برنامه سازی پیشرفته
(مبانی تحلیل الگوریتم)

صادق اسکندری – دانشکده علوم ریاضی، گروه علوم کامپیوتر

eskandari@guilan.ac.ir

عوامل موثر در زمان اجرای یک الگوریتم

۱- پردازنده

هر چه سخت افزار قوی تر باشد، اجرای الگوریتم سریعتر است

۲- زبان برنامه نویسی

معمولاً زبان های کامپایلری (مانند **C++**) سریعتر از زبانهای مفسری مانند پایتون هستند.

۳- اندازه ورودی

یک الگوریتم، برای مرتب سازی یک لیست یک میلیارد عددی نیازمند زمان بیشتری نسبت به مرتب سازی یک لیست ده عددی دارد.

۴- پیچیدگی الگوریتم

برای محاسبه جمله **n** ام دنباله فیبونانچی، الگوریتم بازگشتی بسیار کندتر از الگوریتم تکراری است.

عوامل موثر در زمان اجرای یک الگوریتم

۱- پردازنده

هر چه سخت افزار قوی تر باشد، اجرای الگوریتم سریعتر است

۲- زبان برنامه نویسی

معمولاً زبان های کامپایلری (مانند **C++**) سریعتر از زبانهای مفسری مانند پایتون هستند.

۳- اندازه ورودی

یک الگوریتم، برای مرتب سازی یک لیست یک میلیارد عددی نیازمند زمان بیشتری نسبت به مرتب سازی یک لیست ده عددی دارد.

۴- پیچیدگی الگوریتم

برای محاسبه جمله **n** ام دنباله فیبوناچی، الگوریتم بازگشتی بسیار کندتر از الگوریتم تکراری است.

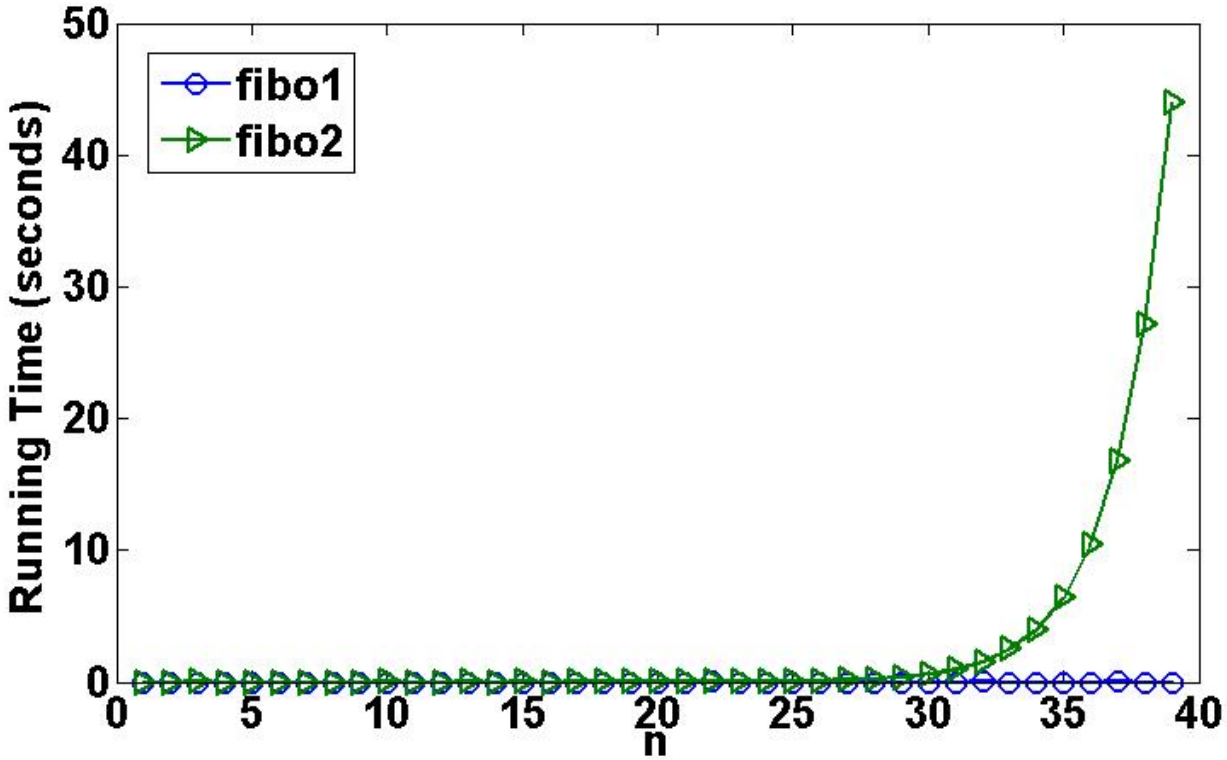
سوال: کدامیک از موارد فوق تحت کنترل برنامه نویس است؟ پیچیدگی الگوریتم

