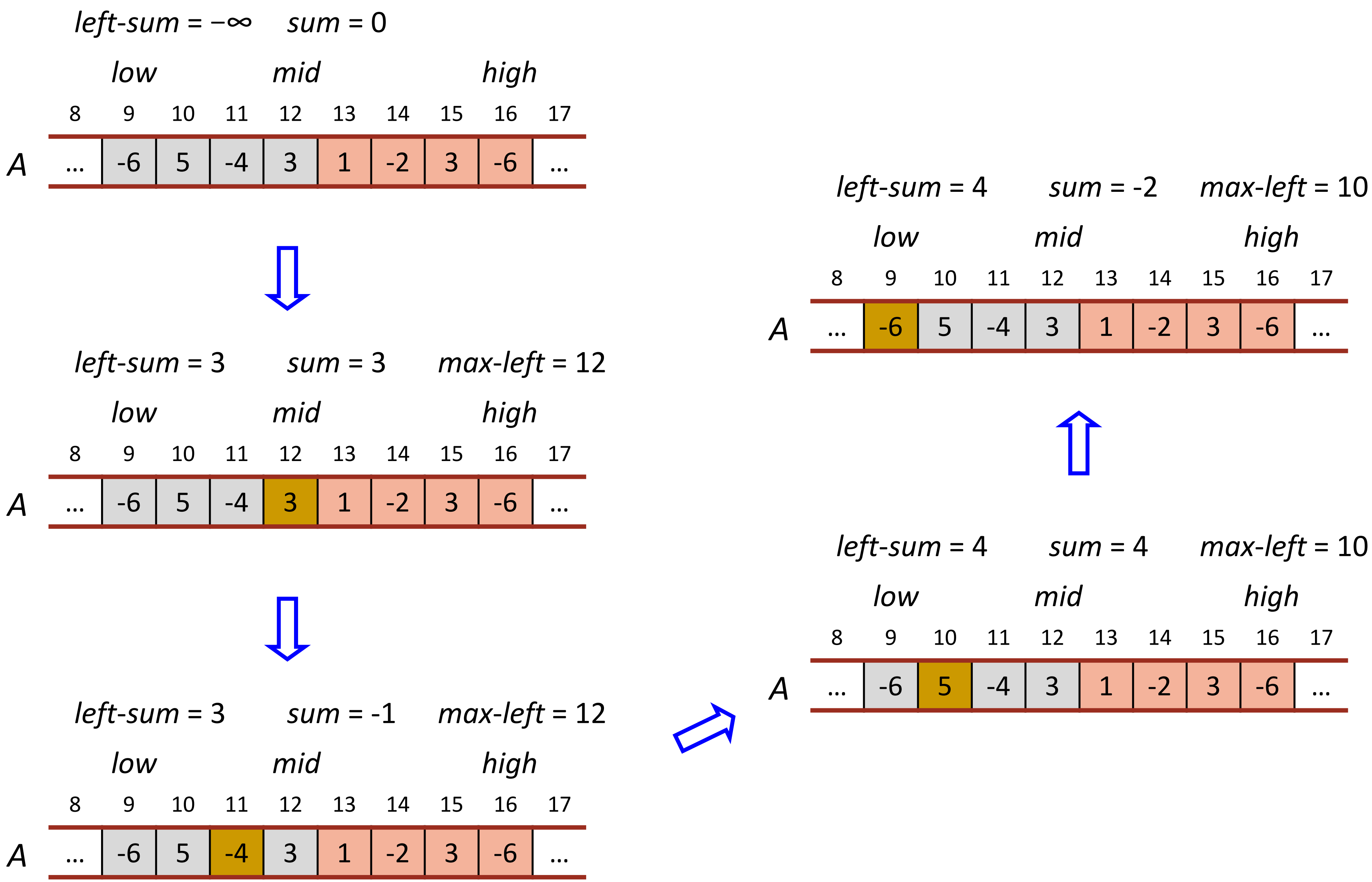
Maximum subarray that crosses the midpoint

- **Not** a smaller instance of the original problem: has the added restriction that the subarray must cross the midpoint.

- Any subarray crossing the midpoint $A[mid]$ is made of two subarrays $A[i \ldots mid]$ and $A[mid+1 \ldots j]$.

