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
    - Closest pair in 2-d space

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:

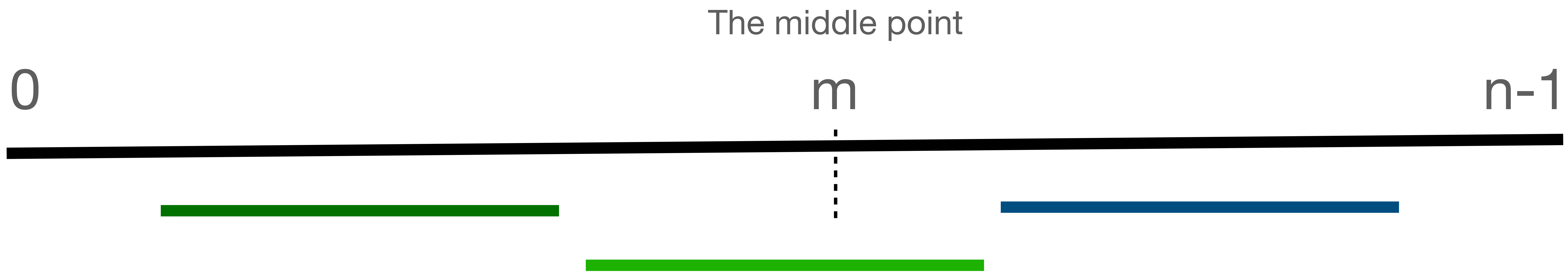
- How many different subarrays exist $\binom{n}{2} = n(n+1)/2 \in O(n^2)$.
- Generate all of them and choose the one with the maximum sum.

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.



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 left-half if $m < i \leq j < n$

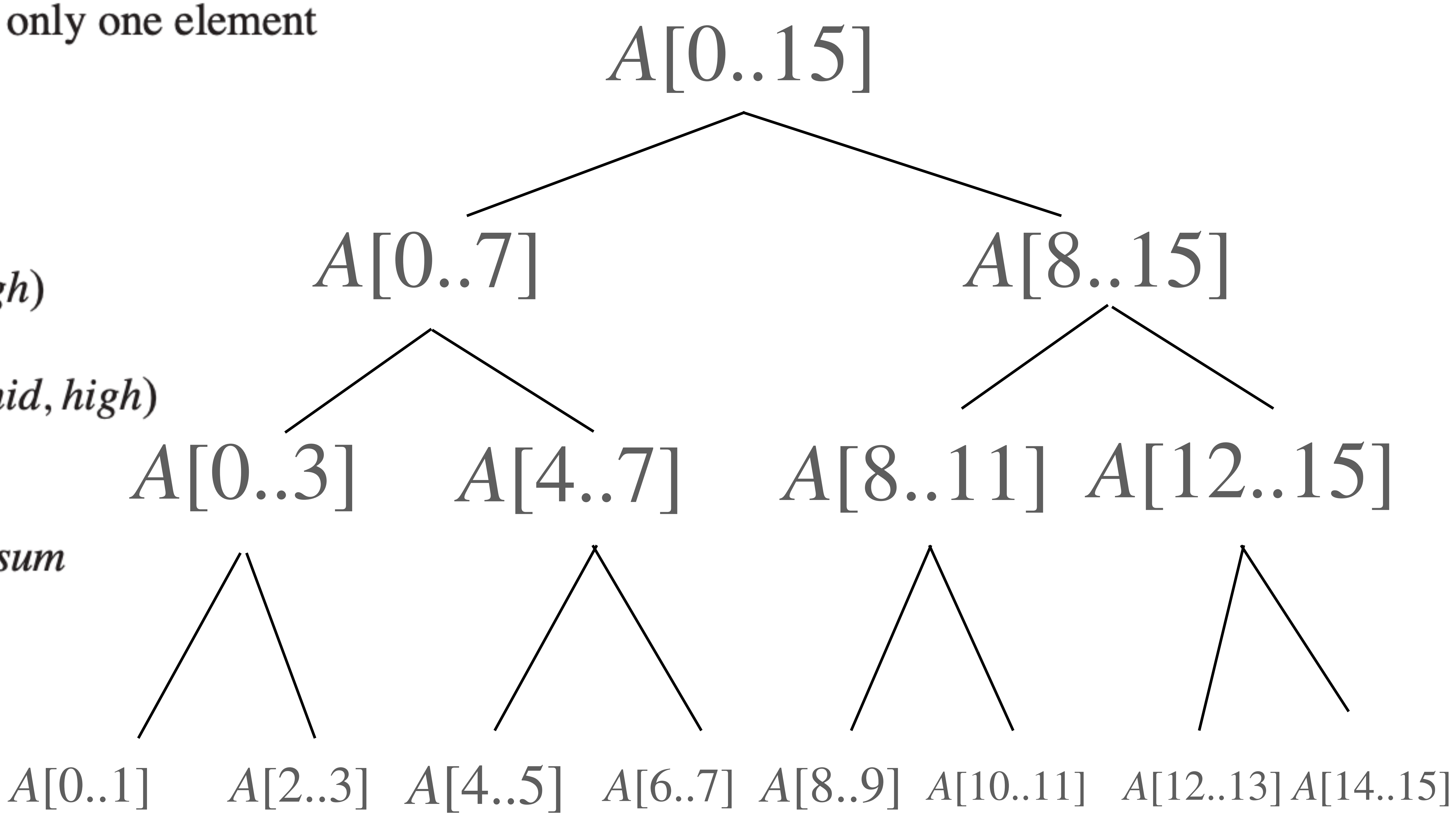
Maximum subarray $A[i,j]$ passes over m if $0 \leq i < m \leq j \leq n$

