

# Data structure

## ساختمان داده ها

# اطلاعات درس

مدرس: جعفر طهمورث نژاد

tahmores@gmail.com

تعداد واحد: ۳

پیش نیازها: ریاضیات گسسته - برنامه سازی پیشرفته

پیش نیاز برای: طراحی الگوریتم ها - نظریه زبان ها و

ماشین ها - سیستم های عامل - هوش مصنوعی -

طراحی کامپایلر - پایگاه داده ها

## مراجع

- محمد قدسی، داده ساختارها و مبانی الگوریتم ها، انتشارات فاطمی، 1393.
- Cormen, Thomas H. *Introduction to algorithms*. MIT press, 2009.

## بارم بندی

➤ میان ترم: ۵ (بدون حذف برای پایان ترم)

➤ پایان ترم: ۹

➤ تکالیف: ۲

➤ کوییزها: ۲

➤ پروژه ها: ۳

➤ مجموع: ۲۱

# سرفصل مطالب

1. روشهای تحلیل الگوریتم
2. آرایه ها
3. صف ها
4. پشته ها
5. درخت ها
6. مرتب سازی
7. درهم سازی

# فصل یک

## روشهای تحلیل الگوریتم

# مثال مرتب سازي Sorting

مرتب کردن عناصر يك آرایه با  $n$  عنصر به ترتیب  
صعودي

$$a[0] \leq a[1] \leq \dots \leq a[n-1]$$

$$8, 6, 9, 4, 3 \rightarrow 3, 4, 6, 8, 9$$

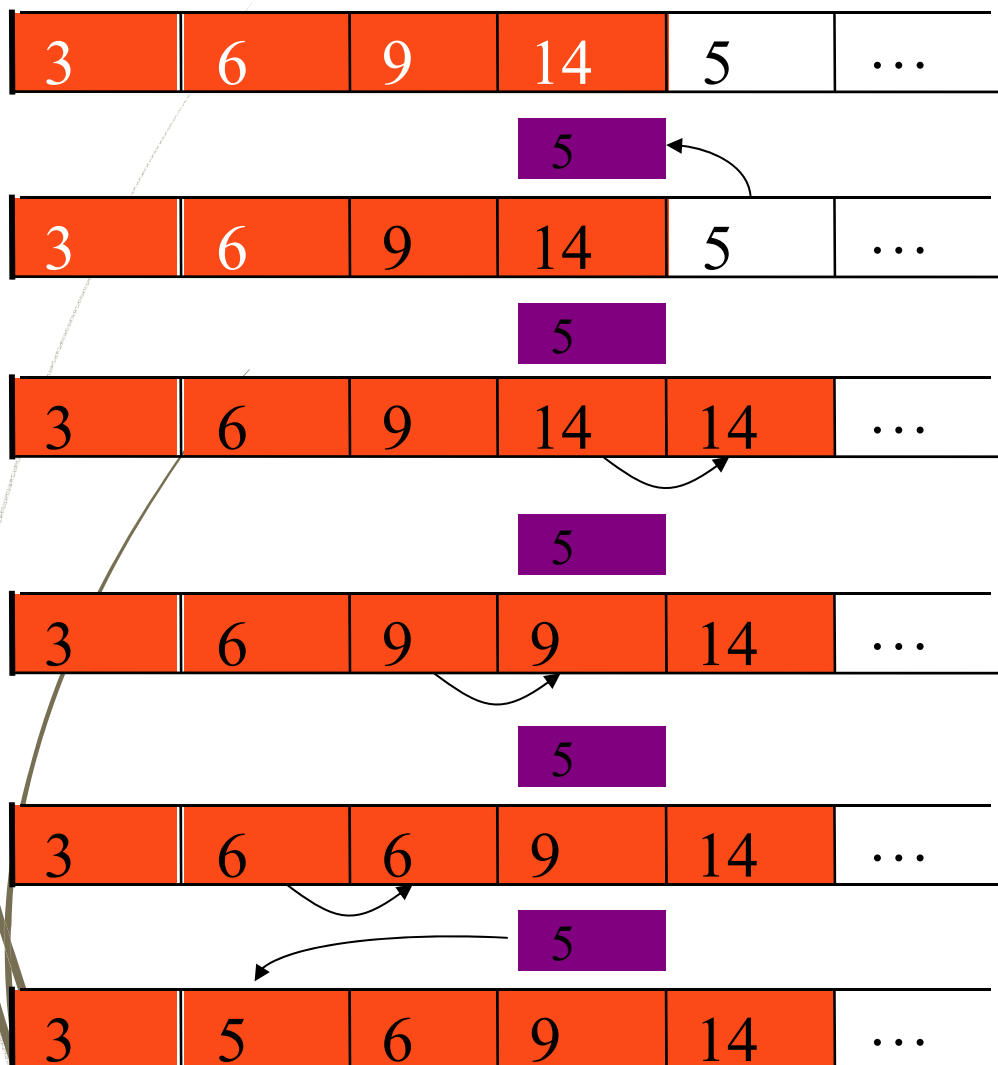
# Insertion Sort

- با يك زیرمجموعه به اندازه 1 شروع كن
- هر بار عنصر بعدي از عناصر مرتب نشده را به زیرمجموعه مرتب شده اضافه كن (insert)



# Insert کردن 5 به 3,6,9,14

9



# مثال: مرتب کردن با insertion Sort

10

**Sorted**

**Unsorted**

23	78	45	8	32	56
----	----	----	---	----	----

Original List

23	78	45	8	32	56
----	----	----	---	----	----

After pass 1

23	45	78	8	32	56
----	----	----	---	----	----

After pass 2

8	23	45	78	32	56
---	----	----	----	----	----

After pass 3

8	23	32	45	78	56
---	----	----	----	----	----

After pass 4

8	23	32	45	56	78
---	----	----	----	----	----

After pass 5

# الگوریتم InsertionSort

11

## InsertionSort

تعداد دفعات اجرا  $\times$  هزینه

cost  $\times$  times executed

for k=2 to length(A)	$\rightarrow$	$c_1 \times n$
key = A[k]	$\rightarrow$	$c_2 \times n-1$
i = k - 1	$\rightarrow$	$c_3 \times n-1$
while (i > 0 and A[i] > key)	$\rightarrow$	$c_4 \times \sum_{k=2}^n t_k$
A[i+1] = A[i]	$\rightarrow$	$c_5 \times \sum_{k=2}^n (t_k - 1)$
i = i - 1	$\rightarrow$	$c_6 \times \sum_{k=2}^n (t_k - 1)$
A[i+1] = key	$\rightarrow$	$c_7 \times n-1$

$t_k$ : وابسته به  
ترتیب عناصر

$$T(n) = c_1 n + c_2 (n-1) + c_3 (n-1) + c_4 \sum_{k=2}^n t_k + c_5 \sum_{k=2}^n (t_k - 1) + c_6 \sum_{k=2}^n (t_k - 1) + c_7 (n-1)$$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{k=2}^n t_k + c_5 \sum_{k=2}^n (t_k - 1) + c_6 \sum_{k=2}^n (t_k - 1) + c_7(n-1)$$

12

$$T(n) = c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{k=2}^n t_k + (c_5 + c_6) \sum_{k=2}^n (t_k - 1)$$

بهترین حالت:  $t_k=1$

$$T(n) = c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4(n-1) = c_1 n + (c_2 + c_3 + c_4 + c_7)(n-1)$$

$$T(n) = A_1 n + B_1$$

بدترین حالت:  $t_k=k$

$$\sum_{k=2}^n t_k = \sum_{k=2}^n k = \frac{n(n+1)}{2} - 1 \quad \sum_{k=2}^n (t_k - 1) = \sum_{k=2}^n (k - 1) = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}$$

$$T(n) = c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \left( \frac{n(n+1)}{2} - 1 \right) + (c_5 + c_6) \left( \frac{n(n-1)}{2} \right)$$

$$T(n) = A_2 n^2 + B_2 n + C_2$$

حالت متوسط: ترکیبهای مختلف ممکن

$$T(n) = A \sum_{k=2}^n \bar{t}_k + Bn + C = A \sum_{k=2}^n \left( \frac{1}{k} \sum_{i=1}^k i \right) + Bn + C$$

$$T(n) = A_3 n^2 + B_3 n + C_3$$

## مقایسه الگوریتم‌های مختلف

13

نسبت، اگر ماشینی 1000 برابر سریعتر استفاده شود	نسبت	برای ماشین 10 برابر سریعتر	حداکثر اندازه مسئله قابل حل در 1000 ثانیه ( $n$ )	$T(n)$
				$100n$
				$5n^2$
				$n^3/2$
				$2^n$

## مقایسه الگوریتم‌های مختلف

14

نسبت، اگر ماشینی 1000 برابر سریعتر استفاده شود	نسبت	برای ماشین 10 برابر سریعتر	حداکثر اندازه مسئله قابل حل در 1000 ثانیه ( $n$ )	$T(n)$
			10	$100n$
			14	$5n^2$
			12	$n^3/2$
			10	$2^n$

## مقایسه الگوریتم‌های مختلف

15

نسبت، اگر ماشینی 1000 برابر سریعتر استفاده شود	نسبت	برای ماشین 10 برابر سریعتر	حداکثر اندازه مسئله قابل حل در 1000 ثانیه ( $n$ )	$T(n)$
	10	100	10	$100n$
	3.2	45	14	$5n^2$
	2.3	27	12	$n^3/2$
	1.3	13	10	$2^n$