تأثیر پیچیدگی الگوریتم بر زمان اجرا

مثال: محاسبه جمله **n** ام دنباله فیبوناچی

```python
def fibo1(n):
    if n == 1 or n == 2:
        return 1
    a, b,c = 1,1,0
    for i in range(3,n+1):
        c = a+b
        a = b
        b = c
    return c
```

```python
def fibo2(n):
    if n == 1 or n == 2:
        return 1
    return fibo2(n-1)+fibo2(n-2)
```



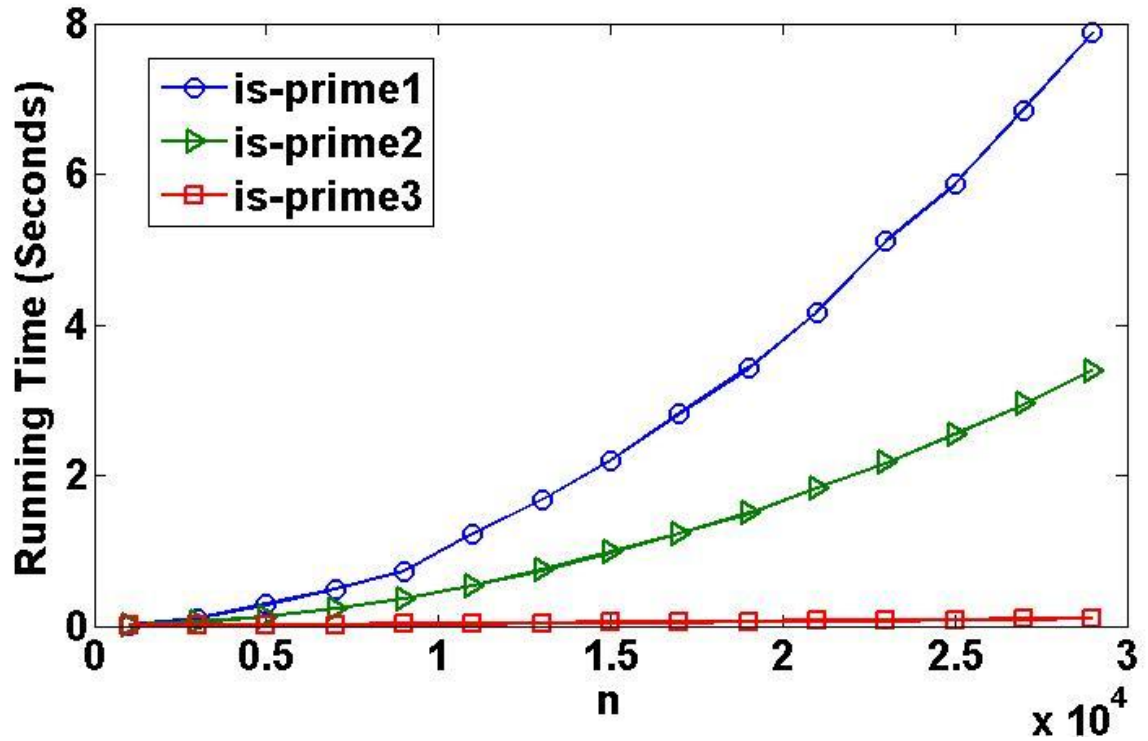
تأثیر پیچیدگی الگوریتم بر زمان اجرا

مثال: محاسبه تعداد اعداد اول کوچکتر از **n**

```python
def is_prime1(n):
    if n == 2:
        return True
    for i in range(2,n-1):
        if n%i == 0:
            return False
    return True
```

```python
def is_prime2(n):
    if n == 2:
        return True
    for i in range(2,n//2+1):
        if n%i == 0:
            return False
    return True
```

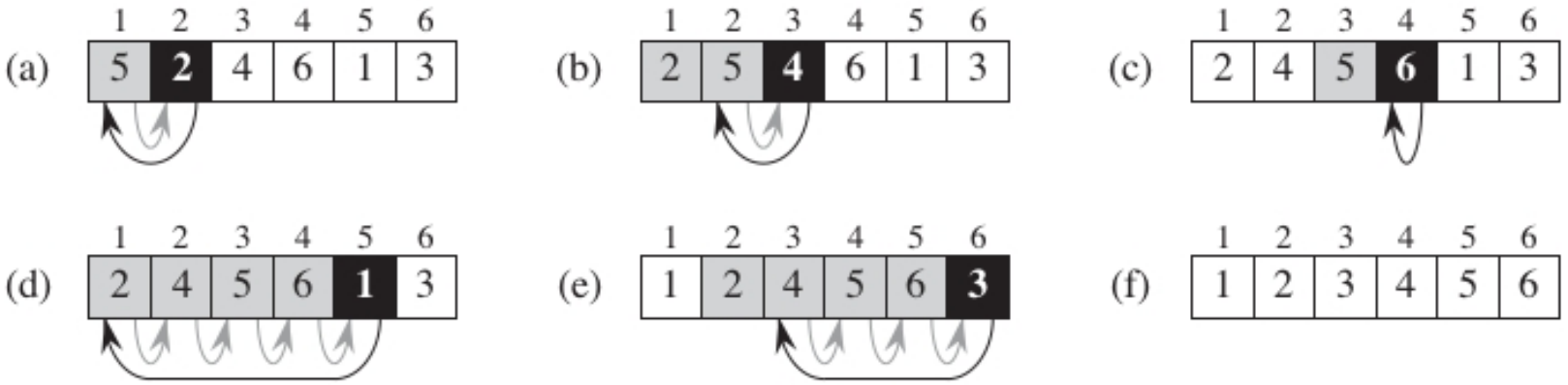
```python
import math
def is_prime3(n):
    if n == 2:
        return True
    for i in range(2,int(math.sqrt(n)+1)):
        if n%i == 0:
