Applied Algorithms

CSCI-B505 / INFO-I500

Lecture 9.

Divide&Conquer Approach

M. Oguzhan Kulekci

- Divide-and-Conquer Approach

  - Maximal subarray problem

  - Master Theorem (or method) for solving D&C type recurrences

  - Karatsuba matrix multiplication

Divide-And-Conquer Approach

Given a problem,

- Partition it into small problems.

- Solve these small problems.

- Gather the results to arrive the final output.

Widely used in solving many computational challenges, e.g., quick sort, merge sort, tree/graph traversals, etc…

Divide-And-Conquer

Problem: Given an array $A[0..n-1]$ of **integers**, find the subarray $A[i..j]$ with the maximum sum.

0	1	2	3	4	5	6	7
-2	-3	4	-1	-2	1	5	-3

Divide-And-Conquer

Problem: Given an array A[0..n-1] of integers, find the subarray A[i..j] with the maximum sum.
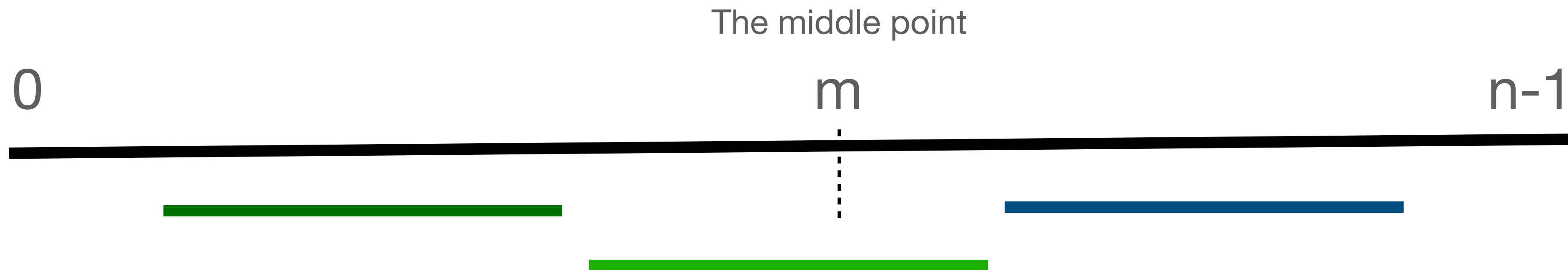
Naive or Brute-Force Solution:

- How many different subarrays exist $\binom{n}{2} = n(n+1)/2 \in O(n^2)$.

- Generate all of them and choose the one with the maximum sum.

How to make it better than $O(n^2)$?

Divide-and-conquer ?

Divide-And-Conquer

Problem: Given an array $A[0..n-1]$ of integers, find the subarray $A[i..j]$ with the maximum sum.

The middle point

0   m   n-1

Maximum subarray $A[i,j]$ is on the left-half if $0 \leq i \leq j < m$

Maximum subarray $A[i,j]$ is on the right-half if $m < i \leq j < n$

Maximum subarray $A[i,j]$ passes over m if $0 \leq i < m \leq j \leq n$

Find left, right and central maximal subarrays and choose the largest one !