## مقایسه الگوریتم‌های مختلف

16

نسبت، اگر ماشین 1000 برابر سریعتر استفاده شود	نسبت	برای ماشین 10 برابر سریعتر	حداکثر اندازه مسئله قابل حل در 1000 ثانیه ( $n$ )	$T(n)$
1000.00	10	100	10	$100n$
31.94	3.2	45	14	$5n^2$
10.50	2.3	27	12	$n^3/2$
2.00	1.3	13	10	$2^n$

$$1000 * 1000 = 10^6$$

$$n = \lg_2 10^6 = 20$$

$$20/10 = 2$$



## مقایسه الگوریتم‌های مختلف

17

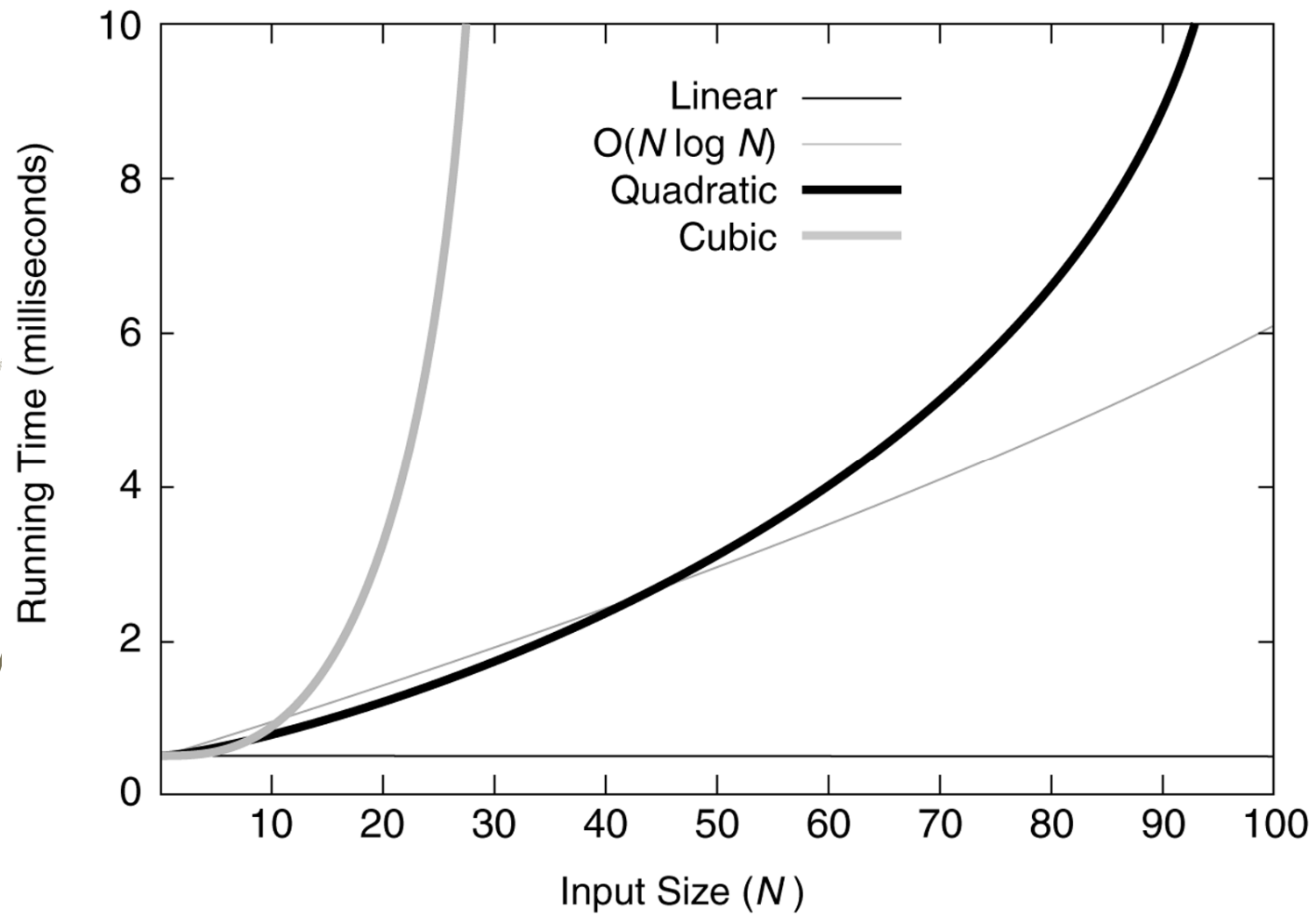
	$T(n)$	حداکثر اندازه مسئله قابل حل در 1000 ثانیه ( $n$ )	برای ماشین 10 برابر سریعتر	نسبت	نسبت، اگر ماشینی 1000 برابر سریعتر استفاده شود
$O(n)$	$100n$	10	100	10	1000.00
$O(n^2)$	$5n^2$	14	45	3.2	31.94
$O(n^3)$	$n^3/2$	12	27	2.3	10.50
$O(2^n)$	$2^n$	10	13	1.3	2.00