            return False
    return True
```



تحلیل خط به خط پیچیدگی الگوریتم ها

مثال: الگوریتم مرتب سازی درجی

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```



تحلیل خط به خط پیچیدگی الگوریتم ها

مثال: الگوریتم مرتب سازی درجی

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
8 $A[i+1] = key$	c_8	$n-1$

پیچیدگی این الگوریتم در حالت کلی:

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + (c_6+c_7)\sum_{j=2}^{n}(t_j-1) + c_8(n-1)$$

تحلیل خط به خط پیچیدگی الگوریتم ها

مثال: الگوریتم مرتب سازی درجی

$$T(n) = c_۱ n + c_۲ (n-۱) + c_۴ (n-۱) + c_۵ \sum_{j=۲}^{n} t_j + (c_۶ + c_۷) \sum_{j=۲}^{n} (t_j - ۱) + c_۸ (n-۱)$$

بهترین حالت: لیست از ابتدا مرتب باشد ($\forall j: t_j = 1$)

$$T(n) = c_۱ n + (c_۲ + c_۴ + c_۸)(n-۱) + c_۵ \sum_{j=۲}^{n} ۱ = An + B$$

بدترین حالت: لیست از ابتدا کاملا نامرتب (معکوس) باشد ($\forall j: t_j = j$)

$$T(n) = c_۱ n + (c_۲ + c_۴ + c_۸)(n-۱) + c_۵ \sum_{j=۲}^{n} j + (c_۶ + c_۷) \sum_{j=۲}^{n} (j-۱) = An^۲ + Bn + C$$