Divide-And-Conquer

Find left, half and central maximal subarrays and choose the largest one !
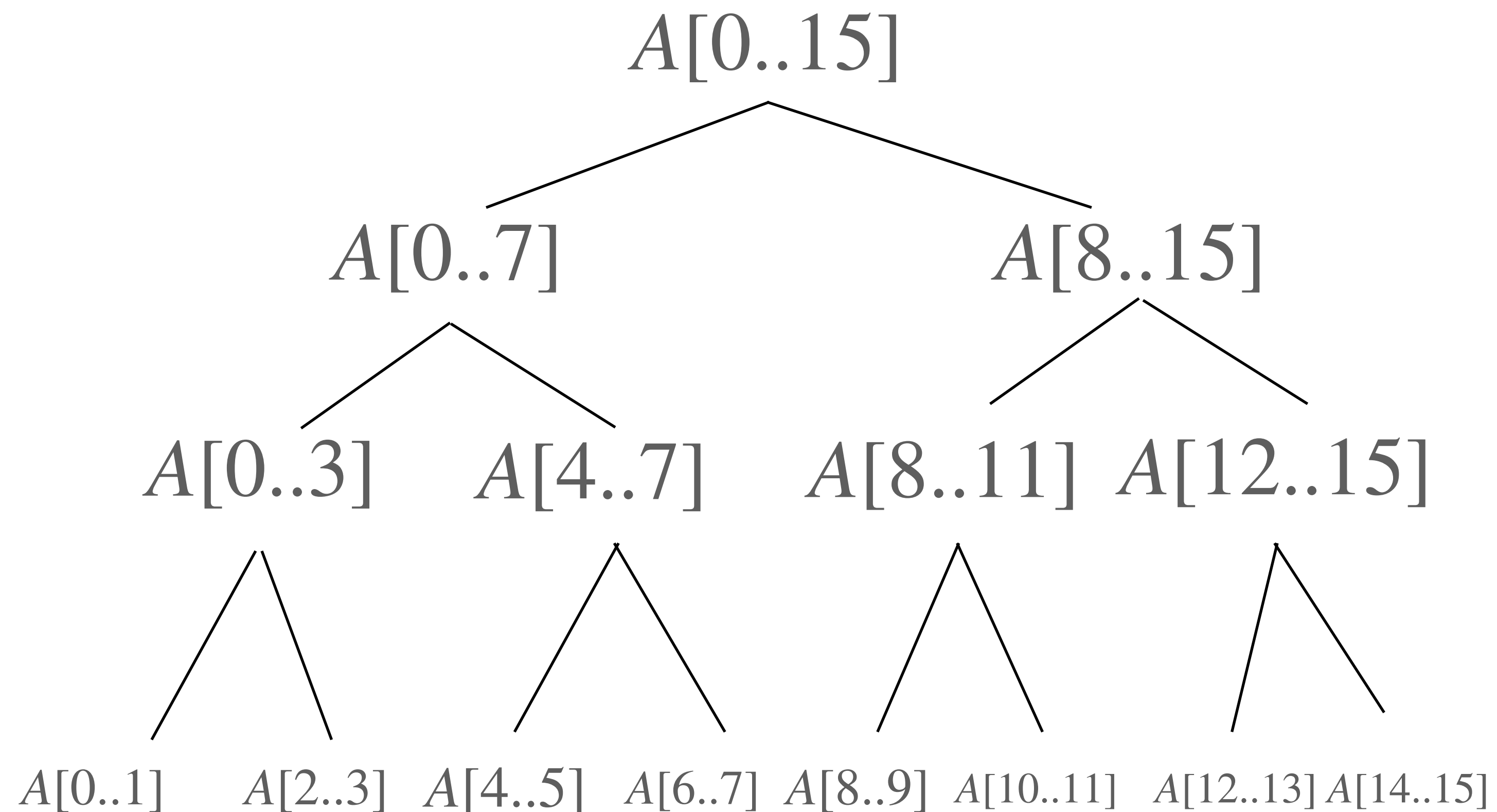
FIND-MAXIMUM-SUBARRAY($A, low, high$)