$$1000 * 1000 = 10^6$$

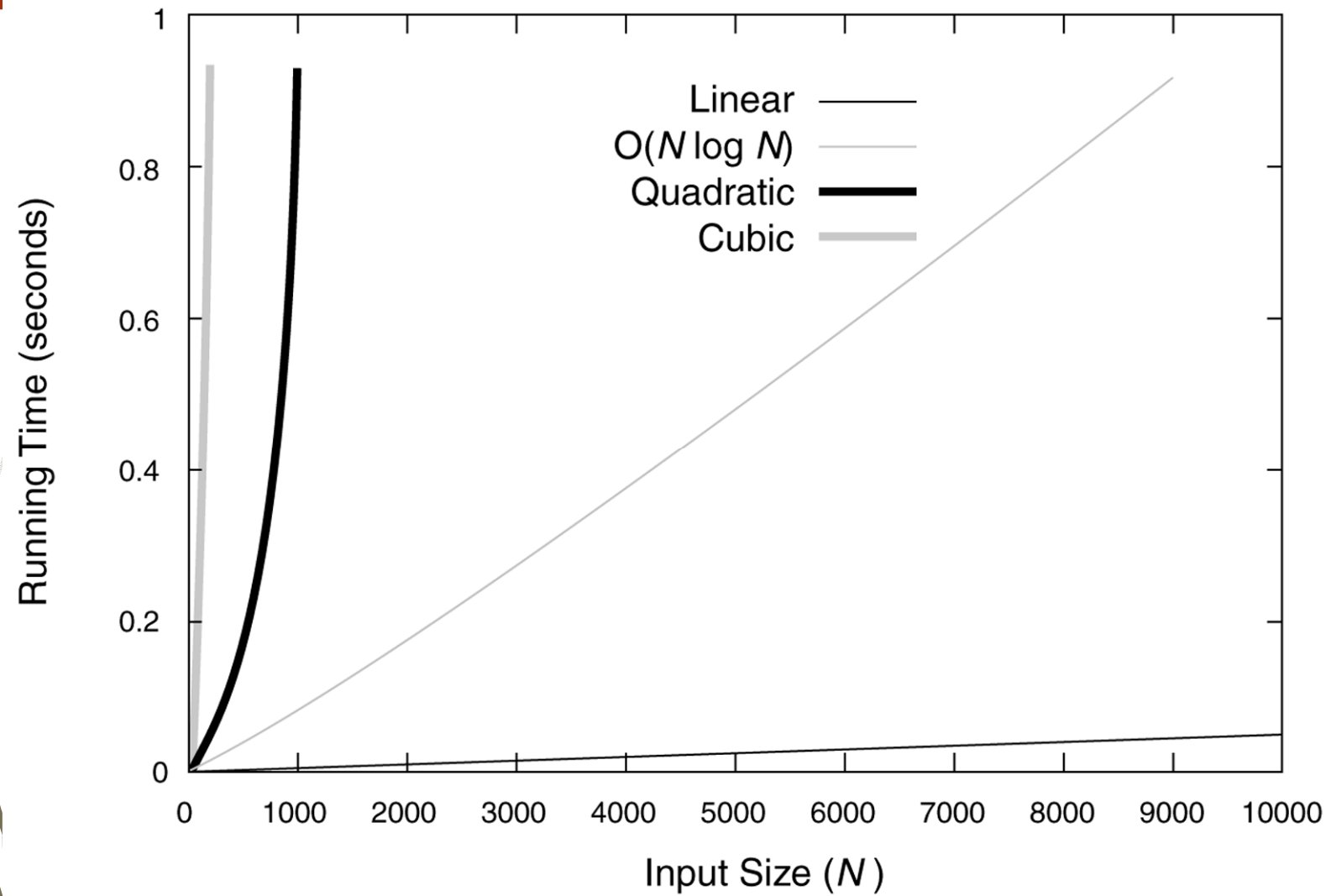
$$n = \lg_2 10^6 = 20$$

$$20/10 = 2$$

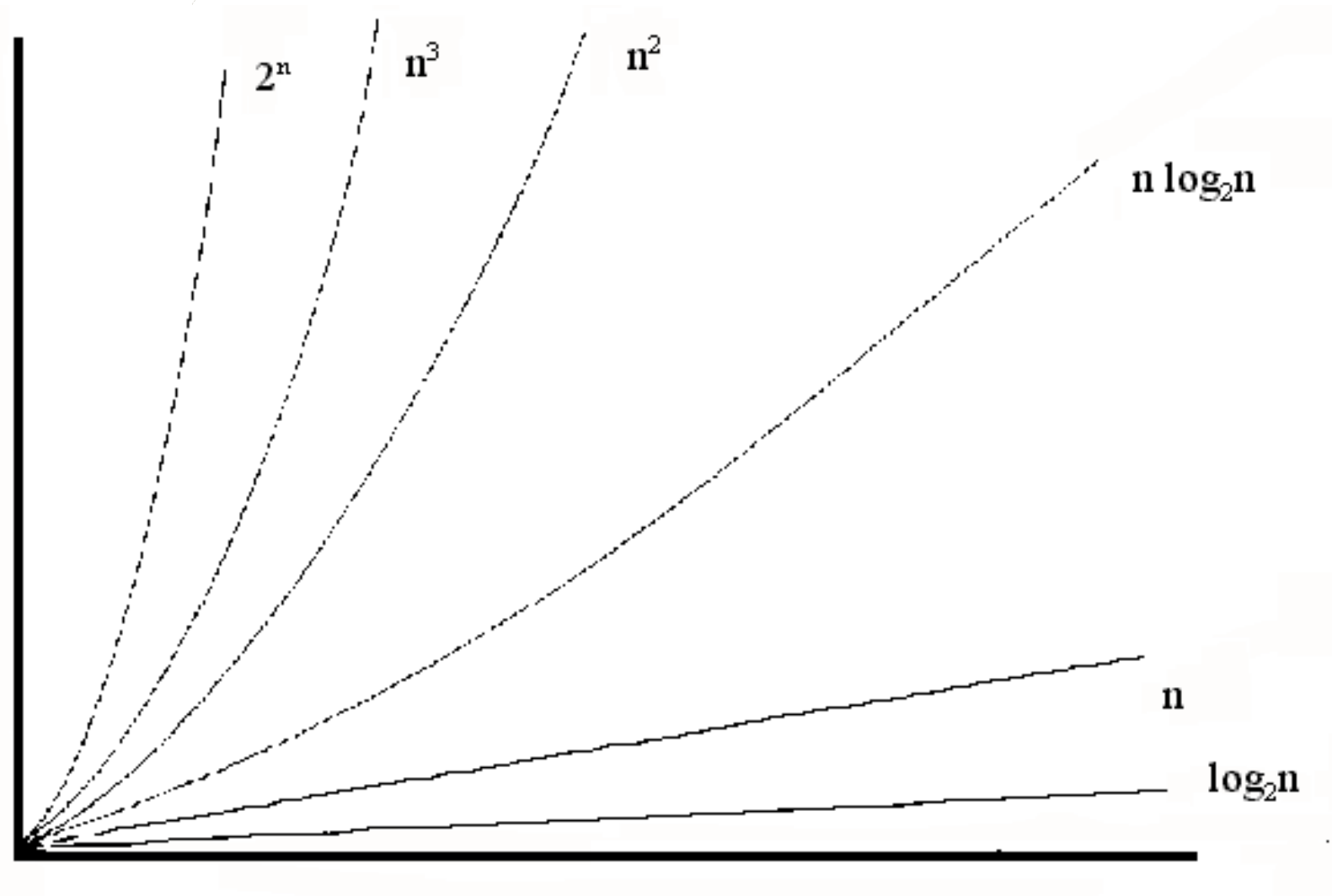
## Running times for small inputs



## Running times for moderate inputs



# Common growth rates نرخ رشد های معمول



## نرخ رشدهای رایج

Function	Growth Rate Name
C	Constant
$\log N$	Logarithmic
N	Linear
$N \log N$	
$N^2$	Quadratic
$N^3$	Cubic
$2^N$	Exponential

# Comparison of $N$ , $\log N$ and $N^2$

<u>N</u>	<u>O(LogN)</u>	<u>O(N<sup>2</sup>)</u>
16	4	256
64	6	4K
256	8	64K
1,024	10	1M
16,384	14	256M
131,072	17	16G
262,144	18	6.87E+10
524,288	19	2.74E+11
1,048,576	20	1.09E+12
1,073,741,824	30	1.15E+18

# Algorithm Analysis: Big Oh

تابع  $f(n)$ ،  $O(g(n))$  است اگر ثابتهاي  $C, n_0 > 0$  وجود داشته باشند به طوري که براي تمام  $n \geq n_0$  داشته باشیم:  $f(n) \leq cg(n)$

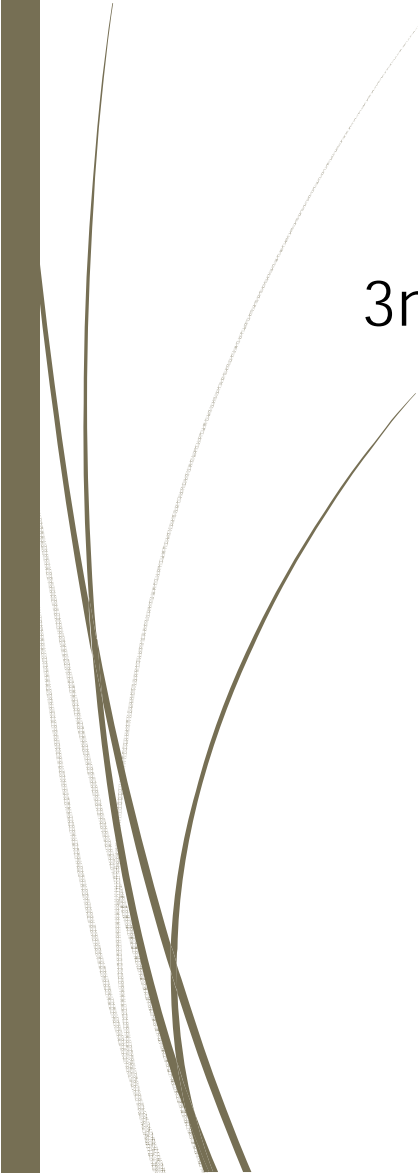
به عبارتي، براي تابع  $g(n)$ ، مجموعه اي از توابع است:

$$O(g(n)) = \{f(n) : \exists c, n_0 > 0 \text{ such that } \forall n \geq n_0 \ 0 \leq f(n) \leq cg(n)\}$$



# Algorithm Analysis: Big Oh

به زبان ساده  $c.g(n)$  به ازاي  $n$  هاي بزرگ، يك حد بالايي  
براي  $f(n)$  است

$$3n+2=O(n)$$




# Algorithm Analysis: Big Oh

به زبان ساده  $c.g(n)$  به ازاي  $n$  هاي بزرگ، يك حد بالايي براي  $f(n)$  است

$$3n+2=O(n) \quad /* \ 3n+2 \leq 4n \text{ for } n \geq 2 \ */$$

# Algorithm Analysis: Big Oh

به زبان ساده  $c.g(n)$  به ازاي  $n$  هاي بزرگ، يك حد بالايي براي  $f(n)$  است

$$3n+2=O(n) \quad /* \ 3n+2 \leq 4n \text{ for } n \geq 2 \ */$$

$$3n+3=O(n)$$

$$100n+6=O(n)$$

$$10n^2+4n+2=O(n^2)$$

$$6 \cdot 2^n + n^2 = O(2^n)$$

$$3n^2+2n+1=O(n^2)$$

# Algorithm Analysis: Big Oh

به زبان ساده  $c.g(n)$  به ازاي  $n$  هاي بزرگ، يك حد بالايي براي  $f(n)$  است

$$3n+2=O(n) \quad /* \ 3n+2 \leq 4n \text{ for } n \geq 2 \ */$$

$$3n+3=O(n) \quad /* \ 3n+3 \leq 4n \text{ for } n \geq 3 \ */$$

$$100n+6=O(n) \quad /* \ 100n+6 \leq 101n \text{ for } n \geq 10 \ */$$

$$10n^2+4n+2=O(n^2) \quad /* \ 10n^2+4n+2 \leq 11n^2 \text{ for } n \geq 5 \ */$$

$$6 \cdot 2^n + n^2 = O(2^n) \quad /* \ 6 \cdot 2^n + n^2 \leq 7 \cdot 2^n \text{ for } n \geq 4 \ */$$

$$3n^2+2n+1=O(n^2)$$

$$2n=O(n^2) \text{ (it is also } O(n) \text{ ) , } \quad 1000n^3 = O(n^{10})$$

## Algorithm Analysis: Omega

تابع  $f(n)$ ،  $\Omega(g(n))$  است اگر ثابتهاي  $C, n_0 > 0$  وجود داشته باشند  
به طوري که برای تمام  $n \geq n_0$  داشته باشیم:  $f(n) \geq cg(n)$

به عبارتي، برای تابع  $g(n)$ ، مجموعه اي از توابع است:

$$\Omega(g(n)) = \{f(n) : \exists c, n_0 > 0 \text{ such that } \forall n \geq n_0 \ 0 \leq cg(n) \leq f(n)\}$$

# Algorithm Analysis: Omega

به زبان ساده  $c.g(n)$  به ازاي  $n$  هاي بزرگ، يك حد پايين براي  $f(n)$  است

Example:

$$2n+10=\Omega(n)$$

$$3n^2+2n+20=\Omega(n^2)$$

$$3n^2+2n+20=\Omega(n)$$

$$an^2+bn+c \text{ is } \Omega(n^2)$$

# Algorithm Analysis: Omega

به زبان ساده  $c.g(n)$  به ازاي  $n$  هاي بزرگ، يك حد پايين براي  $f(n)$  است

Example:

$$2n+10=\Omega(n) \quad /* 2n+10 \geq 2n \text{ for } n \geq 1 */$$

$$3n^2+2n+20=\Omega(n^2) \quad /* 3n^2+2n+20 \geq 3n^2 \text{ for } n \geq 1 */$$

$$3n^2+2n+20=\Omega(n)$$

$$an^2+bn+c \text{ is } \Omega(n^2)$$

## Algorithm Analysis: Theta

تابع  $f(n)$ ،  $\Theta(g(n))$  است اگر ثابتهاي  $n_0 > 0$ ،  $C_1, C_2$  وجود داشته باشند به طوري که براي تمام  $n \geq n_0$  داشته باشیم:

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

به عبارتي، براي تابع  $g(n)$ ،  $\Theta(g(n))$  مجموعه اي از توابع است:

$$\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0 > 0 \text{ s.t. } \forall n \geq n_0 \ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