تأثیر پیچیدگی های مختلف بر زمان اجرای الگوریتم

الگوریتم	T(n) (ثانیه)	حداکثر اندازه ورودی که در کمتر از ۱۰۰۰ ثانیه اجرا می شود	حداکثر اندازه ورودی با پردازنده ۱۰ برابر سریعتر	نسبت	نسبت افزایش حداکثر اندازه ورودی با پردازنده ۱۰۰۰ برابر سریعتر
A_1	$100n$	$100n<1000 \Rightarrow n=10$	$\frac{(100n)}{10}<1000 \Rightarrow n=100$	10	1000
A_2	$5n^2$	$5n^2<1000 \Rightarrow n=14$	45	3.2	131.9
A_3	$\frac{n^3}{2}$	$\frac{n^3}{2}<1000 \Rightarrow n=12$	27	2.3	125.9
A_4	2^n	$2^n<1000 \Rightarrow n=9$	13	1.3	2

سوال: چقدر طول میکشد تا هر یک از الگوریتم های فوق، مسئله ای با اندازه $\boldsymbol{n = 100}$ را حل کنند.

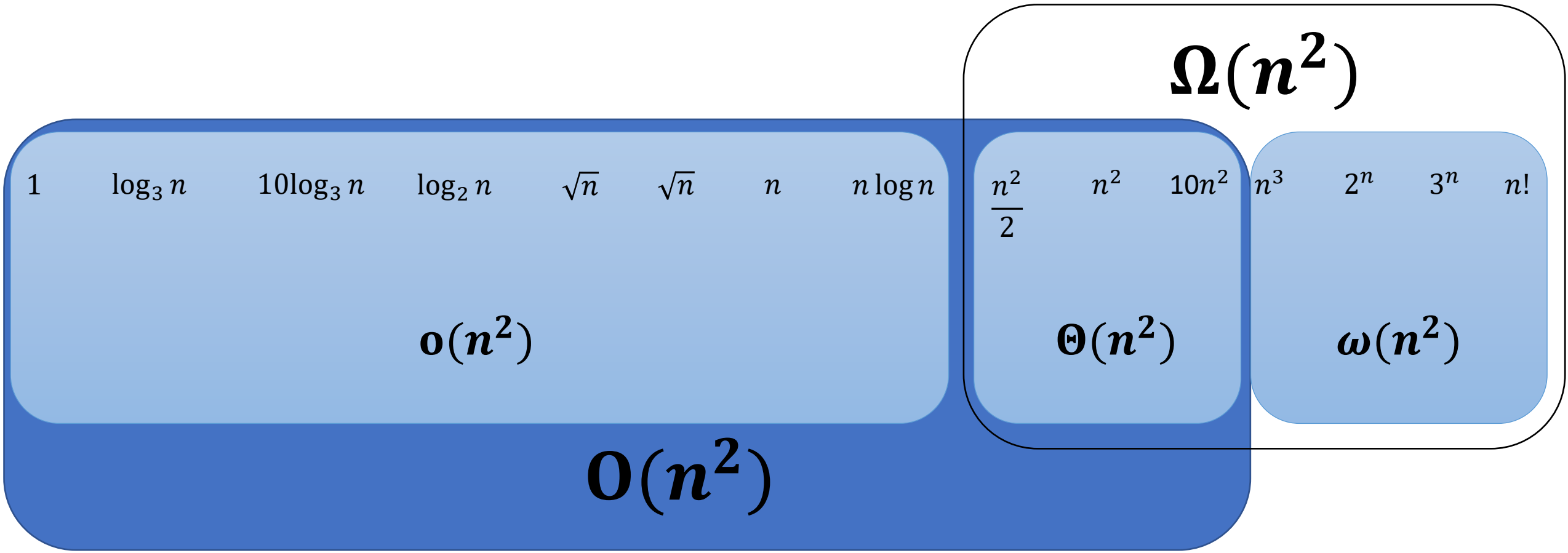
$A_۱ : ۱۰۰ \times ۱۰۰ = ۱۰۰۰۰ \approx ۳\ hours$