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

Divide-And-Conquer

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

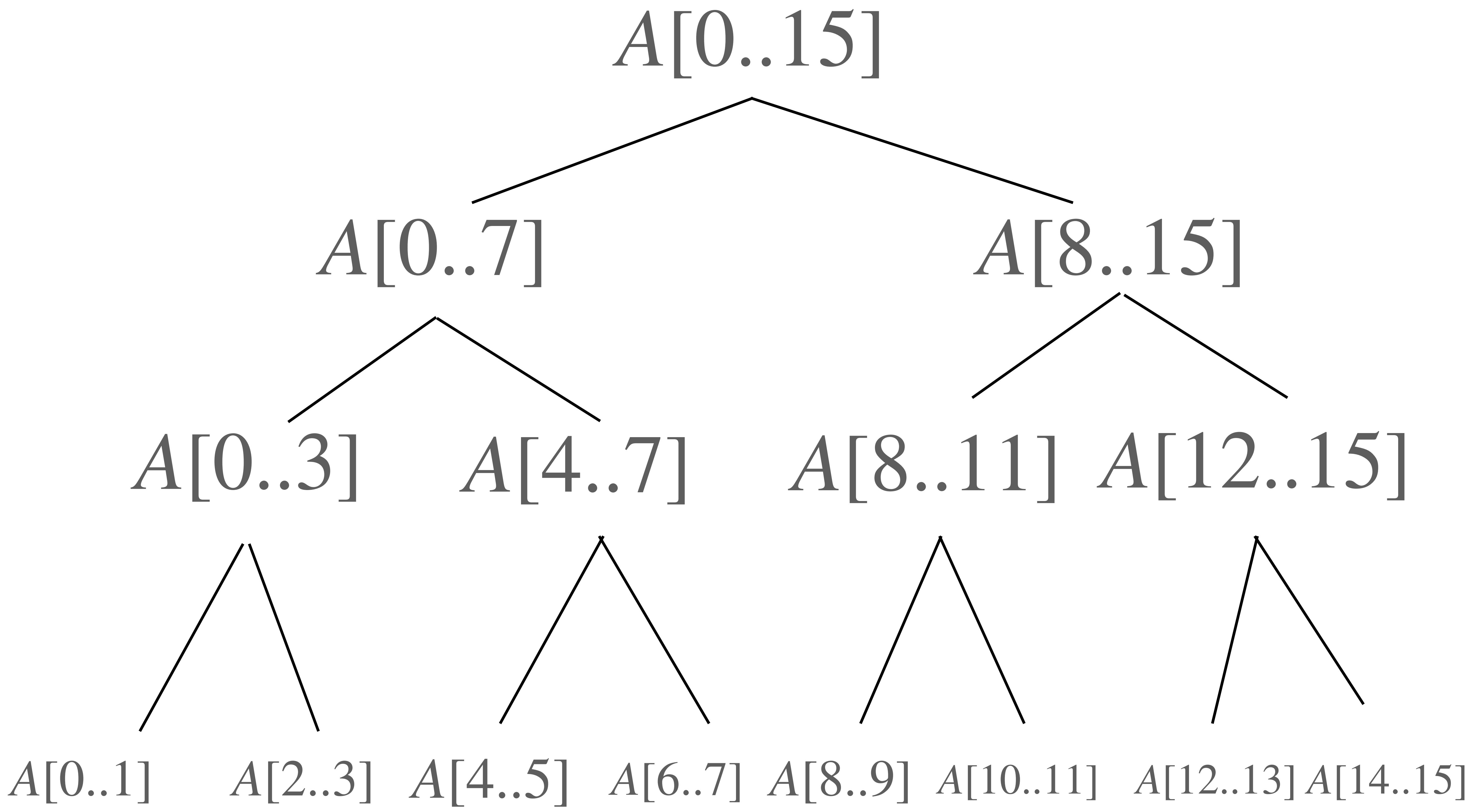
```
FIND-MAXIMUM-SUBARRAY(A, low, high)
if high == low
    return (low, high, A[low])          // base case: only one element
else mid = ⌊(low + high)/2⌋
    (left-low, left-high, left-sum) =
        FIND-MAXIMUM-SUBARRAY(A, low, mid)
    (right-low, right-high, right-sum) =
        FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
    (cross-low, cross-high, cross-sum) =
        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
        return (right-low, right-high, right-sum)
    else return (cross-low, cross-high, cross-sum)
```