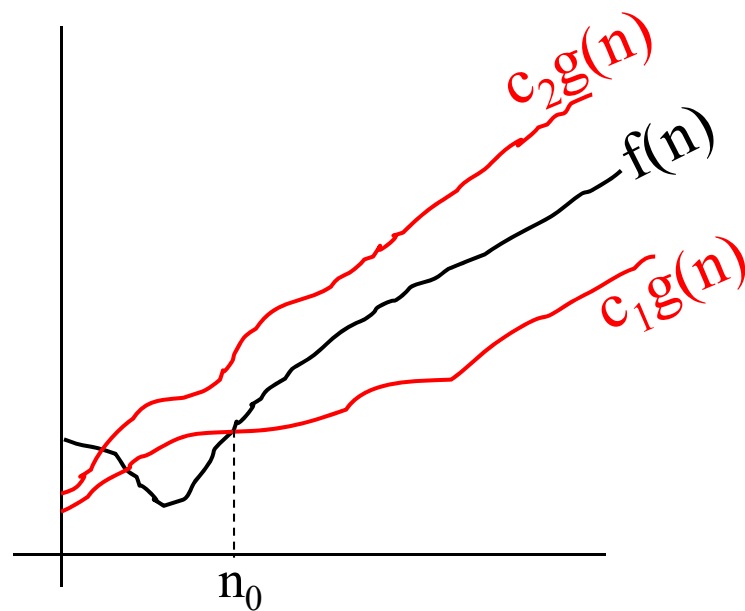
## Algorithm Analysis: Theta

➡ به زبان ساده  $f(n)$  و  $g(n)$  نرخ رشد یکسانی دارند  
 $f(n)$  از مرتبه  $\Theta(g(n))$  است اگر هم  $O(g(n))$  و هم  
 $\Omega(g(n))$  باشد

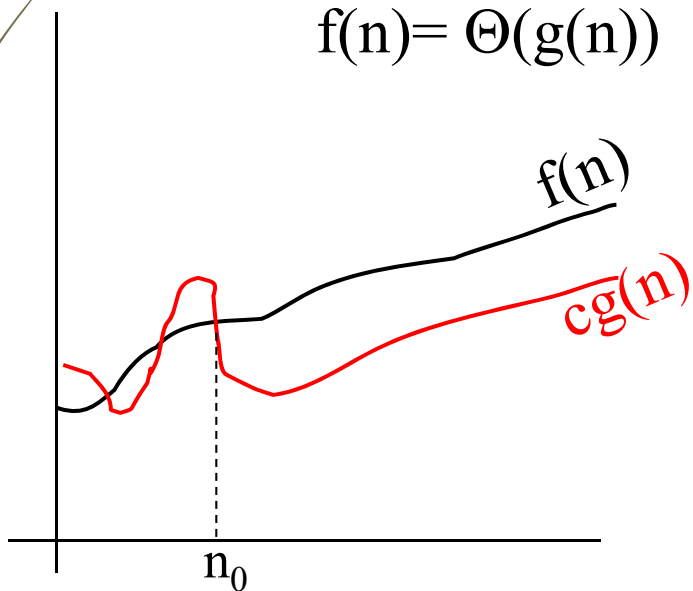
Example:

$$f(n) = 2n+1 \text{ is } \Theta(n)$$

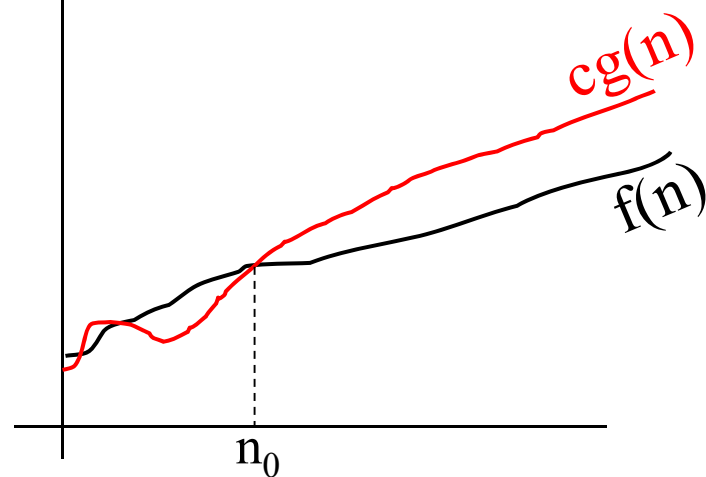




$$f(n) = \Theta(g(n))$$



$$f(n) = \Omega(g(n))$$



$$f(n) = O(g(n))$$

## Algorithm Analysis: Little Oh

تابع  $f(n)$ ،  $o(g(n))$  است اگر برای هر ثابت  $C > 0$  يك ثابت  $n_0 > 0$  وجود داشته باشند به طوري که برای تمام  $n \geq n_0$  داشته باشیم:

$$0 \leq f(n) < Cg(n)$$

به عبارتي، برای تابع  $g(n)$ ، مجموعه اي از توابع است:

الزاماً کوچکتر

$$o(g(n)) = \{f(n) : \forall C > 0 \exists n_0 > 0 \text{ such that } \forall n \geq n_0 \ 0 \leq f(n) < Cg(n)\}$$

## Algorithm Analysis: Little Oh

به زبان ساده  $c.g(n)$  به ازاي  $n$  هاي بزرگ، يك حد بالايي محض براي  $f(n)$  است

Example:

$$f(n) = 3n+4 \text{ is } o(n^2)$$

$$3n+4 < cn^2$$

$$cn^2 - 3n - 4 > 0 \Rightarrow n_0 > \frac{3 + \sqrt{9 + 16c}}{2c}$$

$$f(n) = 2n \text{ is } o(n^2)$$

$$2n < cn^2 \Rightarrow n_0 > \frac{2}{c}$$

( it is also  $O(n)$  and  $O(n^2)$  But is not  $o(n)$  )

## Algorithm Analysis: Little Omega

تابع  $f(n)$ ،  $\omega(g(n))$  است اگر برای هر ثابت  $C > 0$  يك ثابت  $n_0 > 0$  وجود داشته باشد بطوري که برای تمام  $n \geq n_0$  داشته باشیم:

$$0 \leq cg(n) < f(n)$$

به عبارتي، برای تابع  $g(n)$ ،  $\omega(g(n))$  مجموعه اي از توابع است:

$$\omega(g(n)) = \{f(n) : \forall c > 0 \exists n_0 > 0 \text{ such that } \forall n \geq n_0 \ 0 \leq cg(n) < f(n)\}$$

الزاماً کوچکتر

# Algorithm Analysis: Little Omega

به زبان ساده  $cg(n)$  به ازاي  $n$  هاي بزرگ، يك حد پاييني محض  
براي  $f(n)$  است

Example:

$f(n) = n^2/2$  is  $\omega(n)$  But is not  $\omega(n^2)$

$$n^2/2 > cn, n_0 > 2c$$

## برخي خواص

38

خاصيت تعدي (transitivity)

If  $f(n)=\Theta(g(n))$  and  $g(n)=\Theta(h(n))$  then  $f(n)=\Theta(h(n))$

If  $f(n)=O(g(n))$  and  $g(n)=O(h(n))$  then  $f(n)=O(h(n))$

If  $f(n)=\Omega(g(n))$  and  $g(n)=\Omega(h(n))$  then  $f(n)=\Omega(h(n))$

If  $f(n)=o(g(n))$  and  $g(n)=o(h(n))$  then  $f(n)=o(h(n))$

If  $f(n)=\omega(g(n))$  and  $g(n)=\omega(h(n))$  then  $f(n)=\omega(h(n))$

خاصيت بازتابي (reflexivity)

$$f(n)=\Theta(f(n))$$

$$f(n)=O(f(n))$$

$$f(n)=\Omega(f(n))$$

خاصيت تقارني (Symmetry)

$$f(n)=\Theta(g(n)) \text{ if and only if } g(n)=\Theta(f(n))$$

## برخي خواص (ادامه)

تقارن ترانهاده (Transpose Symmetry)

$f(n)=O(g(n))$  if and only if  $g(n)=\Omega(f(n))$

$f(n)=o(g(n))$  if and only if  $g(n)=\omega(f(n))$

- اگر  $f(n)=O(kg(n))$  براي ثابت  $k>0$ ، آنگاه  $f(n)=O(g(n))$

- اگر  $f_1(n) = O(g_1(n))$  و  $f_2(n) = O(g_2(n))$  آنگاه  $(f_1 + f_2)(n) = O(\max(g_1(n), g_2(n)))$

- اگر  $f_1(n) = O(g_1(n))$  و  $f_2(n) = O(g_2(n))$  آنگاه  $f_1(n)f_2(n) = O(g_1(n)g_2(n))$

# Runtime Analysis

قواعد مفید:

جملات ساده (read, write, assign) :  $O(1)$  (ثابت)

عملیات ساده ( + - \* / == > >= < <= ) :  $O(1)$  (ثابت)

ترتیبی از جملات ساده و/یا عملیات ساده :  $O(1)$  (قاعده جمع)

چند تکه برنامه که پشت سرهم اجرا می شود: پرهزینه ترین قسمت

حلقه ها (for, do, while, . . .) : هزینه بدنه حلقه  $\times$  تعداد تکرار حلقه

جملات شرطی (if (cond) then body<sub>1</sub> else body<sub>2</sub> endif):

هزینه  $T_1(n) = \text{body}_1$  ,  $T_2(n) = \text{body}_2$  ,  $T(n) = O(\text{Max}(T_1(n), T_2(n)))$



## Running Time Example(1)

`a = b;` constant time, so it is  $\Theta(1)$ .

```
sum = 0;  
for (i=1; i<=n; i++)  
    sum += n;
```