$A_۲ : ۵ \times ۱۰۰۰۰ = ۵۰۰۰۰ \approx ۱۴\ hours$

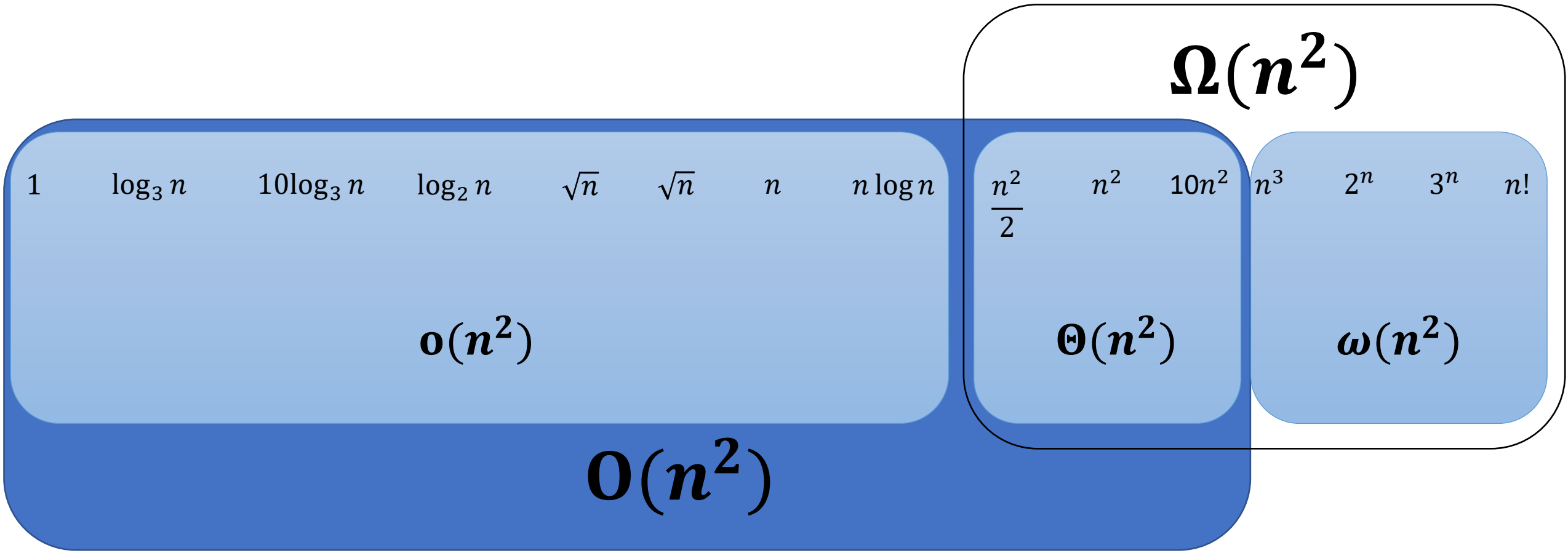
$A_۳ : ۰/۵ \times ۱۰۰۰۰۰۰ = ۵۰۰۰۰۰ \approx ۱۳۹\ hours$