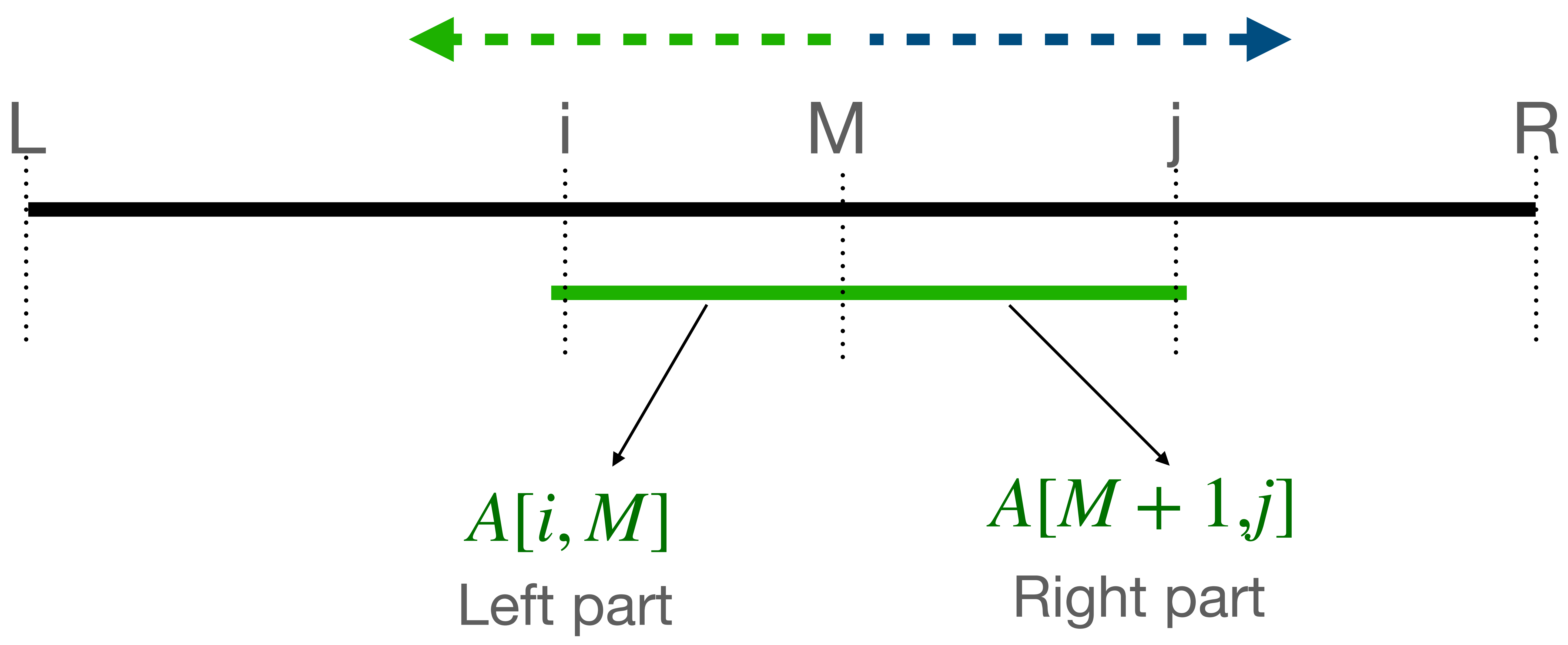
Divide-And-Conquer

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 is the main cost ?**



Divide-And-Conquer




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

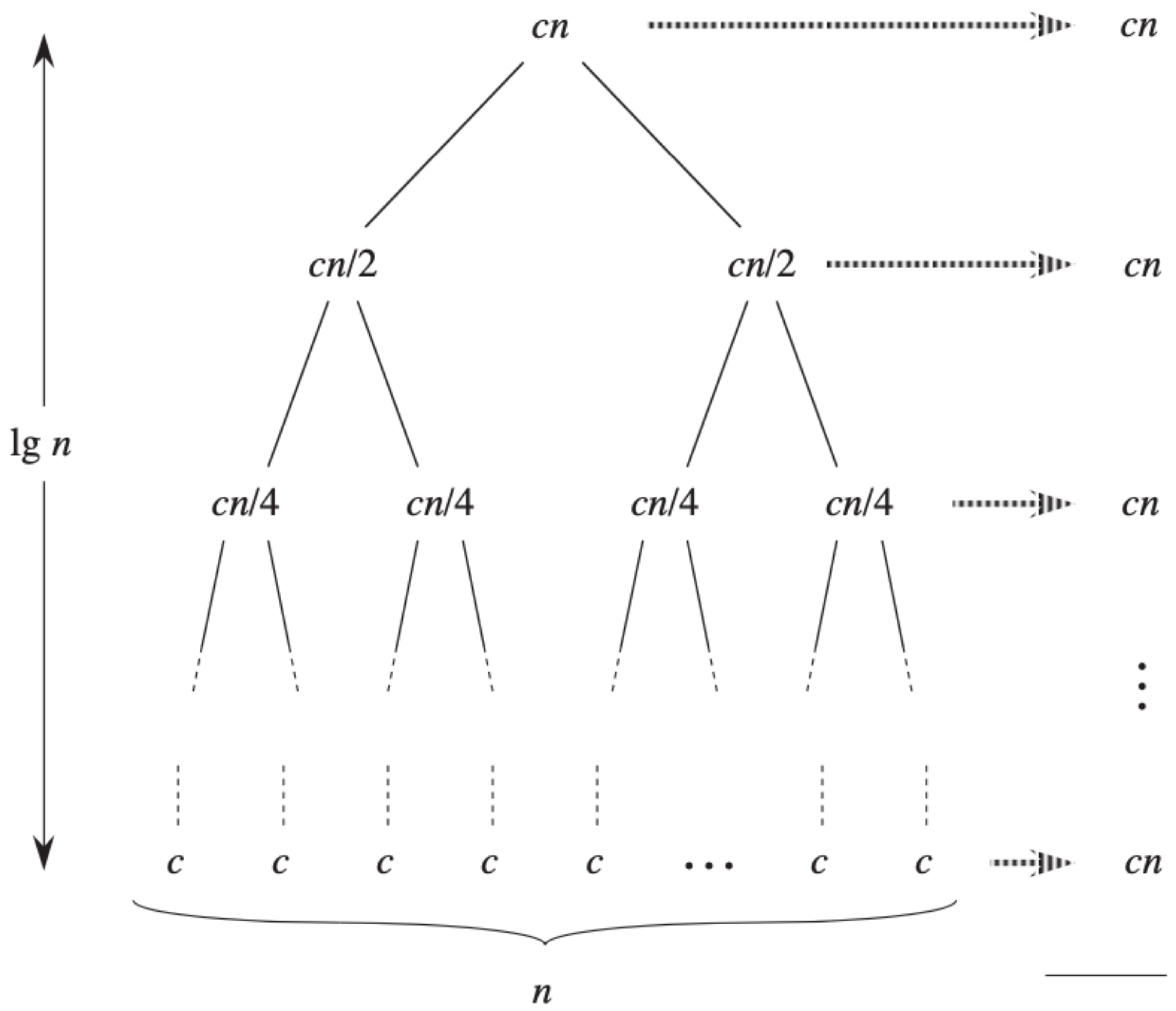
- The central maximal subarray has a left part A[i,M] and a right part A[M+1,j].
- Would it be possible to find a better subarray $A[i', M]$ for $L \leq i' < i$???
- Would it be possible to find a better subarray $A[M+1, j']$ for $j < j' \leq R$???
- **Therefore, detecting the central maximal subarray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

Divide-And-Conquer

What is the complexity ?

