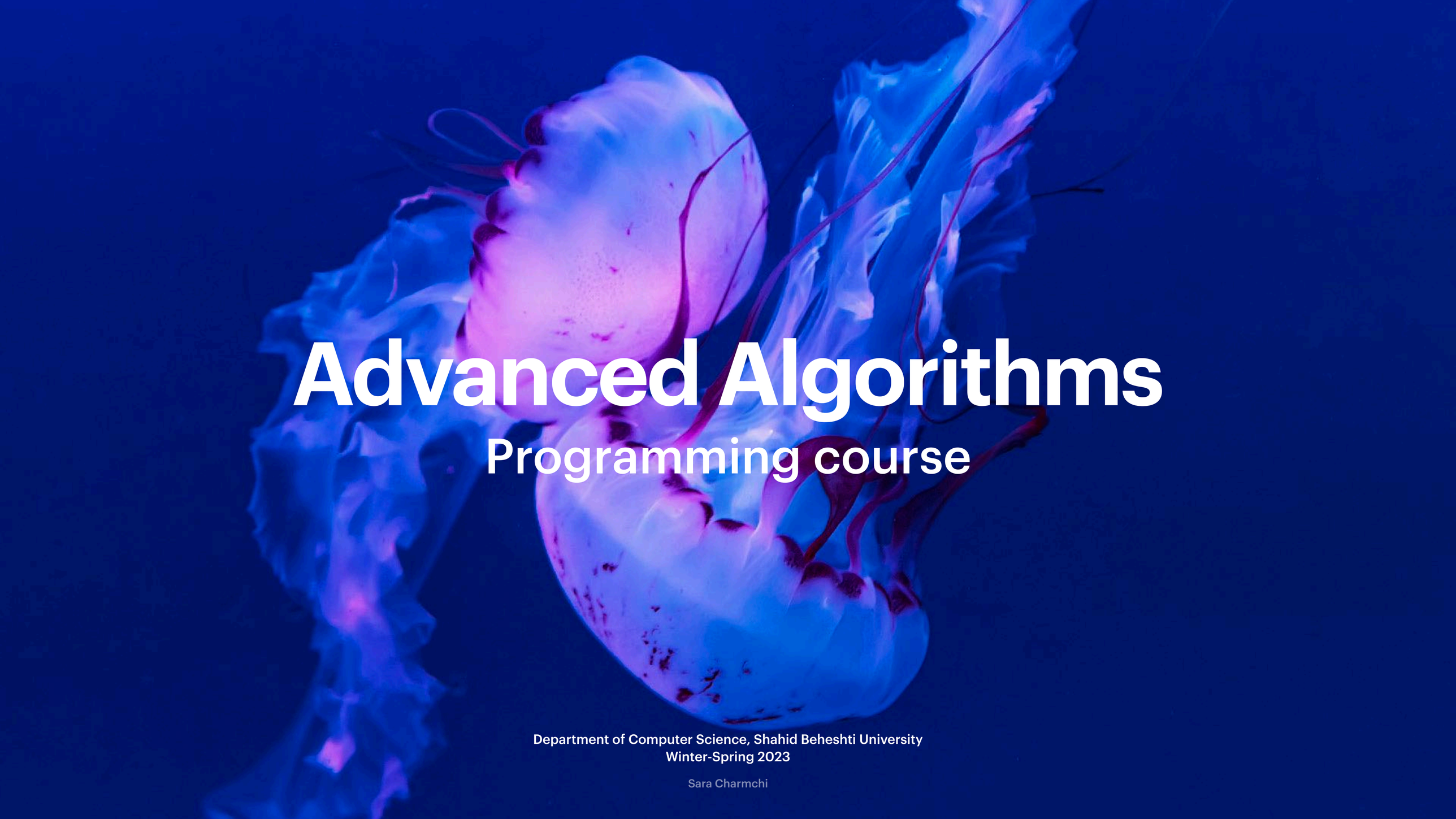


Advanced Algorithms

Programming course

Department of Computer Science, Shahid Beheshti University
Winter-Spring 2023

Sara Charmchi

Introduction

How to understand any algorithm?

- Understand what problem the algorithm solves
- Understand the logic behind the algorithm : global idea, how did we think about it, what elements it takes advantage from, what patterns it uses,...
- Understand the steps of the algorithm: pseudocode
- Visualize the process applied on some examples
- Understand the implementation: do by yourself But how?Gain knowledge in different data structures and algorithm design strategies
- Understand the Proof of correctness
- Understand the complexity analysis : time & space
- Understand when to use it : pros & cons ; Applications in real life