- Find maximum subarrays of the form $A[i \ldots mid]$ and $A[mid+1 \ldots j]$ and then combine them.




left-sum = −∞ sum = 0
low mid high
8 9 10 11 12 13 14 15 16 17
A ... -6 5 -4 3 1 -2 3 -6 ...
left-sum = 3 sum = 3 max-left = 12
low mid high
8 9 10 11 12 13 14 15 16 17
A ... -6 5 -4 3 1 -2 3 -6 ...
left-sum = 3 sum = -1 max-left = 12
low mid high
8 9 10 11 12 13 14 15 16 17
A ... -6 5 -4 3 1 -2 3 -6 ...
left-sum = 4 sum = 4 max-left = 10
low mid high
8 9 10 11 12 13 14 15 16 17
A ... -6 5 -4 3 1 -2 3 -6 ...
left-sum = 4 sum = -2 max-left = 10
low mid high
8 9 10 11 12 13 14 15 16 17
A ... -6 5 -4 3 1 -2 3 -6 ...

Find-Max-Crossing-Subarray

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
    left-sum ← −∞
    sum ← 0                                   } Θ(1)
    for i ← mid downto low
        sum ← sum + A[i]
        if sum > left-sum                     } Θ(n)
            left-sum ← sum
            max-left ← i
    right-sum ← −∞
    sum ← 0                                   } Θ(1)
    for j ← mid +1 to high
        sum ← sum + A[j]
        if sum > right-sum                    } Θ(n)
            right-sum ← sum
            max-right ← j
    return (max-left, max-right, left-sum + right-sum)
```

Time: $\Theta(n)$.

Divide-and-conquer procedure

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

if *high* == *low* $\Theta(1)$
    return (*low*, *high*, *A*[*low*])   // base case: only one element
else *mid* ← $\lfloor (low + high)/2 \rfloor$
    (*left-low*, *left-high*, *left-sum*) ← $T(n/2)$
        FIND-MAXIMUM-SUBARRAY(*A*, *low*, *mid*)
    (*right-low*, *right-high*, *right-sum*) ← $T(n/2)$
        FIND-MAXIMUM-SUBARRAY(*A*, *mid*+1, *high*)
    (*cross-low*, *cross-high*, *cross-sum*) ← $\Theta(n)$
        FIND-MAX-CROSSING-SUBARRAY(*A*, *low*, *mid*, *high*)
if *left-sum* ≥ *right-sum* and *left-sum* ≥ *cross-sum*
    return (*left-low*, *left-high*, *left-sum*)
elseif *right-sum* ≥ *left-sum* and *right-sum* ≥ *cross-sum* $\Theta(1)$
    return (*right-low*, *right-high*, *right-sum*)
else return (*cross-low*, *cross-high*, *cross-sum*)

Initial call : FIND-MAXIMUM-SUBARRAY(*A*, 1, *n*)

Analyzing maximum-subarray

- For simplicity, assume that *n* is a power of 2.

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

  - The base case occurs when $n = 1$.
  - **Divide**: compute the middle of the subarray, $D(n) = \Theta(1)$.
  - **Conquer**: Recursively solve 2 subproblems, each of size $n/2$.
  - **Combine**: Combining consists of calling FIND-MAX-CROSSING-SUBARRAY, which takes $\Theta(n)$ time, and a constant number of constant-time tests $\Rightarrow C(n) = \Theta(n) + \Theta(1)$ time for combining.
- By using master method, we have $T(n) = \Theta(n \lg n)$.

Outline

- The substitution method
- The recursion-tree method
- The master method
- The maximum-subarray problem
- **Strassen's algorithm for matrix multiplication**

Matrix multiplication

- **Input:** Two $n \times n$ matrices, $A = (a_{ij})$ and $B = (b_{ij})$.
- **Output:** $n \times n$ matrix, $C = (c_{ij})$, where $C = A \cdot B$,i.e.,

$$c_{ji} = \sum_{k=1}^{n} a_{ik} b_{kj} \qquad \text{for } i, j = 1, 2, \dots, n.$$

  - Need to compute n^2 entries of C.
  - Each entry is the sum of n values.

SQUARE-MATRIX-MULTIPLY (A, B)

$n \leftarrow A.rows$
let C be a new $n \times n$ *matrix*
for $i \leftarrow 1$ **to** n
    for $j \leftarrow 1$ **to** n
        $c_{ij} \leftarrow 0$
        for $k \leftarrow 1$ **to** n
            $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$
Return C

Time: $\Theta(n^3)$.

Simple divide-and-conquer method

- Partition each of A, B and C into four $n/2 \times n/2$ matrices, so that we rewrite the equation $C = A \cdot B$ as

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

- The four corresponding equations are:
  - $C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$,
  - $C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$,
  - $C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$,