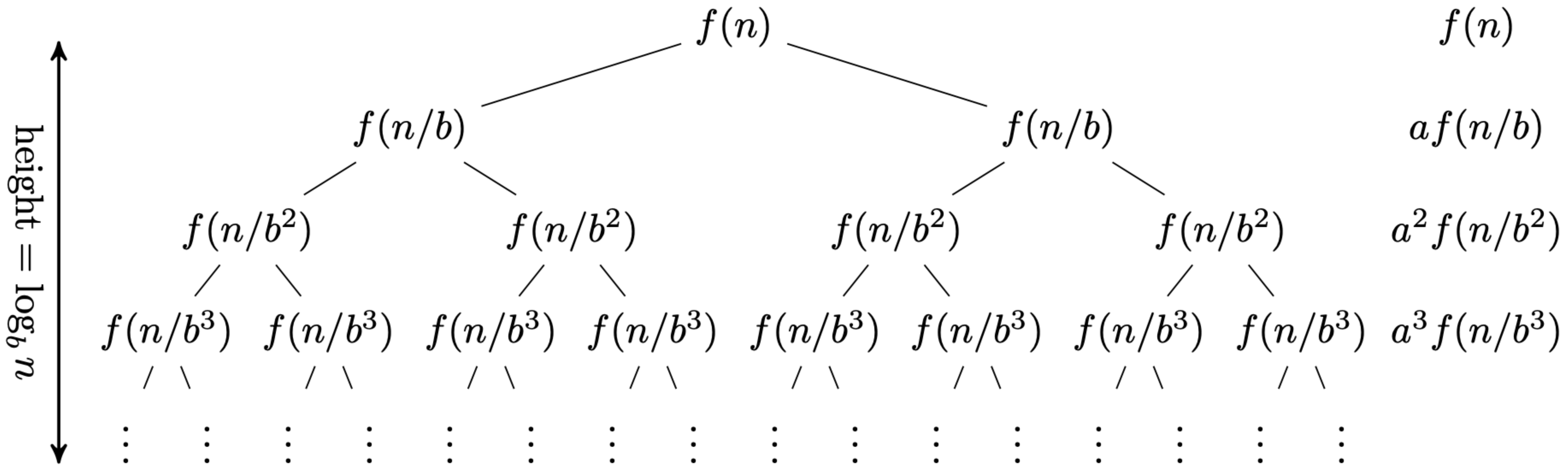
- At each node, the cost is central maximal subarray computation plus the recursions (that are free).

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



Divide-and-Conquer

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



https://www.cs.cornell.edu/courses/cs3110/2012sp/lectures/lec20-master/mm-proof.pdf