$A_۴ : ۲^{۱۰۰} \approx ۴ \times ۱۰^{۲۲}\ years$

نمادهای مجانبی

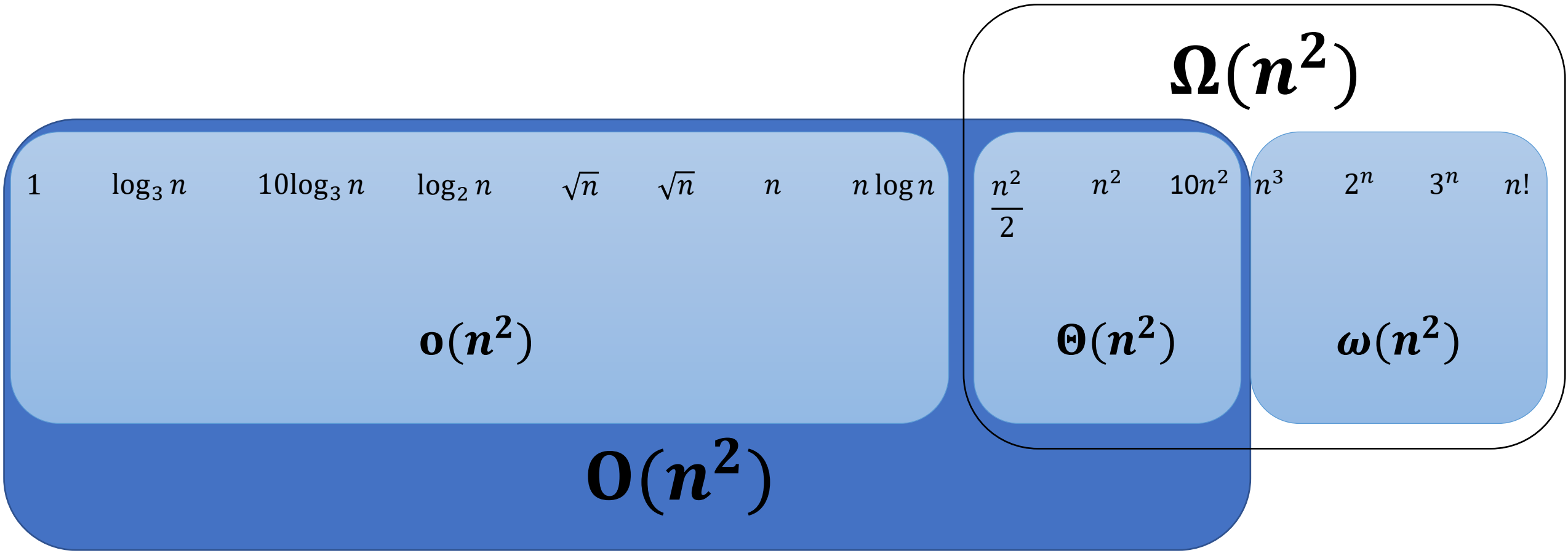


نمادهای مجانبی



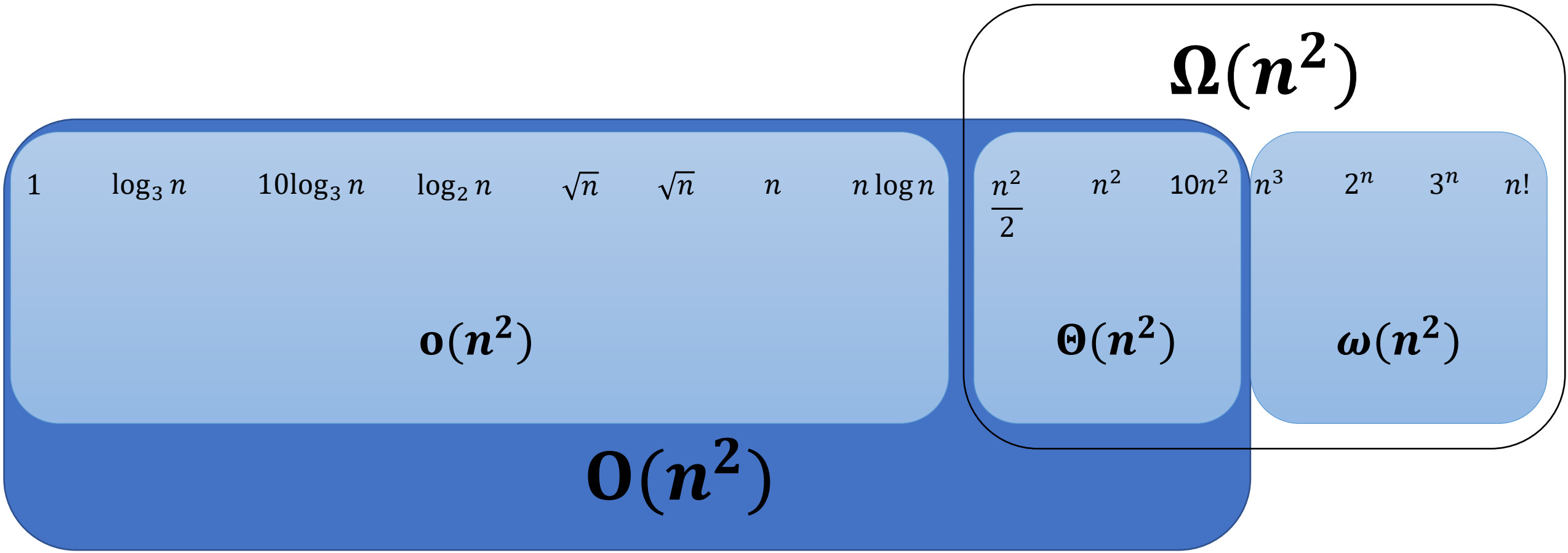
پیچیدگی زمانی الگوریتم درجی در بهترین حالت $\Theta(n)$ است