1 **if** $high == low$
2   **return** $(low, high, A[low])$ // base case: only one element
3 **else** $mid = \lfloor (low + high)/2 \rfloor$
4      $(left\text{-}low, left\text{-}high, left\text{-}sum) =$
            FIND-MAXIMUM-SUBARRAY(A, low, mid)
5      $(right\text{-}low, right\text{-}high, right\text{-}sum) =$
            FIND-MAXIMUM-SUBARRAY($A, mid + 1, high$)
6      $(cross\text{-}low, cross\text{-}high, cross\text{-}sum) =$
            FIND-MAX-CROSSING-SUBARRAY($A, low, mid, high$)
7   **if** $left\text{-}sum \geq right\text{-}sum$ and $left\text{-}sum \geq cross\text{-}sum$
8   **return** $(left\text{-}low, left\text{-}high, left\text{-}sum)$
9   **elseif** $right\text{-}sum \geq left\text{-}sum$ and $right\text{-}sum \geq cross\text{-}sum$
10   **return** $(right\text{-}low, right\text{-}high, right\text{-}sum)$
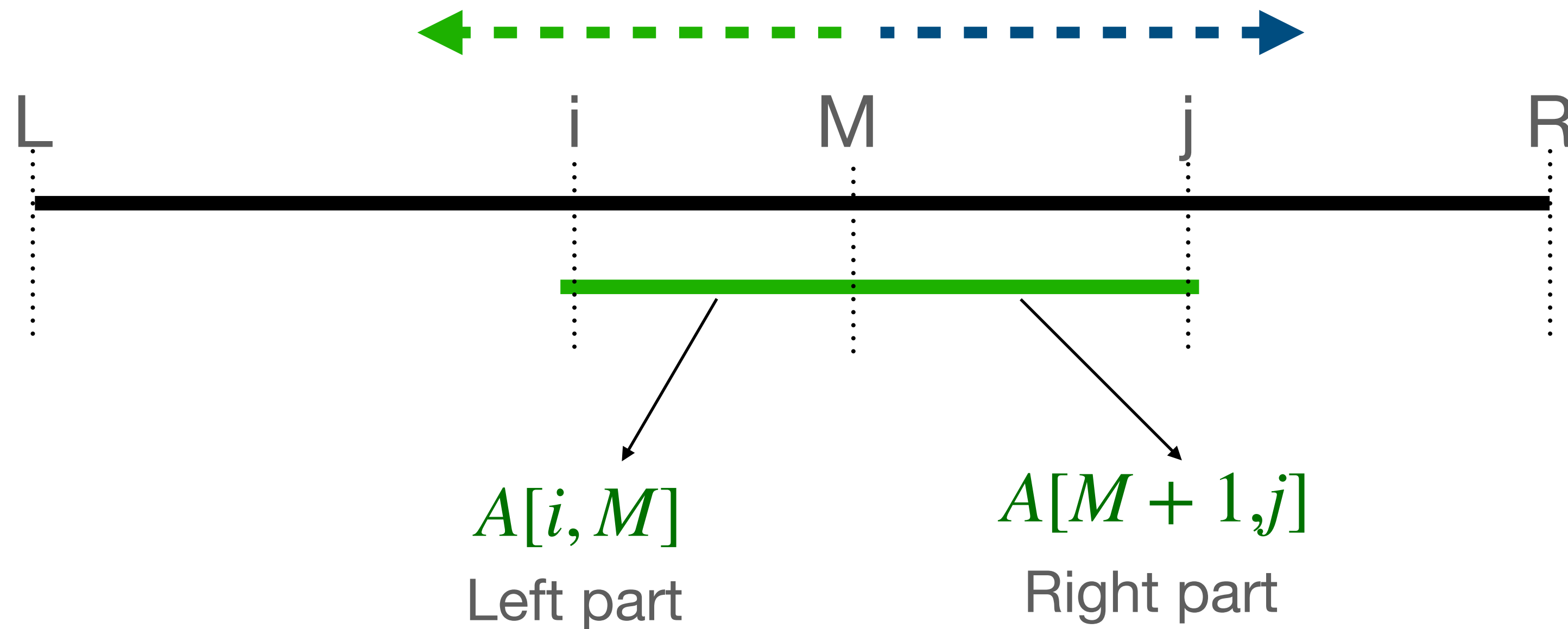11   **else return** $(cross\text{-}low, cross\text{-}high, cross\text{-}sum)$
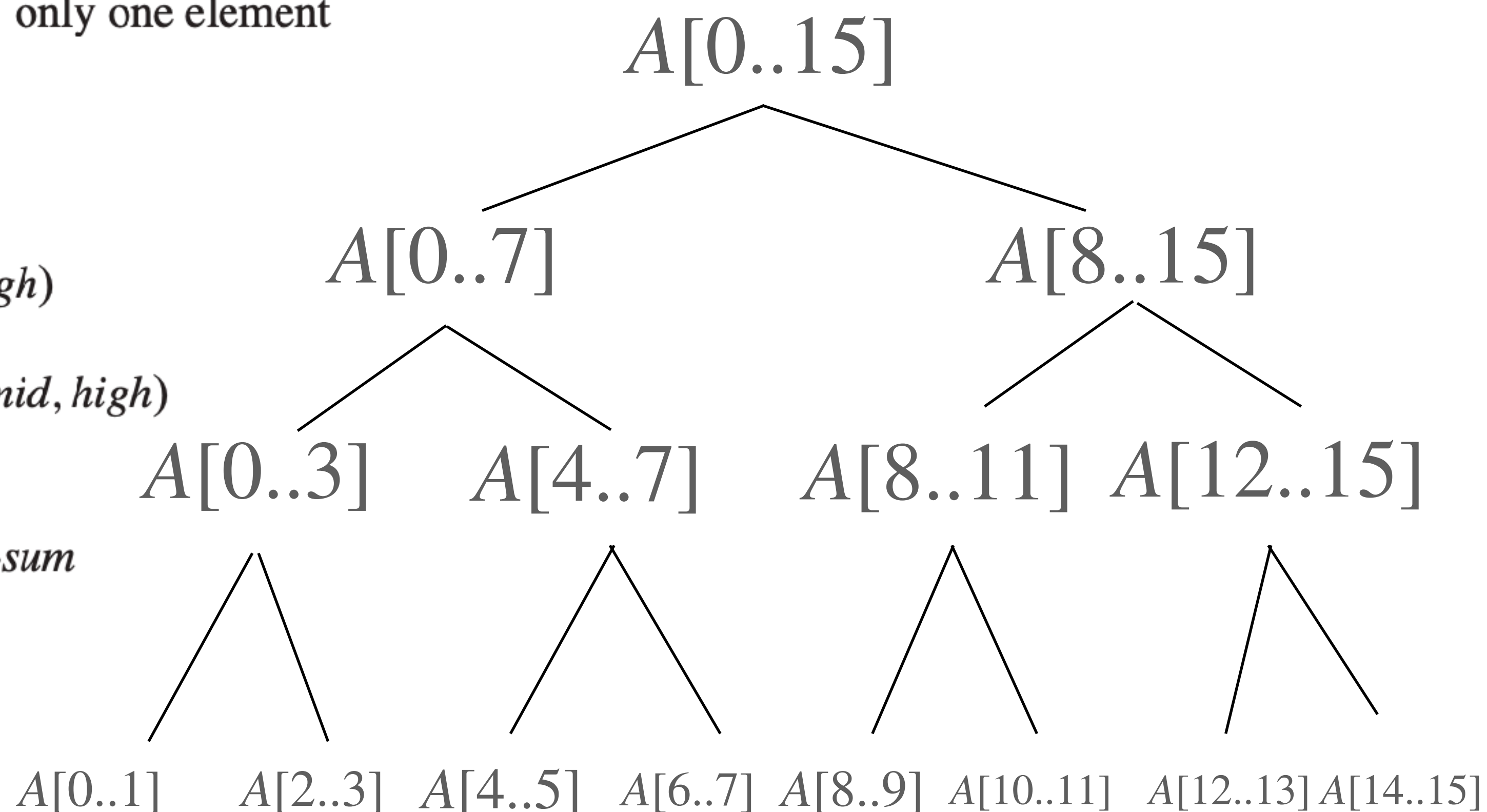
Divide-And-Conquer