Asymptotic Complexity:  $O(n)$

# Running Time Example(2)

42

```
sum = 0;  
for (i=1; i<=n; i++)  
    for (j=1; j<=i; j++)  
        sum++;  
for (k=0; k<n; k++)  
    A[k] = k;
```

```
for (i=1; i<=n; i++)  
    for (j=1; j<=n; j++)  
        sum++;
```

$$\sum_{i=1}^n \sum_{j=1}^i 1 = \sum_{i=1}^n i = \frac{n(n+1)}{2}$$
$$\sum_{k=0}^{n-1} 1 = n$$

$$\sum_{i=1}^n \sum_{j=1}^n 1 = \sum_{i=1}^n n = n^2$$

Asymptotic Complexity:  $O(n^2)$

## Running Time Example(3)

```
for (i=1; i<n; i++)  
    if A(i) > maxVal then  
        maxVal= A(i);  
        maxPos= i;
```

## Running Time Example(3)

```
for (i=1; i<n; i++)  
    if A(i) > maxVal then  
        maxVal= A(i);  
        maxPos= i;
```

Asymptotic Complexity:  $O(n)$

## Running Time Example(4)

```
for (i=1; i<n-1; i++)  
    for (j=n; j>=i+1; j--)  
        if (A(j-1) > A(j)) then  
            temp = A(j-1);  
            A(j-1) = A(j);  
            A(j) = temp;  
        endif  
    endfor  
endfor
```

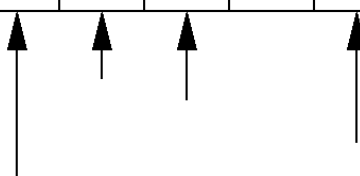
## Running Time Example(4)

```
for (i=1; i<n-1; i++)  
  for (j=n; j>=i+1; j--)  
    if (A(j-1) > A(j)) then  
      temp = A(j-1);  
      A(j-1) = A(j);  
      A(j) = temp;  
    endif  
  endfor  
endfor
```

➡ Asymptotic Complexity is  $O(n^2)$

## جستجوی دودویی

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Key	11	13	21	26	29	36	40	41	45	51	54	56	65	72	77	83

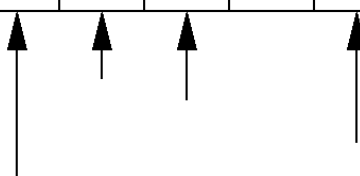


تعداد عناصری که چک می شوند؟

```
// Return position of element in sorted
// array of size n with value K.
int binary(int array[], int n, int K) {
    int l = -1;
    int r = n; // l, r are beyond array bounds
    while (l+1 != r) { // Stop when l, r meet
        int i = (l+r)/2; // Check middle
        if (K < array[i]) r = i; // Left half
        if (K == array[i]) return i; // Found it
        if (K > array[i]) l = i; // Right half
    }
    return n; // Search value not in array
}
```

## جستجوی دودویی

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Key	11	13	21	26	29	36	40	41	45	51	54	56	65	72	77	83



تعداد عناصری که چک می شوند؟

```
// Return position of element in sorted
// array of size n with value K.
int binary(int array[], int n, int K) {
    int l = -1;
    int r = n; // l, r are beyond array bounds
    while (l+1 != r) { // Stop when l, r meet
        int i = (l+r)/2; // Check middle
        if (K < array[i]) r = i; // Left half
        if (K == array[i]) return i; // Found it
        if (K > array[i]) l = i; // Right half
    }
    return n; // Search value not in array
}
```

**Asymptotic Complexity is  $O(\log_2 n)$**



# Max Subsequence Problem

دنباله اي از اعداد صحيح  $A_1, A_2, \dots, A_n$  داده شده، بیشترین حاصل جمع ترتیبی از اعداد  $(\text{subsequence}) A_i, \dots, A_j$  در این دنباله را پیدا کنید.

اعداد می توانند منفی باشند.

اگر همه اعداد منفی باشند، پاسخ عدد صفر است.

مثال:  $-2, 11, -4, 13, -5, -2$

جواب:  $(A_2 + A_3 + A_4) = 20$

چهار الگوریتم مختلف برای این مسئله بررسی می کنیم با پیچیدگیهای  $O(n)$  و  $O(n \log n)$ ،  $O(n^2)$ ،  $O(n^3)$

# 1 الكوريثم

➡ Given  $A_1, \dots, A_n$ , find the maximum value of  $A_i + A_{i+1} + \dots + A_j$   
0 if the max value is negative

```
int maxSum = 0;
```

$$\updownarrow \mathcal{O}(1)$$

```
for( int i = 0; i < a.size( ); i++ )
    for( int j = i; j < a.size( ); j++ )
    {
```

```
int thisSum = 0;
```

$$\updownarrow \mathcal{O}(1)$$

```
for( int k = i; k <= j; k++ )
```

```

thisSum += a[ k ];

```

$$\uparrow O(1) \downarrow O(j-i)$$

```
if( thisSum > maxSum )
```

```
maxSum = thisSum;  $\downarrow$   $O(1)$ 
```

$$\updownarrow \mathcal{O}(1)$$

}

```
return maxSum;
```

$$O(\sum_{j=i}^{n-1} (j-i))$$

$$O(\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j-i))$$

- Time complexity:  $O(n^3)$

## الگوریتم 2

ایده اصلی: اگر مجموع اعداد از  $i$  تا  $j-1$  محاسبه شده، مجموع  $i$  تا  $j$  در زمان ثابت قابل محاسبه است.

به این ترتیب یکی از حلقه های داخلی حذف می شود و زمان اجرای الگوریتم به  $O(n^2)$  کاهش می یابد.

```
int maxSum = 0;

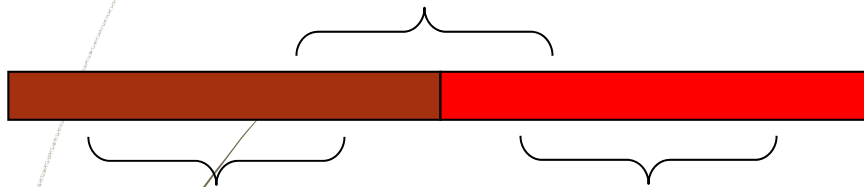
for( int i = 0; i < a.size( ); i++ )
    int thisSum = 0;
    for( int j = i; j < a.size( ); j++ )
    {
        thisSum += a[ j ];
        if( thisSum > maxSum )
            maxSum = thisSum;
    }
return maxSum;
```

## الگوریتم 3

استفاده از روش تقسیم و حل (Divide-and-Conquer)

ورودی را به دو قسمت (تقریباً) مساوی تقسیم می‌کنیم

ترتیب اعداد مورد نظر



یا در نیمه سمت چپ است

یا در نیمه سمت راست است

یا قسمتی از آن در نیمه سمت چپ و قسمتی از آن در نیمه سمت راست است.

مثال:

left half		right half
1 -3 5 2		-1 2 6 -2

max in left : 7      max in right : 8

max : 14

### الگوریتم 3 (ادامه)

- با استفاده از **recursion**، هریک از دو نیمه مورد جستجو قرار می گیرند.
- برای پیدا کردن بیشینه ای که قسمتی از آن در نیمه چپ و قسمتی از آن در نیمه راست باشد، حاصل جمع بیشینه در انتهای راست نیمه سمت چپ و ابتدای چپ نیمه سمت راست بصورت خطی جستجو می شود.

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

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

### آنا ليز الگور يتم 3

$$T(1)=1$$

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

$$= 2.cn/2 + 4T(n/4) + cn$$

$$= 4T(n/4) + 2cn$$

$$= 8T(n/8) + 3cn$$

$$= \dots\dots\dots$$

$$= 2^i T(n/2^i) + icn$$

$$= \dots\dots\dots \text{ (reach a point when } n = 2^i \text{ } i=\log n)$$

$$= n.T(1) + cn \log n$$

## الگوریتم 4

55

2, 3, -2, 1, -5, 4, 1, -3, 4, -1, 2

```
int maxSum = 0, thisSum = 0;

for( int j = 0; j < a.size( ); j++ )
{
    thisSum += a[ j ];

    if ( thisSum > maxSum )
        maxSum = thisSum;
    else if ( thisSum < 0 )
        thisSum = 0;
}
return maxSum;
}
```

هزینه الگوریتم:  $O(n)$

- ترتیب اعداد مورد جستجو با عدد منفی شروع یا تمام نمی شود.
- ترتیب اعداد مورد جستجو نمی تواند پیشوندی با مجموع کمتر از صفر داشته باشد.
- مثال: -2 11 -4 13 -5 -2
- بنا براین اگر مجموع  $A_i..A_j$  کمتر از صفر شود، می توان  $i$  را تا  $j+1$  جلو برد (و  $thisSum$  را صفر کرد).

## خلاصه

برای حل یک مسئله، چهار الگوریتم متفاوت با هزینه های زیر ارائه شد.

$$O(n^3)$$

$$O(n^2)$$

$$O(n \log n)$$

$$O(n)$$

برای مسئله ای با اندازه  $10^6$ ، الگوریتم اول در زمان عمر ما تمام نمی شود و الگوریتم چهارم در حد ثانیه تمام می شود.



## تحليل الگوریتمهای بازگشتی (recursive)

➤  $T(n)$  بصورت بازگشتی برحسب  $T(k)$  ،  $K < n$  بیان می شود.