- 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

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

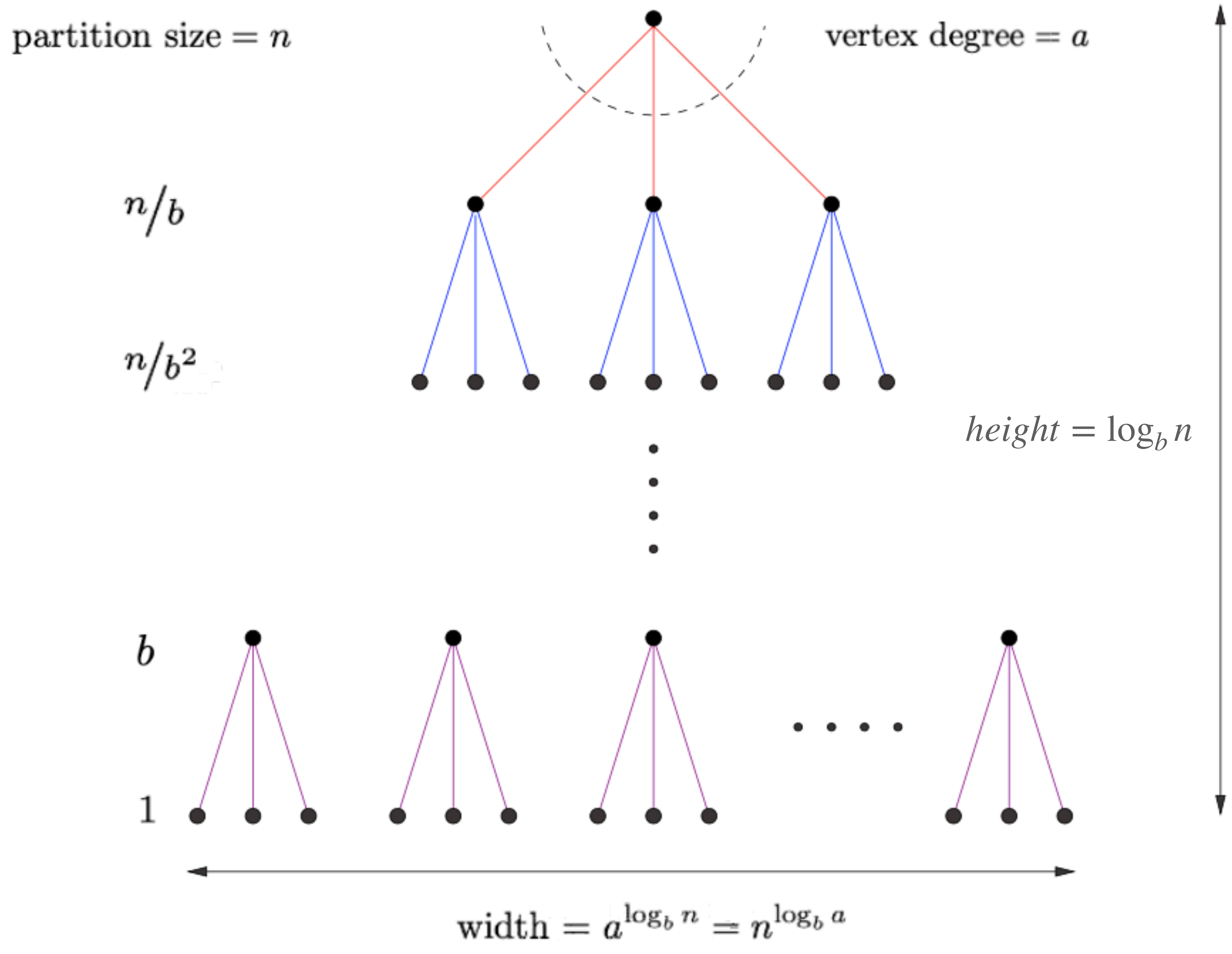
Master Method:

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 $af(n/b) \le cf(n)$ for some $c < 1$, then $T(n) = \Theta(f(n))$.

You can check the proof from Cormen's book.

- In our example $T(n) = T(n/2) + f(n)$, $f(n) \in \Theta(n)$ 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 $af(n/b) \leq cf(n)$ for some $c < 1$, then $T(n) = \Theta(f(n))$.

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

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.

$$
\begin{array}{r}
a_3a_2a_1 \\
\times \quad b_3b_2b_1 \\
\hline
b_1 \times a_3a_2a_1 \\
b_20 \times a_3a_2a_1 \\
+ \quad b_300 \times a_3a_2a_1 \\
\hline
\end{array}
$$

1234 x 4567 = (7 x1234) +
(60 x 1234) +
(500 x 1234) +
(4000 x 1234) = 5635678

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 $w = 10^{(n/2)}$.

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

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

- 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:

$$T(n) = 4T(n/2) + O(n) \to T(n) \in O(n^{\log_2 4}) = \Theta(n^2)$$

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.

$$
\boxed{
\begin{array}{c}
q_0 = a_0 \cdot b_0 \qquad q_1 = (a_1 + a_0)(b_1 + b_0) \qquad q_2 = a_1 \cdot b_1 \\
A \cdot B = q_0 + (q_1 - q_0 - q_2)w + q_2 w^2
\end{array}
}
$$

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

$$T(n) = 3T(n/2) + O(n) \to T(n) \in \theta(n^{\log_2 3}) = \Theta(n^{1.585})$$

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.