Strategies for Algorithm Design

Divide and Conquer

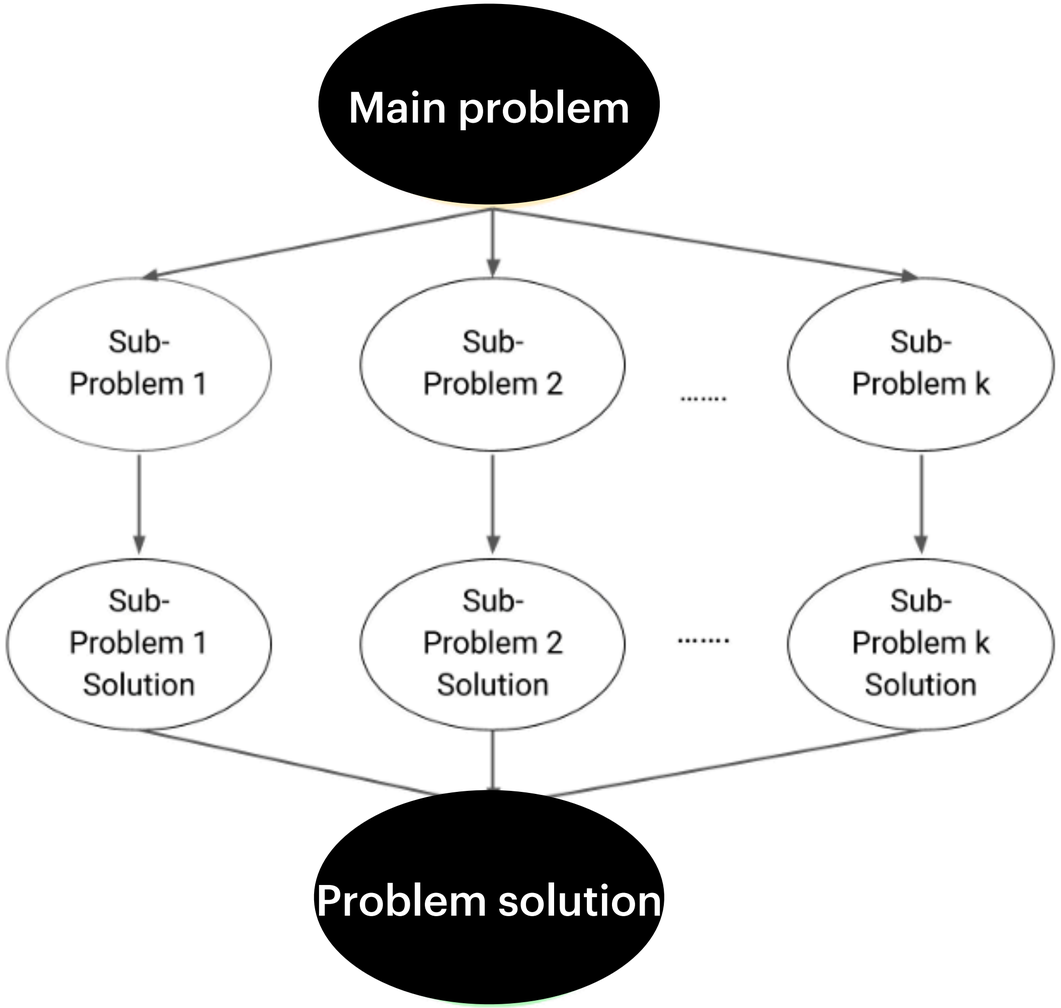
Divide & Conquer

Main Idea

Divide
Dividing the problem into smaller sub-problems

Conquer
Solving each sub-problems recursively

Combine
Combining sub-problem solutions to build the original problem solution



Divide & Conquer

Pseudocode

```
Divide_Conquer(problem P){
   if Small(P) return SimpleSolution(P);
   else {

     divide P into smaller instances P_1, P_2, ..., P_k, k≥2;

     Apply Divide_Conquer to each of these subproblems;

      return Combine(Divide_Conque(P_1), Divide_Conque(P_2),...,
      Divide_Conque(P_k));
   }
}
```

Divide & Conquer

Main Problems

- Binary search
- Finding maximum and minimum
- Merge sort
- Quick sort
- Strassen’s matrix multiplication

You should know how to write recursive algorithms/functions & analyze their complexity

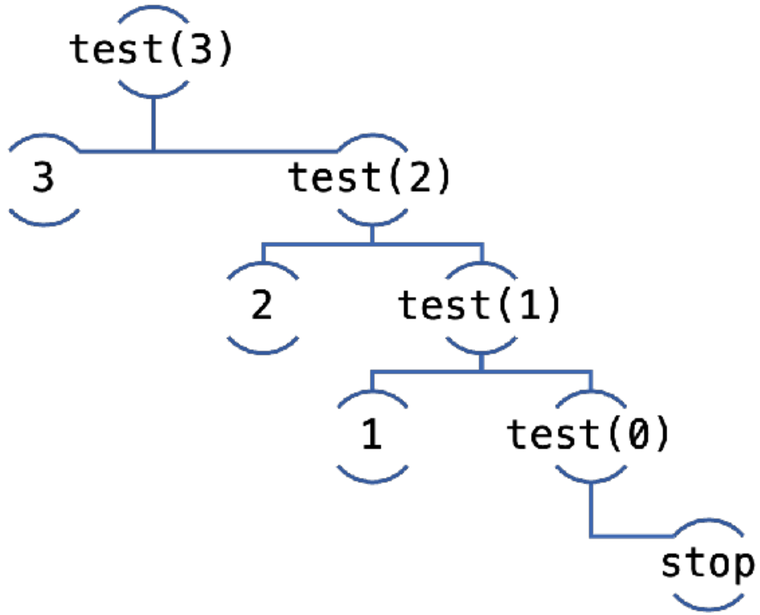
Review : Recurrence Relations

Decreasing Functions : recursive tree

```
void test(int n):
{
    if (n > 0 ):
 {
     print(n)
     test(n-1)
 }
}
```

let's call test(n) for n=3 →

we call this tracing tree or recursive tree



let's check its time complexity for n=3 :

how many times it is printing values? 3

how about recursive calls? 3 + 1

generalize to n →

pass n —> n+1 calls

time complexity depends on recursive calls

f(n) = n+1 —> Time complexity : O(n)

Review : Recurrence Relations

Decreasing Functions : prepare recurrence relation

```
void test(int n):
{
    if (n > 0 ):
 {
     print(n)
     test(n-1)
 }
}
```

→ T(n)

→ constant

→ one unit of time

→ T(n-1)