What is the complexity ?

- At each node, the cost is central maximal subarray computation plus the recursions.

- We assume calling the recursion is free.

- **Then finding the maximal subarray crossing the middle point is the main cost ?**

$A[0..15]$

$A[0..7]$ $A[8..15]$

$A[0..3]$ $A[4..7]$ $A[8..11]$ $A[12..15]$

$A[0..1]$ $A[2..3]$ $A[4..5]$ $A[6..7]$ $A[8..9]$ $A[10..11]$ $A[12..13]$ $A[14..15]$

Divide-And-Conquer

FIND-MAX-CROSSING-SUBARRAY($A, low, mid, high$)

```
1   left-sum = −∞
2   sum = 0
3   for i = mid downto low
4       sum = sum + A[i]
5       if sum > left-sum
6           left-sum = sum
7           max-left = i
8   right-sum = −∞
9   sum = 0
10  for j = mid + 1 to high
11      sum = sum + A[j]
12      if sum > right-sum
13          right-sum = sum
14          max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

L   i   M   j   R

$A[i, M]$

Left part

$A[M + 1, j]$

Right part

- The central maximal subarray has a left part A[i,M] and a right part A[M+1,j].
- Would it be possible to find a better subarray $A[i', M]$ for $L \leq i' < i$???
- Would it be possible to find a better subarray $A[M + 1, j']$ for $j < j' \leq R$???
- **Therefore, detecting the central maximal subarray is $O(n)$?**
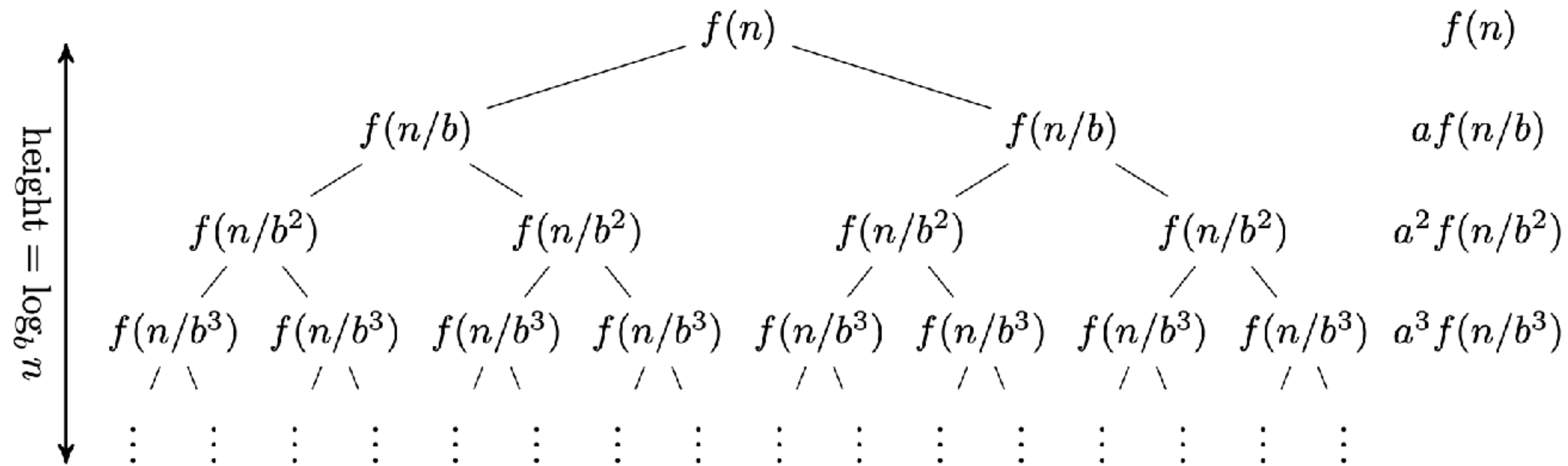
Divide-And-Conquer

What is the complexity ?