نمادهای مجانبی



پیچیدگی زمانی الگوریتم درجی در بدترین حالت $\Theta(n^2)$ است

نمادهای مجانبی



پیچیدگی زمانی الگوریتم درجی در حالت کلی $O(n^2)$ است

الگوریتم های مرتب سازی

مرتب سازی (Sorting) جزء مهمی از بسیاری از الگوریتم های کامپیوتری می باشد.

برای یک لیست داده شده، هدف از مرتب سازی، یافتن جایگشتی از عناصر است به گونه ای که ترتیب خاصی (صعودی یا نزولی) را برآورده کنند.

در الگوریتم هایی که در ادامه ارائه می شوند، ورودی یک لیست از اعداد و خروجی لیست مرتب صعودی خواهد بود.

الگوریتم های مرتب سازی

مهمترین الگوریتم های مرتب سازی:

الگوریتم	پیچیدگی بهترین حالت	پیچیدگی بدترین حالت	پیچیدگی
درجی (Insertion)	$\Theta(n)$	$\Theta(n^2)$	$O(n^2)$
حبابی (Bubble)			
ادغامی (Merge)			
سریع (Quick)			

الگوریتم های مرتب سازی

الگوریتم مرتب سازی حبابی

```python
def bubbleSort(arr):
    n = len(arr)
    for i in range(n-1):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
```