$T(n) = T(n-1) + 1$

Recurrence relation :

$$T(n) \begin{cases} 1 & n = 0 \\ T(n-1) + 1 & n > 0 \end{cases}$$

Review : Recurrence Relations

Decreasing Functions : solve recurrence relation : Substitution

$T(n) = T(n-1)+ 1 \longrightarrow T(n-1) = T(n-2)+ 1$

substitute T(n-1) :

$T(n) = [T(n-2)+ 1]+ 1 \longrightarrow T(n) = T(n-2)+ 2$

substitute again for n-2 :

$T(n) = [T(n-3)+ 1]+ 2 \longrightarrow T(n) = T(n-3)+ 3$

...

how long ? k :

$T(n) = T(n-k)+ k$

Review : Recurrence Relations

Decreasing Functions : solve recurrence relation : Substitution

$T(n) = T(n-k)+ k$

what is the smallest known value? T(n) for n=0

so continue substituting until n=0

assume we reach n=0 —> let's consider n-k = 0 —> n=k

$T(n) = T(n-n)+ n$

$T(n) = T(0)+ n$

$T(n) = n+1$ —> Time complexity : O(n)

Review : Recurrence Relations

Decreasing Functions : Master Theorem

$T(n) = T(n-1) + 1 \longrightarrow O(n)$

$T(n) = T(n-1) + n \longrightarrow O(n^2)$

$T(n) = T(n-1) + \log n \longrightarrow O(n \log n)$

$T(n) = 2\,T(n-1) + 1 \longrightarrow O(2^n)$

$T(n) = 3\,T(n-1) + 1 \longrightarrow O(3^n)$

$T(n) = 2\,T(n-1) + n \longrightarrow O(n2^n)$

$T(n) = a\,T(n-b) + f(n)$

$a > 0$

$b > 0$

$f(n) = O(n^k),\ k > 0$

Review : Recurrence Relations

Decreasing Functions : Master Theorem

$T(n) = T(n-1) + 1 \longrightarrow O(n)$

$T(n) = T(n-1) + n \longrightarrow O(n^2)$

$T(n) = T(n-1) + \log n \longrightarrow O(n \log n)$

$T(n) = 2\,T(n-1) + 1 \longrightarrow O(2^n)$

$T(n) = 3\,T(n-1) + 1 \longrightarrow O(3^n)$

$T(n) = 2\,T(n-1) + n \longrightarrow O(n2^n)$

$T(n) = a\,T(n-b) + f(n)$

$a > 0$

$b > 0$

$f(n) = O(n^k),\ k > 0$

Case I

$a = 1$

$O(n^{k+1})$

$O(n * f(n))$

Review : Recurrence Relations

Decreasing Functions : Master Theorem

$T(n) = T(n-1) + 1 \longrightarrow O(n)$

$T(n) = T(n-1) + n \longrightarrow O(n^2)$

$T(n) = T(n-1) + \log n \longrightarrow O(n \log n)$

$T(n) = 2\,T(n-1) + 1 \longrightarrow O(2^n)$

$T(n) = 3\,T(n-1) + 1 \longrightarrow O(3^n)$

$T(n) = 2\,T(n-1) + n \longrightarrow O(n2^n)$

$T(n) = a\,T(n-b) + f(n)$

$a > 0$

$b > 0$

$f(n) = O(n^k),\ k > 0$

Case II

$a > 1$

$O(n^k\, a^{n/b})$

$O(f(n) * a^{n/b})$

Review : Recurrence Relations

Decreasing Functions : Master Theorem

$T(n) = T(n-1) + 1 \longrightarrow O(n)$

$T(n) = T(n-1) + n \longrightarrow O(n^2)$

$T(n) = T(n-1) + \log n \longrightarrow O(n \log n)$

$T(n) = 2\,T(n-1) + 1 \longrightarrow O(2^n)$

$T(n) = 3\,T(n-1) + 1 \longrightarrow O(3^n)$

$T(n) = 2\,T(n-1) + n \longrightarrow O(n2^n)$

$T(n) = a\,T(n-b) + f(n)$

$a > 0$

$b > 0$

$f(n) = O(n^k),\ k > 0$

Case III

$a < 1$

$O(n^k)$

$O(f(n))$

Review : Recurrence Relations

Dividing Functions : prepare recurrence relation

```
void test(int n):
{
    if (n > 1 ):
 {
     print(n)
     test(n/2)
 }
}
```

→ T(n)

→ constant

→ one unit of time

→ T(n/2)

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

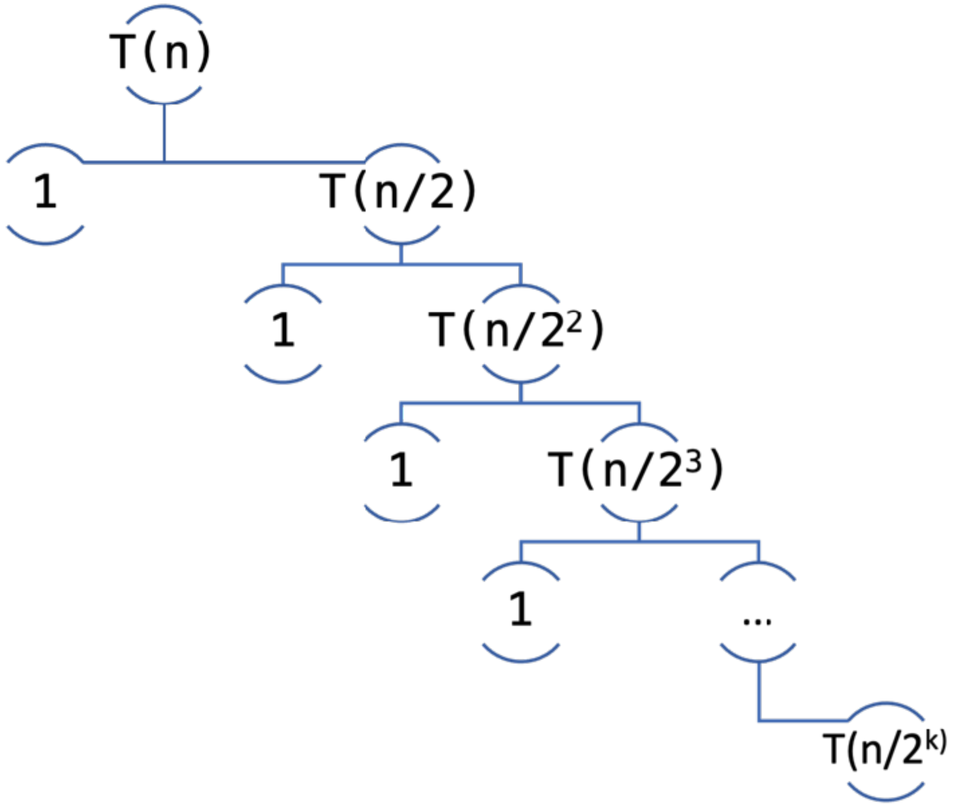
Recurrence relation :