- At each node, the cost is central maximal subarray computation plus the recursions (that are free).

- **Total cost of maximal subarray detection is** $O(n \log n)$ **with the recursion.**
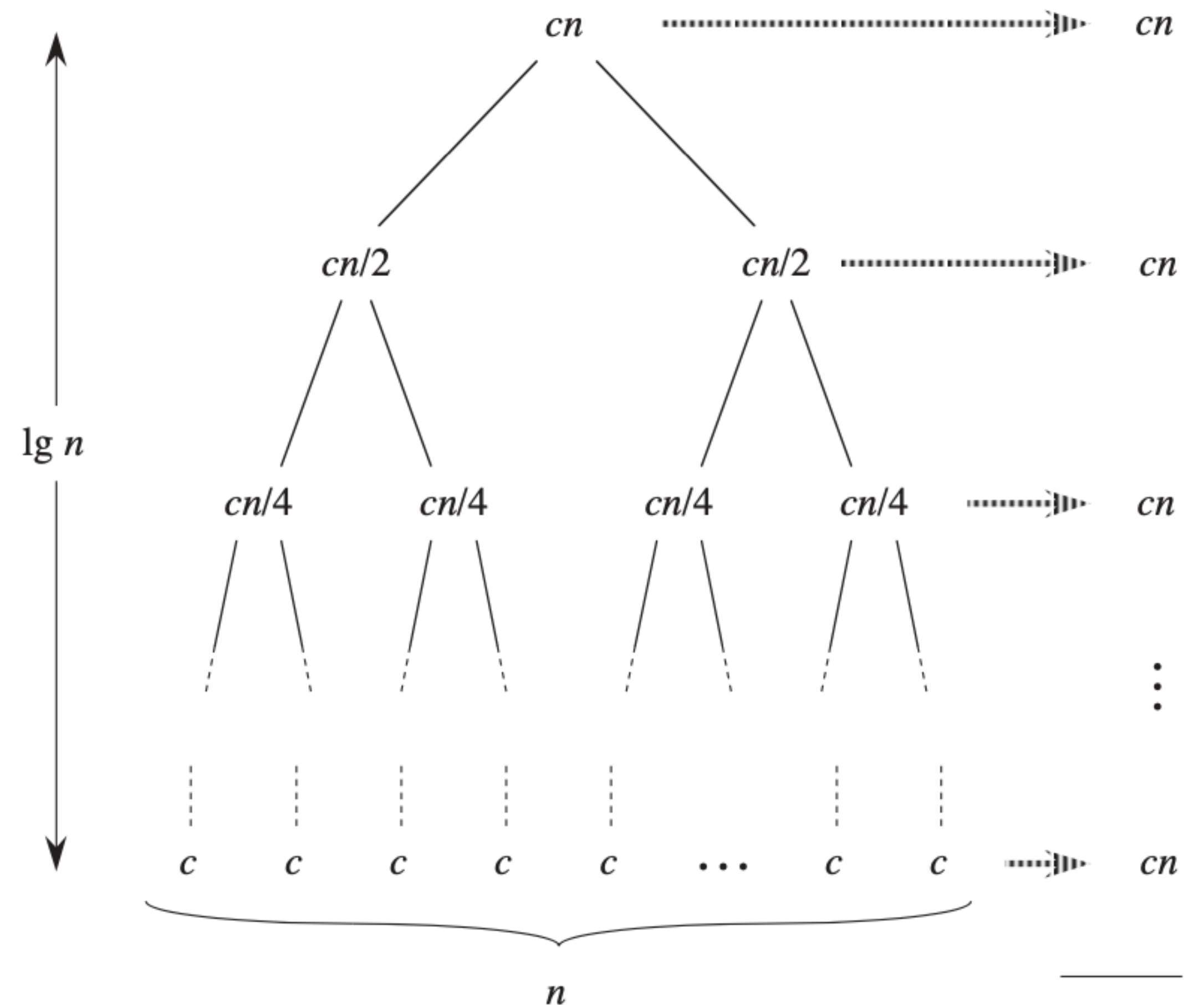
Divide-and-Conquer

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

- Recursively split problem of size n into problems of size n/b until the problem size reaches the unit size 1.
- We can visualize it with a recursion tree that shows the resursion $T(n) = aT(n/b) + f(n)$.
- On this tree each node includes a cost that is required to be done to join the parent. For instance, the cost of finding the maximal subarray crossing the central point in our example.

# Solution of Divide-and-Conquer Type Recursions

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

Master Method:

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

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

You can check the proof from Cormen's book.

- In our example $T(n) = 2T(n/2) + f(n)$, $f(n) \in \Theta(n)$ is the cost of detecting the central maximal subarray (why $\Theta(n)$?).
- Therefore, $T(n) = \Theta(n \log n)$ due to case 2 with $a = 2$, $b = 2$.

Master Theorem
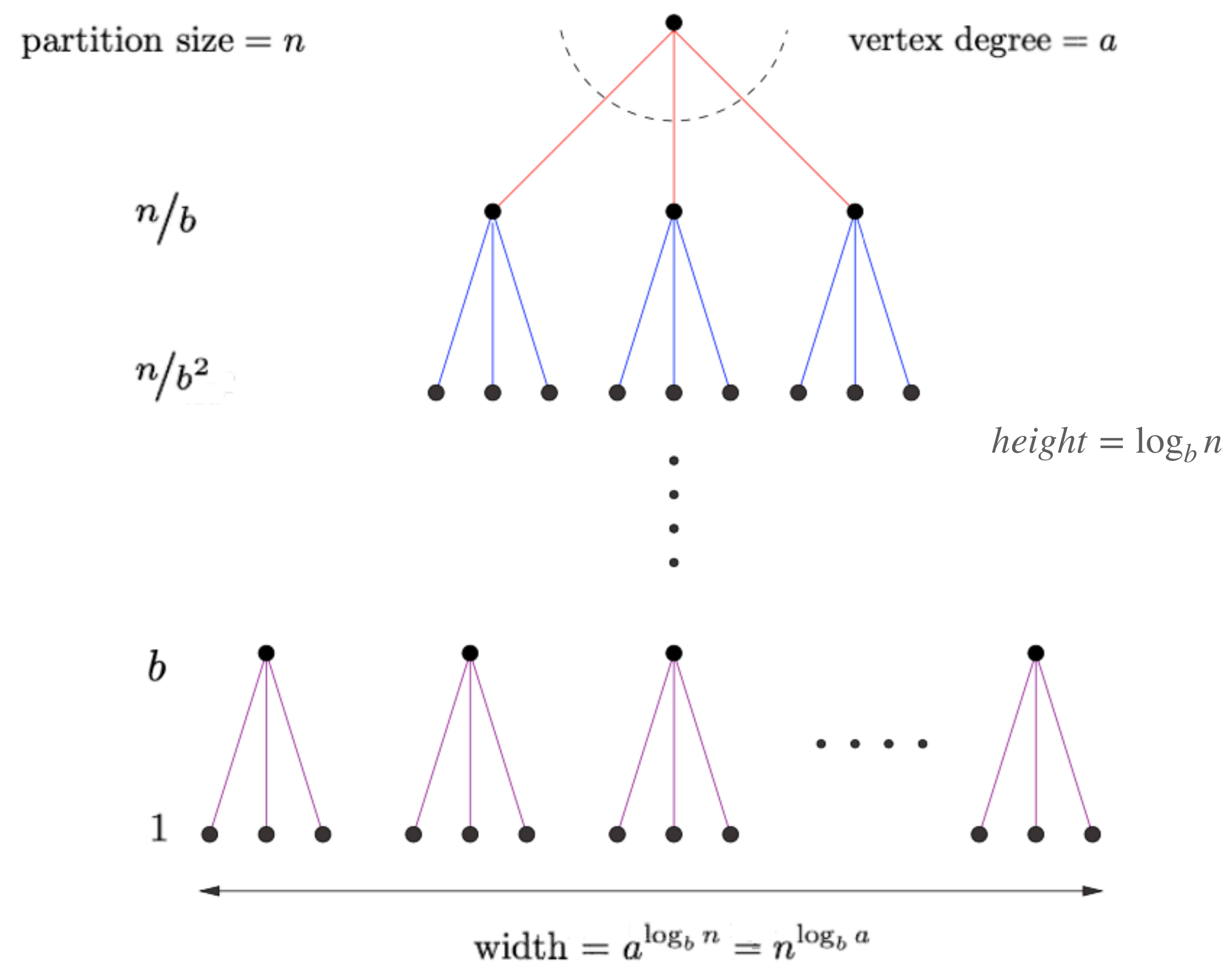
Case 1: Too many leaves
Case 2: Equal amount of work at each level