  - $C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$.
- Each of these equations multiplies two $n/2 \times n/2$ matrices and then adds their $n/2 \times n/2$ products.

Procedure of matrix-multiply-recursive

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

$n \leftarrow A.rows$

let C be a new $n \times n$ matrix — $\Theta(1)$

if $n == 1$ // base case: only one element

$c_{11} \leftarrow a_{11} \cdot b_{11}$ — $\Theta(1)$

else partition each of A, B and C into four $n/2 \times n/2$ matrices

$C_{11} \leftarrow$ SQUARE-MATRIX-MULTIPLY-RECURSIVE (A_{11}, B_{11})
+ SQUARE-MATRIX-MULTIPLY-RECURSIVE (A_{12}, B_{21})

$C_{12} \leftarrow$ SQUARE-MATRIX-MULTIPLY-RECURSIVE (A_{11}, B_{12})
+ SQUARE-MATRIX-MULTIPLY-RECURSIVE (A_{12}, B_{22})

$C_{21} \leftarrow$ SQUARE-MATRIX-MULTIPLY-RECURSIVE (A_{21}, B_{11})
+ SQUARE-MATRIX-MULTIPLY-RECURSIVE (A_{22}, B_{21})

$C_{22} \leftarrow$ SQUARE-MATRIX-MULTIPLY-RECURSIVE (A_{21}, B_{12})
+ SQUARE-MATRIX-MULTIPLY-RECURSIVE (A_{22}, B_{22}) — $8T(n/2) + \Theta(n^2)$

return C

Analyzing

- For simplicity, assume that n is a power of 2.

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

  - The base case occurs when $n = 1$.
  - **Divide**: Partition A, B and C into four $n/2 \times n/2$ matrices by index calculation takes $\Theta(1)$, $D(n) = \Theta(1)$.
  - **Conquer**: Recursively solve 8 subproblems, each of size $n/2$.
  - **Combine**: Combining takes $\Theta(n^2)$ time to add $n/2 \times n/2$ matrices four times. $\Rightarrow C(n) = \Theta(n^2)$ time for combining.
- By using master method, we have $T(n) = \Theta(n^3)$.

Strassen's method

- Step 1: partition each of A, B and C into four $n/2 \times n/2$ matrices. **Time: $\Theta(1)$.**
- Step 2: create 10 matrices $S_1, S_2, \ldots, S_{10}$, each of which is $n/2 \times n/2$ and is the sum or difference of two matrices created in step 1. **Time: $\Theta(n^2)$.**
- Step 3: using the submatrices created in Step 1 and the 10 matrices created in step 2, recursively compute seven matrix products $P_1, P_2, \ldots, P_7$. Each matrix P_i is $n/2 \times n/2$. **Time: $7T(n/2)$.**
- Step 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. **Time: $\Theta(n^2)$.**

Step 2: create the 10 matrices

- $S_1 = B_{12} - B_{22}$,
- $S_2 = A_{11} + A_{12}$,
- $S_3 = A_{21} + A_{22}$,
- $S_4 = B_{21} - B_{11}$,
- $S_5 = A_{11} + A_{22}$,
- $S_6 = B_{11} + B_{22}$,
- $S_7 = A_{12} - A_{22}$,
- $S_8 = B_{21} + B_{22}$,
- $S_9 = A_{11} - A_{21}$,
- $S_{10} = B_{11} + B_{12}$. **Time: $\Theta(n^2)$.**

Step 3: create the 7 matrices

- $P_1 = A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22}$,
- $P_2 = S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22}$,
- $P_3 = S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11}$,
- $P_4 = A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11}$,
- $P_5 = S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}$,
- $P_6 = S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}$,
- $P_7 = S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}$.

Time: 7*T*(*n*/2).

Step 4: construct submatrices of C

- $C_{11} = P_5 + P_4 - P_2 + P_6$,
- $C_{12} = P_1 + P_2$,
- $C_{21} = P_3 + P_4$,
- $C_{22} = P_5 + P_1 - P_3 - P_7$.

Time: $\Theta(n^2)$.

Analyzing

- For simplicity, assume that *n* is a power of 2.

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

  - The base case occurs when $n = 1$.
  - **Divide**: Partition A, B and C into four $n/2 \times n/2$ matrices by index calculation takes $\Theta(1)$. Creating the matrices $S_1, S_2, \ldots, S_{10}$, each of which is $n/2 \times n/2$ takes $\Theta(n^2)$, $D(n) = \Theta(1) + \Theta(n^2) = \Theta(n^2)$.
  - **Conquer**: Recursively solve 7 subproblems, each of size $n/2$.
  - **Combine**: Combining takes $\Theta(n^2)$ time to add and subtract $n/2 \times n/2$ matrices. $\Rightarrow C(n) = \Theta(n^2)$ time for combining.

- By using master method, we have $T(n) = \Theta(n^{\lg 7})$.