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

Review : Recurrence Relations

Dividing Functions : recursive tree

```
void test(int n):
{
    if (n > 1 ):
 {
     print(n)
     test(n/2)
 }
}
```



stopping condition : n = 1 :

$n/2^k = 1 \rightarrow n = 2^k \rightarrow k = \log_2 n$

time complexity depends on number of steps : k steps ⟶ **Time complexity : O(logn)**

Review : Recurrence Relations

Dividing Functions : solve recurrence relation : Substitution

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

substitute T(n/2) :

$T(n) = [T(n/2^2)+ 1]+ 1 \longrightarrow T(n) = T(n/2^2)+ 2$

substitute again for $n/2^2$:

$T(n) = [T(n/2^3)+ 1]+ 2 \longrightarrow T(n) = T(n/2^3)+ 3$

...

how long ? k : $T(n) = T(n/2^k)+ k$

smallest known value :$n=1 \rightarrow n/2^k = 1 \rightarrow n = 2^k \rightarrow k = \log_2 n$

$T(n) = T(n=1)+ k \rightarrow T(n) = 1 + \log n$ ⟶ **Time complexity : $\Theta(\log n)$**

Review : Recurrence Relations

Dividing Functions : Master Theorem

$T(n) = a\,T(n/b) + \Theta(n^c \log^d n)$

$c < \log_b a \longrightarrow T(n) = \Theta(n^{\log_b a})$

$c = \log_b a \longrightarrow$

- $d > -1 \quad T(n) = \Theta(n^c \log^{d+1} n)$
- $d = -1 \quad T(n) = \Theta(nc\ \text{loglogn})$
- $d < -1 \quad T(n) = \Theta(n^c)$

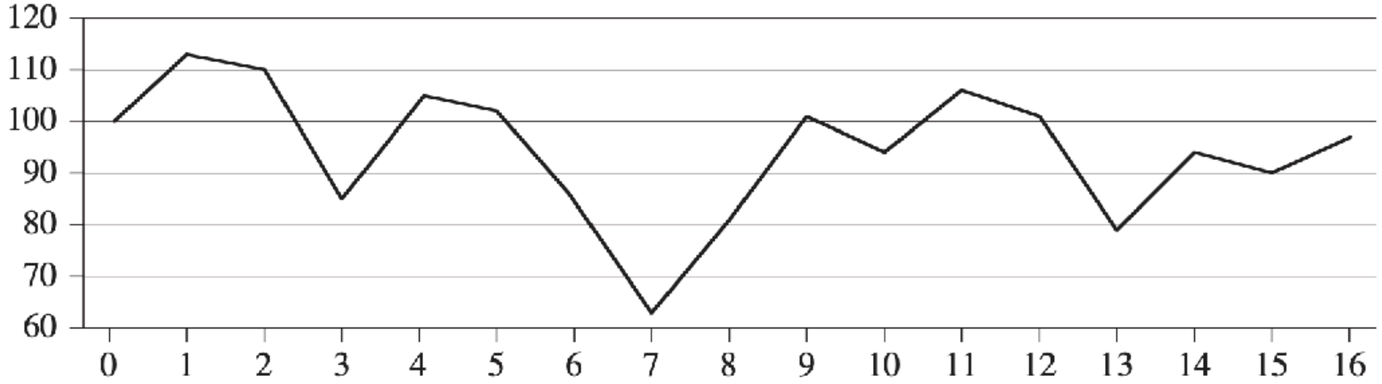
$c > \log_b a \longrightarrow$

- $d >= 0 \quad T(n) = \Theta(n^c \log^d n)$
- $d < 0 \quad T(n) = \Theta(n^c)$

The maximum-subarray problem

Definition

- Find the series of contiguous elements with the maximum sum in any given array



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

Figure 4.1 Information about the price of stock in the Volatile Chemical Corporation after the close of trading over a period of 17 days. The horizontal axis of the chart indicates the day, and the vertical axis shows the price. The bottom row of the table gives the change in price from the previous day.