Case 3: Aggregating the nodes to the parent
   dominates everything

At each case compare $f(n)$ with $n^{\log_b a}$.
Whichever dominates determine the solution

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

3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \le cf(n)$ for some $c < 1$, then $T(n) = \Theta(f(n))$.

partition size $= n$ vertex degree $= a$

n/b

n/b^2

$height = \log_b n$

b

1

$\text{width} = a^{\log_b n} = n^{\log_b a}$

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

Fast n-bit Multiplication
Karatsuba Algorithm

Multiplying two n-digit numbers takes $O(n^2)$ time with the classic way since it takes n^2 multiplication and $O(n)$ additions.

$$a_3a_2a_1$$
$$\times \quad b_3b_2b_1$$
$$\overline{}$$
$$b_1 \times a_3a_2a_1$$
$$b_20 \times a_3a_2a_1$$
$$+ \quad b_300 \times a_3a_2a_1$$
$$\overline{}$$

$$
\begin{aligned}
1234 \times 4567 \; = \; & (7 \times 1234) + \\
& (60 \times 1234) + \\
& (500 \times 1234) + \\
& (4000 \times 1234) = \; 5635678
\end{aligned}
$$

Would it be possible to achieve it with less multiplications?

Fast n-bit Multiplication
Karatsuba Algorithm

Let's split each n-digit into two n/2 digit numbers and assume $w = 10^{(n/2)}$.

$A = 1234 \rightarrow (a_1 = 12, a_0 = 34) : 1234 = a_1 \cdot w + a_0$ $B = 4567 \rightarrow (b_1 = 45, b_0 = 67) : 4567 = b_1 \cdot w + b_0$