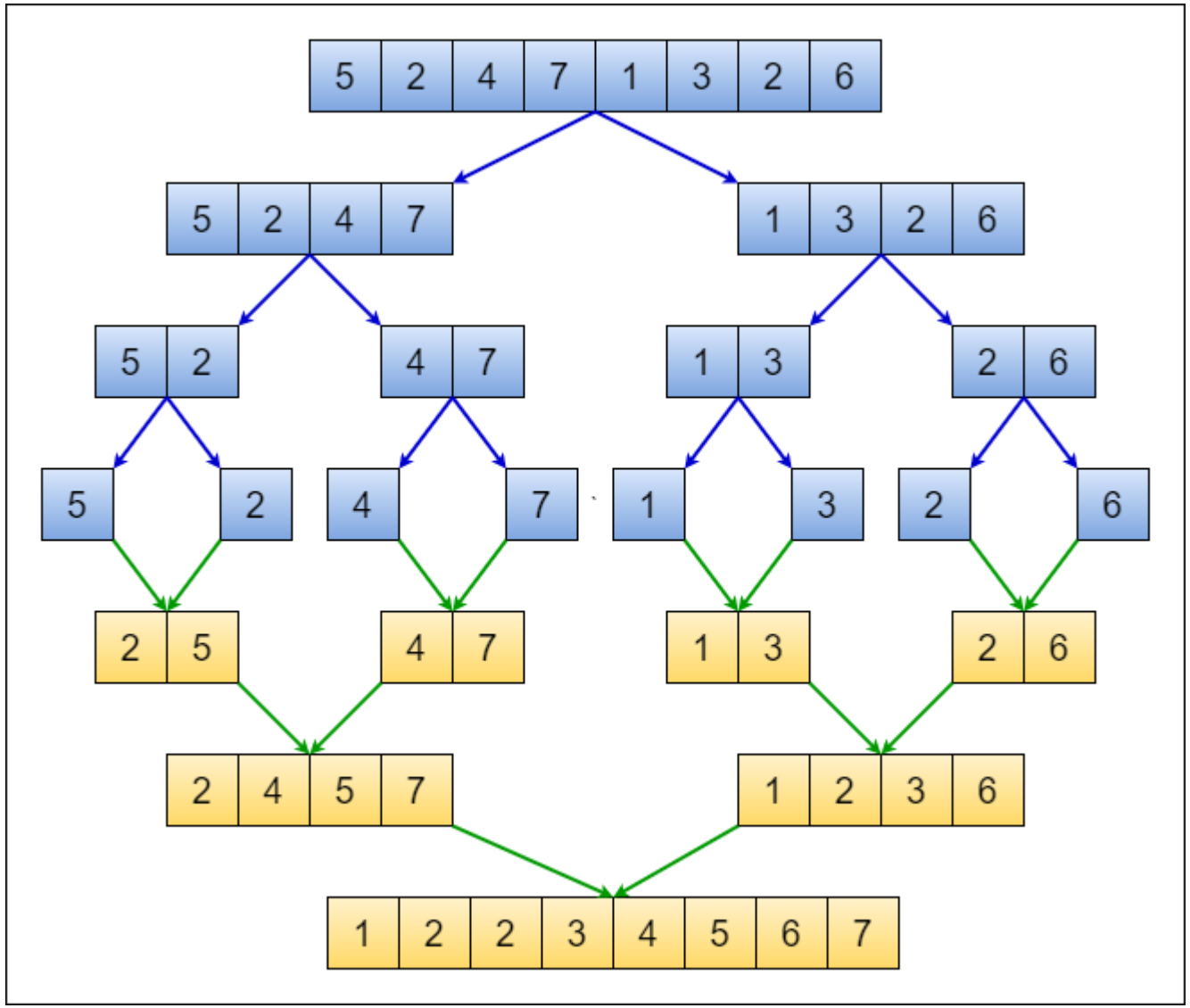
الگوریتم های مرتب سازی

مهمترین الگوریتم های مرتب سازی:

الگوریتم	پیچیدگی بهترین حالت	پیچیدگی بدترین حالت	پیچیدگی
درجی (Insertion)	$\Theta(n)$	$\Theta(n^2)$	$O(n^2)$
حبابی (Bubble)	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
ادغامی (Merge)			
سریع (Quick)			

الگوریتم های مرتب سازی

الگوریتم مرتب سازی ادغامی



الگوریتم های مرتب سازی

الگوریتم مرتب سازی ادغامی

```
def mergeSort(arr):
    if len(arr) > 1:


        mid = len(arr)//2   # Finding the mid of the list
        L = arr[:mid]   # Dividing the list elements
        R = arr[mid:]   # into 2 halves

        mergeSort(L)   # Sorting the first half
        mergeSort(R)   # Sorting the second half

        i = j = k = 0
        while i < len(L) and j < len(R):  # Merging the two sorted lists
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1
```

الگوریتم های مرتب سازی

مهمترین الگوریتم های مرتب سازی:

الگوریتم	پیچیدگی بهترین حالت	پیچیدگی بدترین حالت	پیچیدگی
درجی (Insertion)	$\Theta(n)$	$\Theta(n^2)$	$O(n^2)$
حبابی (Bubble)	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
ادغامی (Merge)	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
سریع (Quick)			

الگوریتم مرتب سازی سریع

تمرین