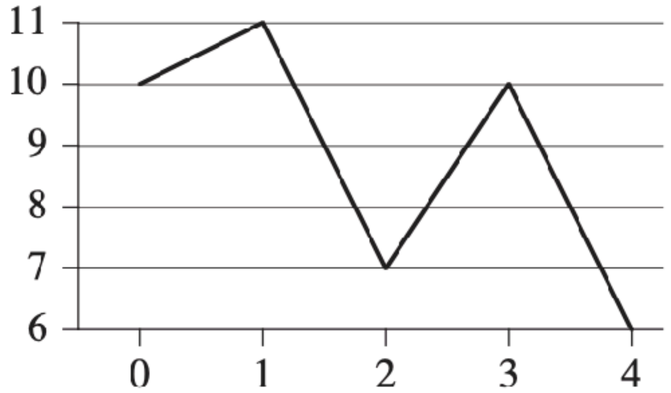
The maximum-subarray problem

Definition

- A possible strategy :

1. find the highest and lowest prices
2. work left from the highest price to find the lowest prior price, work right from the lowest price to find the highest later price
3. take the pair with the greater difference.

- How about this situation ?

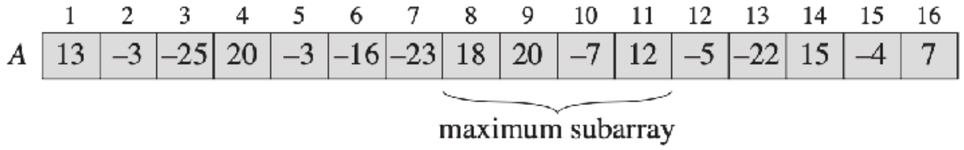


Day	0	1	2	3	4
Price	10	11	7	10	6
Change		1	−4	3	−4

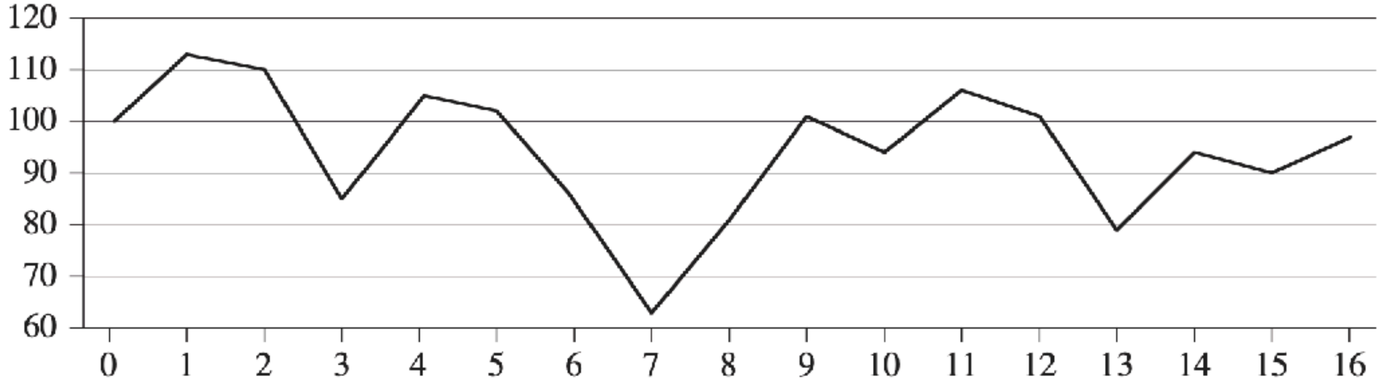
The maximum-subarray problem

Definition

- Find the series of contiguous elements with the maximum sum in any given array
- Brute-force solution; $O(n^2)$



- Any better idea?

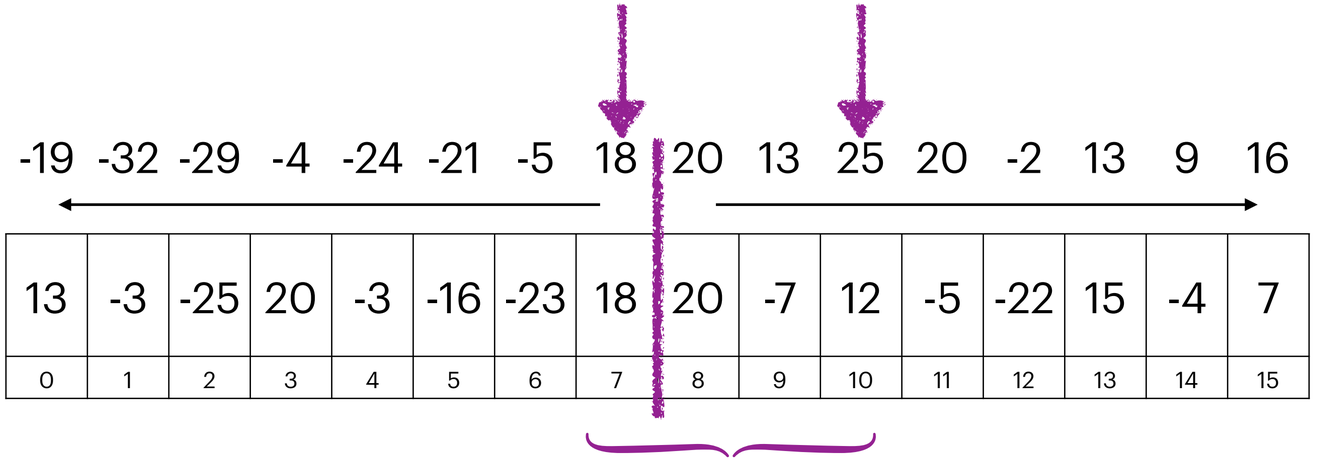


Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

Figure 4.1 Information about the price of stock in the Volatile Chemical Corporation after the close of trading over a period of 17 days. The horizontal axis of the chart indicates the day, and the vertical axis shows the price. The bottom row of the table gives the change in price from the previous day.