$$A \cdot B = (a_1 w + a_0)(b_1 w + b_0) = a_1 b_1 w^2 + a_1 b_0 w + a_0 b_1 w + a_0 b_0$$

- Multiplications with w are simply padding with zeros, so doesn't matter.

- Now we need 4 $(n/2)$ multiplications with $f(n) \in O(n)$ additions. Therefore, according to first case (why?) of master theorem:

$$T(n) = 4T(n/2) + O(n) \rightarrow f(n) \in O(n^{\log_2 4 - \epsilon}) \Rightarrow T(n) \in \Theta(n^2)$$

No improvement over the n-digit case.

Fast n-digit Multiplication

Karatsuba Algorithm

- Kolmogorov conjectured n-digit multiplication is $\Omega(n^2)$ time. However, Karatsuba showed its possible by computing the product differently, which makes it in $\Theta(n^{1.585})$, although with a penalty of a bit more additions, not hurting the complexity.

$$q_0 = a_0 \cdot b_0 \qquad q_1 = (a_1 + a_0)(b_1 + b_0) \qquad q_2 = a_1 \cdot b_1$$

$$A \cdot B = q_0 + (q_1 - q_0 - q_2)w + q_2 w^2$$

According to the **first case** of the master theorem again

$$T(n) = 3T(n/2) + O(n) \rightarrow f(n) \in \theta(n^{\log_2 3 - \epsilon}) \Rightarrow T(n) \in \Theta(n^{1.585})$$

Reading assignment

- Read the recursion and divide-and-conquer chapters from the text books, particularly from Cormen and Skiena, I suggest. Also you can refer chapter 33.4 from Cormen for a deep explanation of closest pair problem.