➤ شرط (های) خاتمه بازگشت وجود دارد (مثلاً  $T(0)$  یا  $T(1)$ )

➤ روشهای حل رابطه های بازگشتی:

➤ حدس و اثبات

➤ تکرار با جایگذاری رابطه بازگشتی

➤ درخت بازگشت

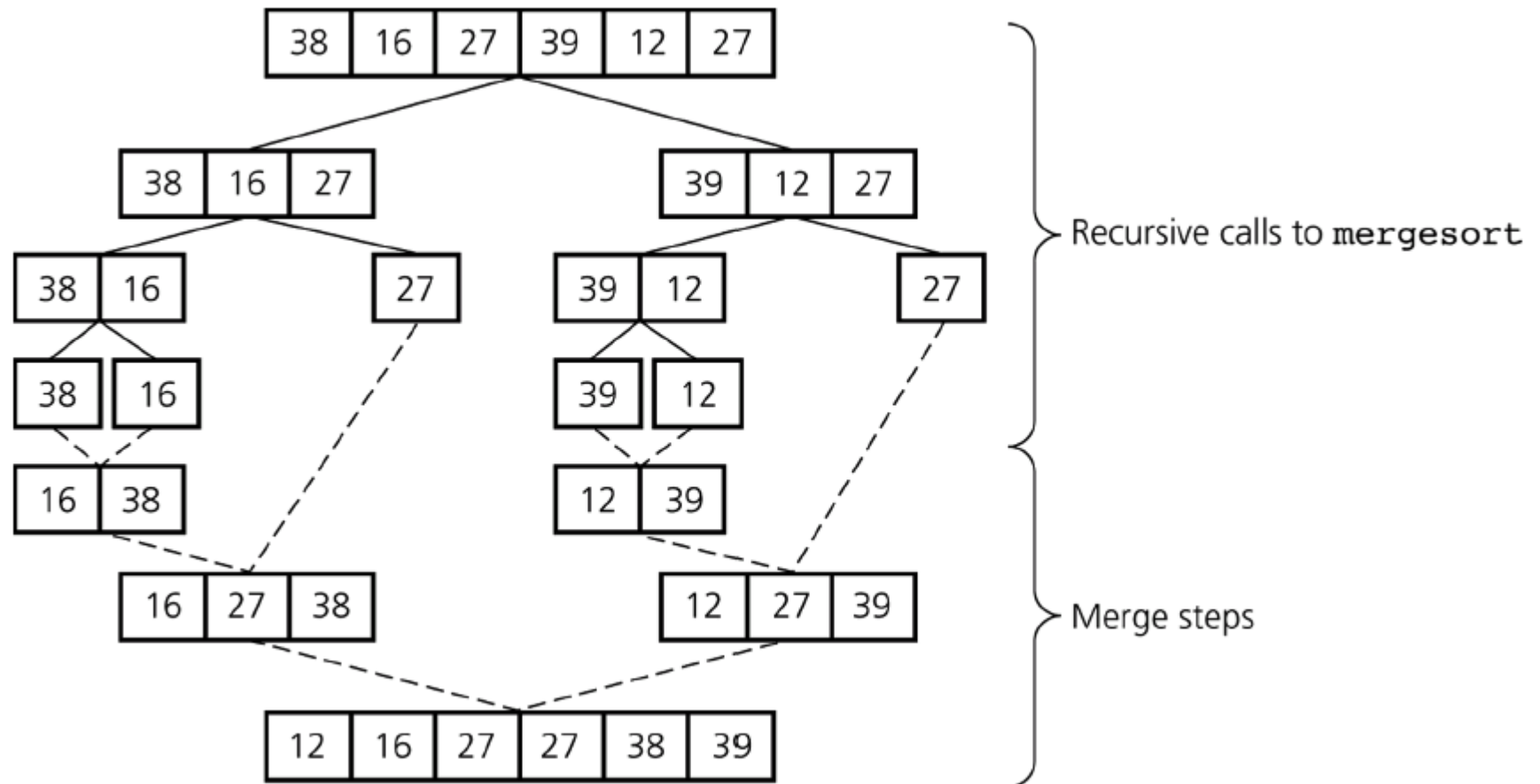
➤ قضیه اصلی

➤ حل روابط بازگشتی

## روش حدس و اثبات

➤ حدس جواب و اثبات آن، (معمولاً با استفاده از استقرا)

➤ مثال: Merge Sort



## مثال Merge Sort

```
MergeSort(A[i..j])
```

```
if (i < j) {
```

```
    mid = (i+j)/2
```

```
    MergeSort(A[i..mid]);
```

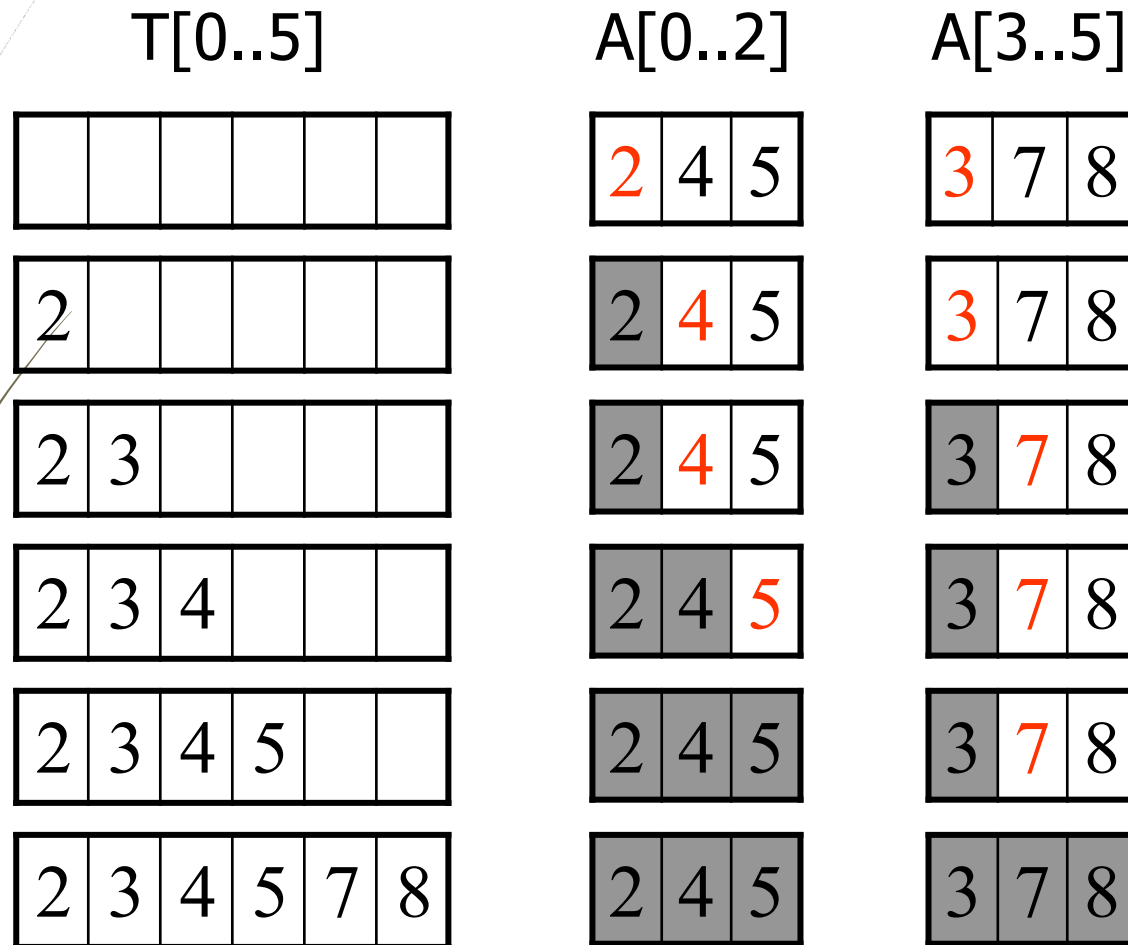
```
    MergeSort(A[mid+1..j]);
```

```
    Merge(A[i..mid], A[mid+1..j]);
```

```
}
```

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

# How to merge two subarrays?



مثال:  $T(n) = 2T(\lfloor n/2 \rfloor) + n$

حدس:  $T(n) = O(n \lg n)$

باید ثابت کنیم ثابت  $C$  وجود دارد که  $T(n) \leq cn \lg n$

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \leq 2(c \lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor) + n$$

$$T(n) \leq cn \lg(n/2) + n = cn \lg(n) - cn \lg(2) + n = cn \lg n - cn + n$$

$$\Rightarrow T(n) \leq cn \lg n$$

برای  $c \geq 1$

## روش تکرار با جایگذاری رابطه بازگشتی

رابطه بازگشتی تا رسیدن به جواب بسط داده می شود.  
مثال:

$$T(n) = 2T(n/2) + n, \quad T(1) = 1$$

$$= 2(2T(n/4) + n/2) + n$$

$$= 4T(n/4) + n + n$$

$$= 4(2T(n/8) + n/4) + 2n$$

$$= 8T(n/8) + 3n$$

$$= 2^i T(n/2^i) + in$$

با فرض  $n = 2^k$

$$= 2^k T(1) + kn$$

$$= n + n \log n$$

# Example: Factorial

63

```
int factorial (int n) {  
    if (n<=1) return 1;  
    else return n * factorial(n-1);  
}
```

$\text{factorial}(n) = n * n-1 * n-2 * \dots * 1$   $T(n)$

$n$   $*$   $\text{factorial}(n-1)$   $T(n-1)$

$n-1$   $*$   $\text{factorial}(n-2)$   $T(n-2)$

$n-2$   $*$   $\text{factorial}(n-3)$

...