The maximum-subarray problem

Divide & Conquer Approach



- entirely in the subarray $A[low\,..\,mid]$, so that $low \le i \le j \le mid$,
- entirely in the subarray $A[mid+1\,..\,high]$, so that $mid < i \le j \le high$, or
- crossing the midpoint, so that $low \le i \le mid < j \le high$.

The maximum-subarray problem

Divide & Conquer Approach

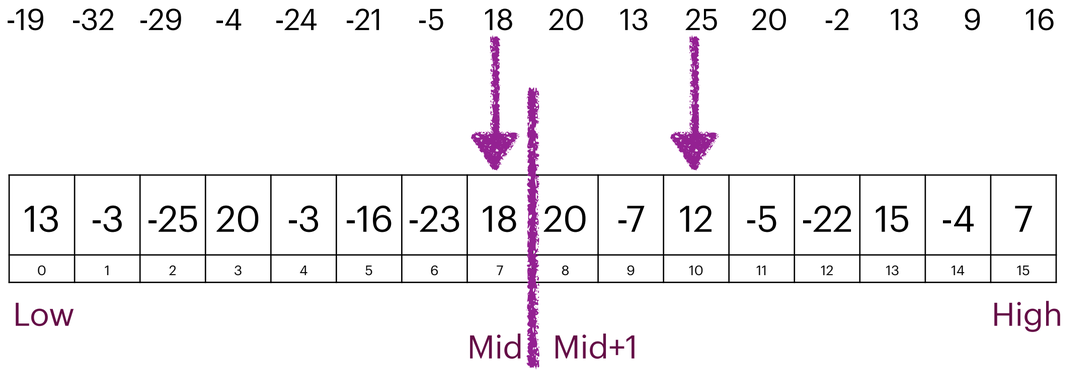
```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 1  left-sum = −∞
 2  sum = 0
 3  for i = mid downto low
 4       sum = sum + A[i]
 5       if sum > left-sum
 6            left-sum = sum
 7            max-left = i
 8  right-sum = −∞
 9  sum = 0
10  for j = mid + 1 to high
11       sum = sum + A[j]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

Find maximum subarray of left half

Every subarray : A [i,..., mid]

Find maximum subarray of right half

Every subarray : A [mid+1,...,j]



The maximum-subarray problem

Divide & Conquer Approach

```
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)
```

size of array : A [low , ... , high] : n —> n = high - low + 1

each iteration : Θ (1) ;
number of iterations : mid - low +1

each iteration : Θ (1) ;
number of iterations : high - mid

Time complexity :Θ (n)

The maximum-subarray problem

Divide & Conquer Approach

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
 1  if high == low
 2      return (low, high, A[low])          // base case: only one element
 3  else mid = ⌊(low + high)/2⌋                                   ──→ Divide
 4      (left-low, left-high, left-sum) =
            FIND-MAXIMUM-SUBARRAY(A, low, mid)
 5      (right-low, right-high, right-sum) =                      } Conquer
            FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
 6      (cross-low, cross-high, cross-sum) =
            FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 7      if left-sum ≥ right-sum and left-sum ≥ cross-sum
 8          return (left-low, left-high, left-sum)
 9      elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10          return (right-low, right-high, right-sum)
11      else return (cross-low, cross-high, cross-sum)
```

Combine

The maximum-subarray problem

Divide & Conquer Approach

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
1  if high == low
2      return (low, high, A[low])          // base case: only one element
3  else mid = ⌊(low + high)/2⌋
4      (left-low, left-high, left-sum) =
           FIND-MAXIMUM-SUBARRAY(A, low, mid)
5      (right-low, right-high, right-sum) =
           FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
6      (cross-low, cross-high, cross-sum) =
           FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
7      if left-sum ≥ right-sum and left-sum ≥ cross-sum
8          return (left-low, left-high, left-sum)
9      elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10         return (right-low, right-high, right-sum)
11     else return (cross-low, cross-high, cross-sum)
```

condition takes constant unit of time

// base case: only one element → T(1) = Θ (1)

condition takes constant unit of time

n/2 elements —> T(n/2)

n/2 elements —> T(n/2)

from previous slides: Θ (n)

condition takes constant unit of time

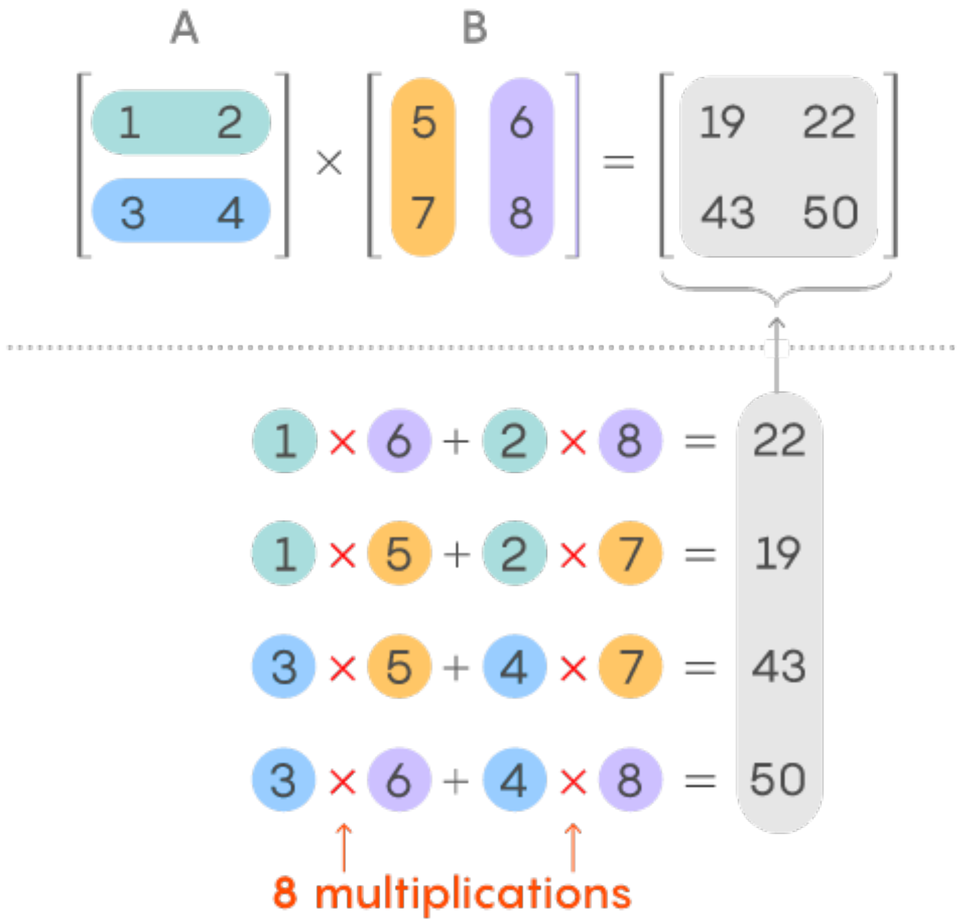
$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Master Theorem

T(n) = Θ (n log n)

Matrix multiplication

Problem Definition



Matrix multiplication

Problem Definition

$$A_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

$$B_{n,p} = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,p} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \cdots & b_{n,p} \end{pmatrix}$$

$$\Rightarrow C = AB = C_{m,p} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,p} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m,1} & c_{m,2} & \cdots & b_{m,p} \end{pmatrix}$$

where $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + ... + a_{in}b_{nj}$

Matrix multiplication

Brute-force approach