$2$   $*$   $\text{factorial}(1)$   $T(1)$

$T(n)$

$= T(n-1) + d$

$= T(n-2) + d + d$

$= T(n-3) + d + d + d$

$= \dots$

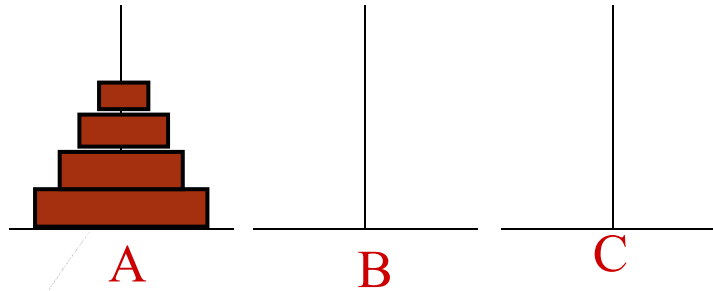
$= T(1) + (n-1) * d$

$= c + (n-1) * d$

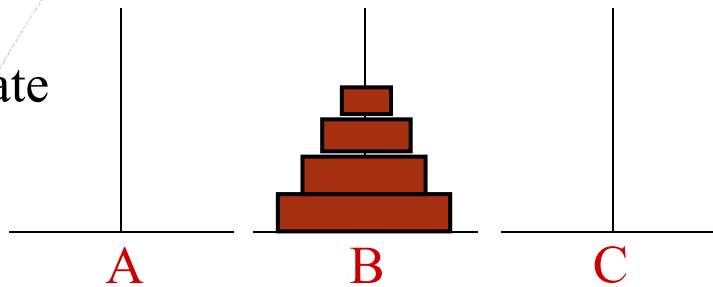
$= O(n)$

initial state

64



final state



```
void tower (int n, char A,char B,char C)
{
    if (n==1)
        move(A,B);
    else {
        tower(n-1,A,C,B);
        move(A,B);
        tower(n-1,C,B,A);
    };
}
```

مثال: برج هانوي

$Hanoi(n, A, B, C) ::$   
 $Hanoi(n-1, A, C, B),$   
 $Hanoi(1, A, B, C),$   
 $Hanoi(n-1, C, B, A)$

$$\begin{aligned} T(n) &= 2T(n-1) + c \\ &= 2^2 T(n-2) + 2c + c \\ &= 2^3 T(n-3) + 2^2 c + 2c + c \\ &= \dots \\ &= 2^{n-1} T(1) + (2^{n-2} + \dots + 2 + 1)c \\ &= (2^{n-1} + 2^{n-2} + \dots + 2 + 1)c \\ &= O(2^n) \end{aligned}$$