```
SQUARE-MATRIX-MULTIPLY(A, B)
1  n = A.rows
2  let C be a new n × 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 · b_kj
8  return C
```

$\Theta(n^3)$

Matrix multiplication
Divide & Conquer approach

- Which size of matrix is small enough for divide & conquer strategy?

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} * \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

A *B* *C*

*2*2* *2*2* *2*2*

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} , \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} , \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} , \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} . \end{aligned}$$

Matrix multiplication

Divide & Conquer approach

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{12} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} * \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} =$$

A

*4*4*

/2

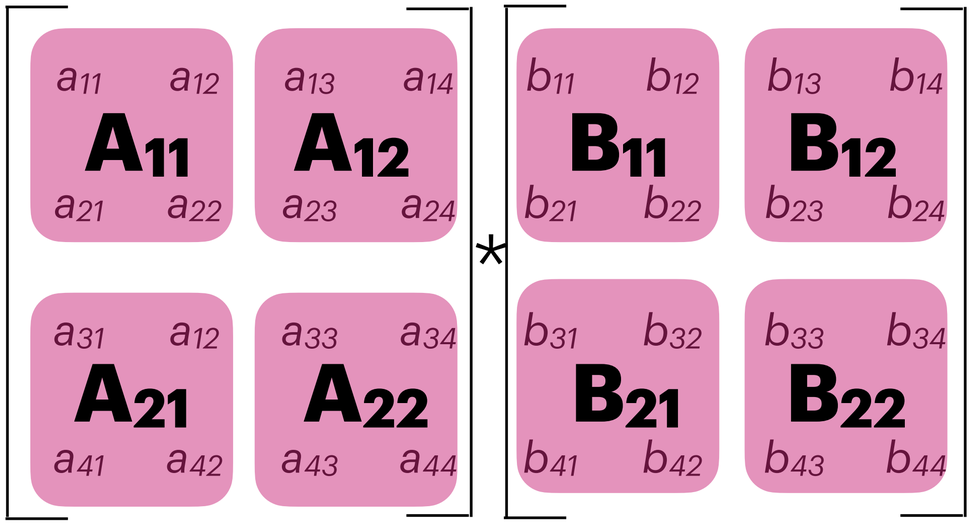
B

*4*4*

/2

Matrix multiplication

Divide & Conquer approach



$$C_{11} = A_{11} * B_{11} + A_{12} * B_{21}$$

$$C_{12} = A_{11} * B_{12} + A_{12} * B_{22}$$

$$C_{21} = A_{21} * B_{11} + A_{22} * B_{21}$$

$$C_{22} = A_{21} * B_{12} + A_{22} * B_{22}$$

Not elements
:Matrices
but act as elements

```
Algorithm MM (A,B,n):
{
    if (n<=2)
  {
      C : 4 formulas for c_11,c_12,c_21,c_22
    //follow the rule of matrix multiplication
      (2*2)

  }
  else
  {
      mid ... n/2
    C_11 = MM (A_11,B_11,n/2) + MM (A_12,B_21,n/2)
    C_12 = MM (A_11,B_12,n/2) + MM (A_12,B_22,n/2)
    C_21 = MM (A_21,B_11,n/2) + MM (A_22,B_21,n/2)
    C_22 = MM (A_21,B_12,n/2) + MM (A_22,B_22,n/2)
  }
  return C

}
```

Matrix addition

Matrix multiplication

Divide & Conquer approach

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
1  n = A.rows
2  let C be a new n × n matrix
3  if n == 1
4       c11 = a11 · b11
5  else partition A, B, and C
6       C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
            + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
7       C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
            + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
8       C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
            + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
9       C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
            + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
10 return C
```

T(1) = Θ (1)

by using index calculations :Θ (1)

each recursive call : n —> n/2 : T(n) —> T(n/2)

each matrix addition : Θ (n^2)

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

Master Theorem

T(n) = Θ (n^3)

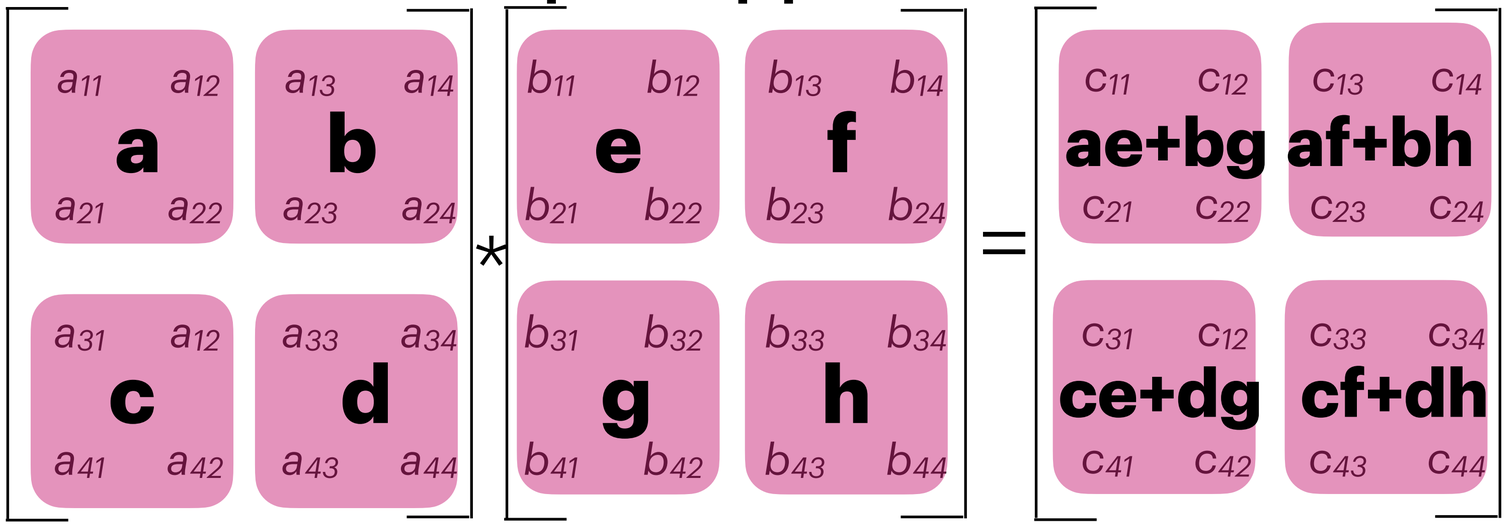
Strassen’s matrix multiplication

Divide & Conquer approach

- Reduce time complexity with a novel technique : instead of performing eight recursive multiplications of n/2 × n/2 matrices, it performs only seven.
- Is there any smart way to rewrite the calculations?

Strassen's matrix multiplication

Divide & Conquer approach



ae +bg = ?
(a+d)(e+h)+d(g-e)-(a+b)h+(b-d)(g+h)
=ae+ah+de+dh+dg-de-ah-bh+bg+bh-dg-dh
=ae+ah+de+dh+dg-de-ah-bh+bg+bh-dg-dh
=ae +bg

af+bh=?
a(f-h)+(a+b)h
= af-ah+ah+bh
= af-ah+ah+bh
=af+bh

ce+dg=?
(c+d)e+d(g-e)
= ce+de+dg-de
= ce+de+dg-de
=ce+dg

cf+dh = ?
a(f-h)+(a+d)(e+h)-(c+d)e-(a-c)(e+f)
=af-ah+ae+ah+de+dh-ce-de-ae-af+ce+cf
=af-ah+ae+ah+de+dh-ce-de-ae-af+ce+cf
=cf+dh

Strassen's matrix multiplication

Divide & Conquer approach

P_1 P_2 P_3 P_4

ae +bg = (a+d)(e+h)+d(g-e)-(a+b)h+(b-d)(g+h)

af+bh=a(f-h)+(a+b)h

ce+dg=(c+d)e+d(g-e)

cf+dh = a(f-h)+(a+d)(e+h)-(c+d)e-(a-c)(e+f)

P_5 P_6 P_7

C_{11} = ae + bg = $P_1 + P_2 - P_2 + P_2$

C_{12} = af + bh = $P_5 + P_3$

C_{21} = ce + dg = $P_6 + P_2$

C_{22} = cf + dh = $P_5 + P_1 - P_6 - P_7$

Strassen's matrix multiplication

Divide & Conquer approach

- Reduce time complexity with a novel technique : instead of performing eight recursive multiplications of n/2 × n/2 matrices, it performs only seven.

$$
\begin{aligned}
S_1 &= B_{12} - B_{22} , \\
S_2 &= A_{11} + A_{12} , \\
S_3 &= A_{21} + A_{22} , \\
S_4 &= B_{21} - B_{11} , \\
S_5 &= A_{11} + A_{22} , \\
S_6 &= B_{11} + B_{22} , \\
S_7 &= A_{12} - A_{22} , \\
S_8 &= B_{21} + B_{22} , \\
S_9 &= A_{11} - A_{21} , \\
S_{10} &= B_{11} + B_{12} .
\end{aligned}
$$

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

Strassen's matrix multiplication

Divide & Conquer approach

- Reduce time complexity with a novel technique : instead of performing eight recursive multiplications of n/2 × n/2 matrices, it performs only seven.

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

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

Master Theorem → $k = 2$, $\log_2 7 = 2.8$ → $T(n) = \Theta(n^{\log 7}) = \Theta(n^{2.8})$