سوال

65

$$T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \lg n$$

$$T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \lg n$$

$$T(2^m) = 2T(2^{m/2}) + m$$

$$S(m) = T(2^m)$$

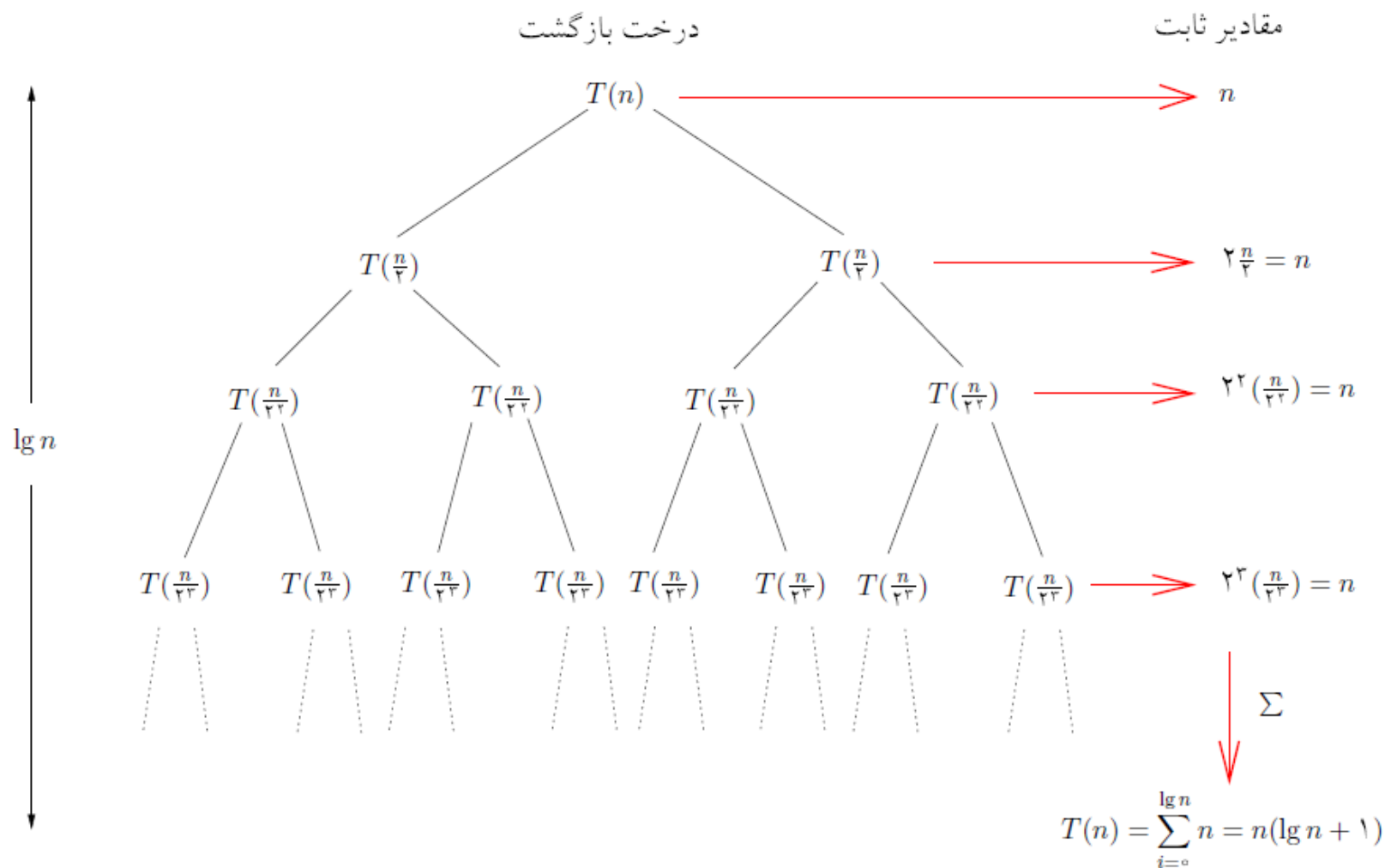
$$S(m) = 2S(m/2) + m$$

$$S(m) = O(m \lg m)$$

$$T(n) = T(2^m) = S(m) = O(m \lg m) = O(\lg n \lg \lg n)$$

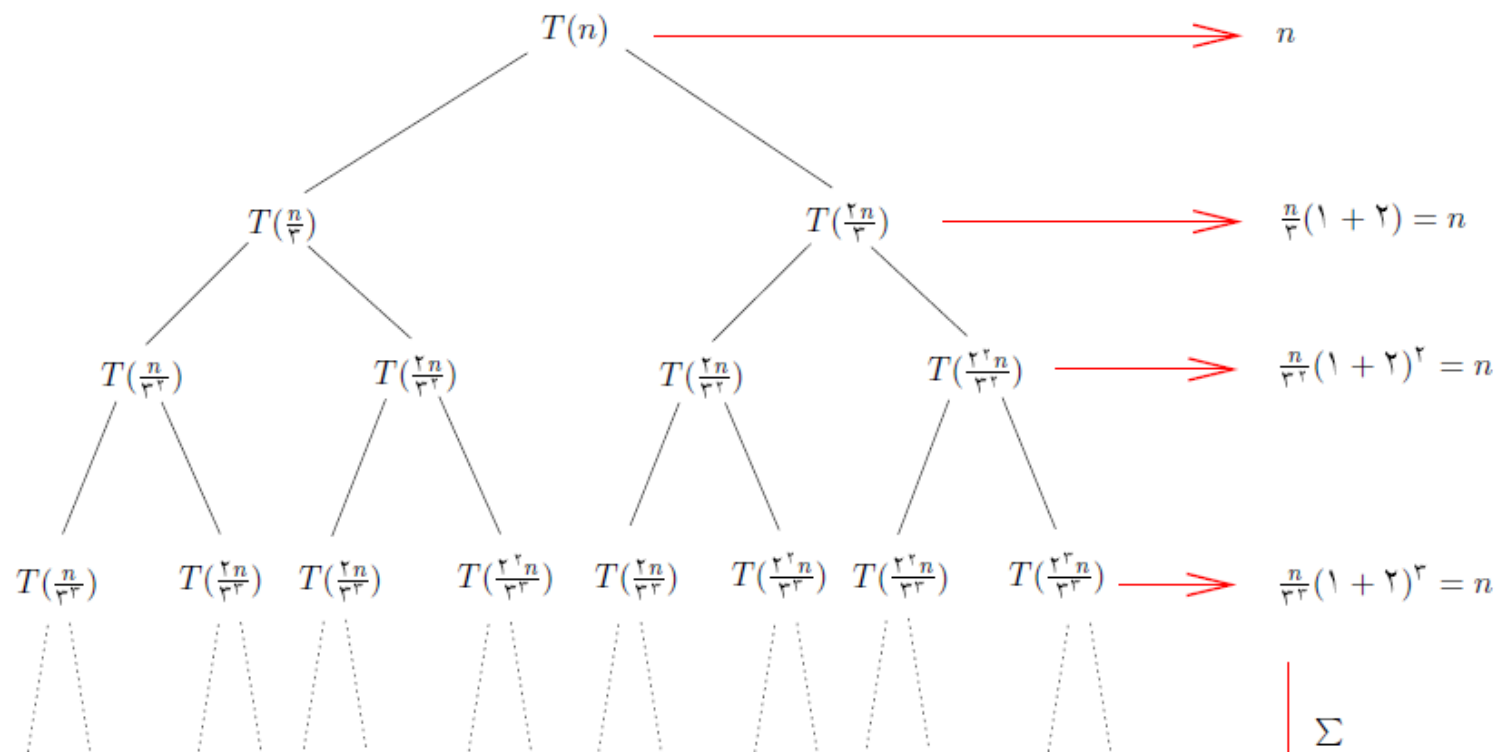
# درخت بازگشت

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



# درخت بازگشت

$$T(n) = T\left(\frac{n}{r}\right) + T\left(\frac{r-1}{r}n\right) + n$$



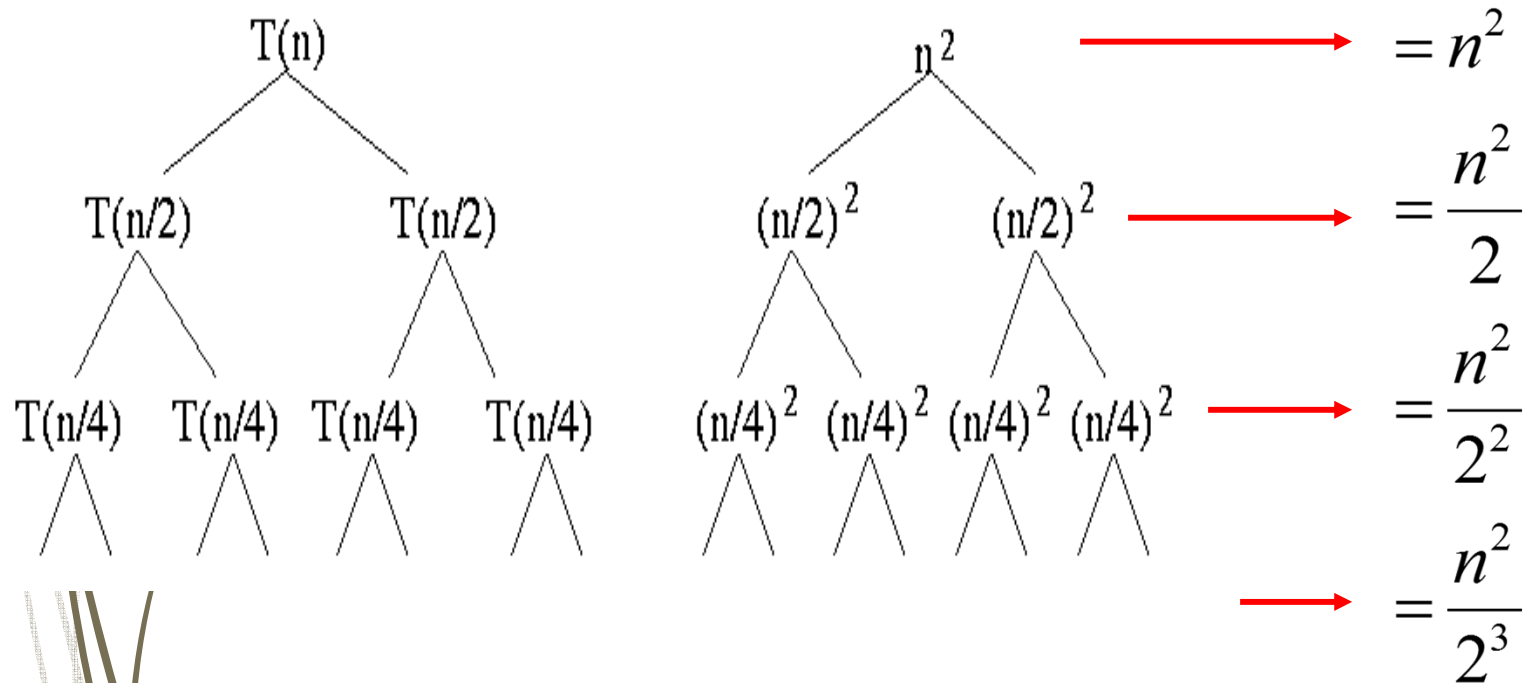
$$T(n) < \sum_{i=0}^{\lg n} n = n(\lg n + 1)$$

# درخت بازگشت

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

# درخت بازگشت

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



$$T(n) = n^2 \left( 1 + 1/2 + 1/2^2 + 1/2^3 + \dots + 1/2^{\lg n} \right)$$

$$T(n) = O(n^2)$$

## قضیه اصلی Master Theorem

برای رابطه بازگشتی  $T(n) = aT(n/b) + f(n)$  که  $a \geq 1$  و  $b > 1$  داریم:

1 اگر  $f(n) = O(n^{\lg_b^{a-\varepsilon}})$  برای  $\varepsilon > 0$  ،  $T(n) = \Theta(n^{\lg_b^a})$

2 اگر  $f(n) = \Theta(n^{\lg_b^a})$  ، آنگاه  $T(n) = \Theta(n^{\lg_b^a} \lg n)$

3 اگر  $f(n) = \Omega(n^{\lg_b^{a+\varepsilon}})$  برای  $\varepsilon > 0$  ، آنگاه  $T(n) = \Theta(f(n))$

## قضیه اصلی Master Theorem

$$\begin{array}{ll} g(n) = n^{\lg_b a} & \text{G آهنگ رشد} \\ f(n) & \text{F آهنگ رشد} \end{array}$$

$$T(n) = \Theta(f(n)) \quad F > G \quad \text{1 اگر}$$

$$T(n) = \Theta(g(n)) \quad G > F \quad \text{2 اگر}$$

$$T(n) = \Theta(g(n) \lg n) \quad F = G \quad \text{3 اگر}$$



## مثال

$$T(n) = 9T(n/3) + n \quad T(1) = c$$

$$a = 9, b = 3, f(n) = n$$

$$g(n) = n^2$$

مثال (حالت 1)

$$T(n) = \Theta(n^2)$$

$$T(n) = T(2n/3) + 1 \quad T(1) = c$$

$$a = 1, b = 3/2, f(n) = 1$$

مثال (حالت 2)

$$g(n) = n^{\lg_b a} = n^{\lg_{3/2} 1} = n^0 = 1 \quad T(n) = \Theta(\lg n)$$

$$T(n) = 3T(n/4) + n \lg n \quad T(1) = c$$

$$a = 3, b = 4, f(n) = n \lg n$$

مثال (حالت 3)

$$g(n) = n^{\lg_b a} = n^{\lg_4 3} = n^{0.793}$$

$$\Rightarrow T(n) = \Theta(n \lg n)$$

## تکلیف 5

$$T(n) = 7T(n/2) + n^2$$

$$T(n) = 4T(n/2) + \lg n$$

$$T(n) = 4T(n/2) + n^2$$

$$T(n) = 3T(n/3) + n/2$$

$$T(n) = 7T(n/3) + n^2$$

$$T(n) = 8T(n/3) + 2^n$$

$$T(n) = 3T(n/3) + n \lg n$$

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

# حل روابط بازگشتی

Fibonacci numbers •

- $F(0) = 0, F(1) = 1$

$$F(n) = F(n-1) + F(n-2) \text{ for } n > 2$$

(0, 1, 1, 2, 3, 5, 8, 13, 21...)

```
int fib(int n){  
    if (n<=2)  
        return 1;  
    else  
        return (fib(n-1)+fib(n-2))  
}
```

## حل روابط بازگشتی همگن

$$F(n) = F(n-1) + F(n-2)$$

اگر  $c_i$  ها اعدادهای حقیقی باشند، به رابطه‌ی بازگشتی زیر

رابطه‌ی بازگشتی همگن از درجه‌ی  $k$

می‌گوییم:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_k a_{n-k}$$

تابع  $g(n)$  یک جواب دنباله‌ی بازگشتی فوق است، اگر دنباله‌ی  $a_n = g(n)$  در  $(n \in \mathcal{N})$  در رابطه‌ی بازگشتی صدق کند.

# روش حل

اگر  $g(n) = x^n$  جواب رابطه‌ی بازگشتی همگن باشد، داریم:

$$x^n - c_1 x^{n-1} - c_2 x^{n-2} - \dots - c_k x^{n-k} = 0$$

یا به عبارت دیگر،

$$x^k - c_1 x^{k-1} - \dots - c_k = 0$$

یعنی  $x$  جواب معادله درجه  $k$  فوق است.

این معادله را معادله‌ی مشخصه (characteristic equation) برای رابطه‌ی بازگشتی می‌نامیم.

# حل روابط بازگشتی همگن

Fibonacci numbers •

- $F(n) = F(n-1) + F(n-2)$

حل: معادله‌ی مشخصه:  $x^2 - x - 1 = 0$

ریشه‌های آن  $x_1 = \frac{1+\sqrt{5}}{2}$  و  $x_2 = \frac{1-\sqrt{5}}{2}$

$$f_n = t_1 \left( \frac{1+\sqrt{5}}{2} \right)^n + t_2 \left( \frac{1-\sqrt{5}}{2} \right)^n$$

و با توجه به مقادیر اولیه داریم:

$$\begin{cases} t_1 + t_2 = f_0 = 0 \\ t_1 \left( \frac{1+\sqrt{5}}{2} \right) + t_2 \left( \frac{1-\sqrt{5}}{2} \right) = f_1 = 1 \end{cases}$$

و از این معادله‌ها نتیجه میشود:

$$t_1 = \frac{1}{\sqrt{5}}, t_2 = -\frac{1}{\sqrt{5}}$$

$$f_n = \frac{1}{\sqrt{5}} \left( \left( \frac{1 + \sqrt{5}}{2} \right)^n - \left( \frac{1 - \sqrt{5}}{2} \right)^n \right)$$

جمله‌ی  $\left(\frac{1-\sqrt{5}}{2}\right)^n$  با بزرگتر شدن  $n$  بسیار کوچک می‌شود و با توجه به اینکه  $f_n$  عددی حسابی است، اگر  $\langle x \rangle$  را نزدیکترین عدد صحیح به  $x$  تعریف کنیم، داریم:

$$\langle x \rangle = \lfloor x + \frac{1}{2} \rfloor$$

و با این تعریف:

$$f_n = \left\langle \frac{1}{\sqrt{5}} \left( \frac{1 + \sqrt{5}}{2} \right)^n \right\rangle$$