AI VIET NAM
@aivietnam.edu.vn

Algorithm design and analysis

Big O - Algorithm Analysis

Nguyễn Quốc Thái

Data Structures and Algorithms

Relation between Data Structures and Algorithms

Program = Data Structures + Algorithms

- Data structure:

- a way of organizing, storing and performing operations on data
- examples: list, set, dictionary,…

- Algorithm:

- describes a sequence of steps to solve a computational problem or perform a calculation
- described in pseudocode, a programming language

Objectives

Objectives of Learning Algorithms

“Life is meaningful, without objective life is vague
At this moment our goal is to learn Algorithms.”

- Algorithm efficiency is typically measured by the algorithm's computational complexity
- Computational complexity is the amount of resources used by the algorithm

AI VIET NAM
@aivietnam.edu.vn

Algorithm

Design an algorithm

- Prove the algorithm is correct

  Loop invariant, recursive function, .

Analysis the algorithm

- Time
- Space

Sequential and parallel algorithms

- Random access model (RAM)
- Parallel Multi-processor access model (PRAM)

Exercise7 (OPTIONAL)

Cho một số nguyên dương n, viết phương trình đảo ngược thứ tự các vị trí trong số n. Chỉ dùng while (hay for) và những phép toán cơ bản như +, -, *, /, %, // ...

- Input: n là một dãy số nguyên dương.
- Output: Đảo ngược vị trí các số trong n. Ví dụ input: 12345678910, output: 1987654321

NOTE: Các bạn chú ý các điều kiện sau

- Không được ép kiểu sang string
- Chỉ sử dụng while hoặc for loop

```python
def reverse_number(n):
    r_num = 0
    while n > 0:
        reminder = n % 10
        r_num = (r_num * 10) + reminder
        n = n // 10 #//: chia làm tròn xuống
    return r_num

print(reverse_number(12))

print(reverse_number(n=123456789))
```

```
21
987654321
```

AI VIET NAM
@aivietnam.edu.vn

Algorithm efficiency

Experimental Analysis

- Computational complexity is the amount of resources used by the algorithm
- Independent of the hardware and software environment
- Study a high-level description of the algorithm without need for implementation
- Takes into account all possible

$\Rightarrow$ The most common resources considered :

Runtime (time) complexity

Memory usage (space) complexity

Running Time

The ‘time’ function of the time module

```
[4] import time
    start_time = time.time()

    i = 0
    while i < 5:
        i = i + 1

    end_time = time.time()

    print(end_time-start_time)
```

```
5.7697296142578125e-05
```

```
[ ] import time
    start_time = time.time()

    # run algorithm

    end_time = time.time()

    print(end_time-start_time)
```

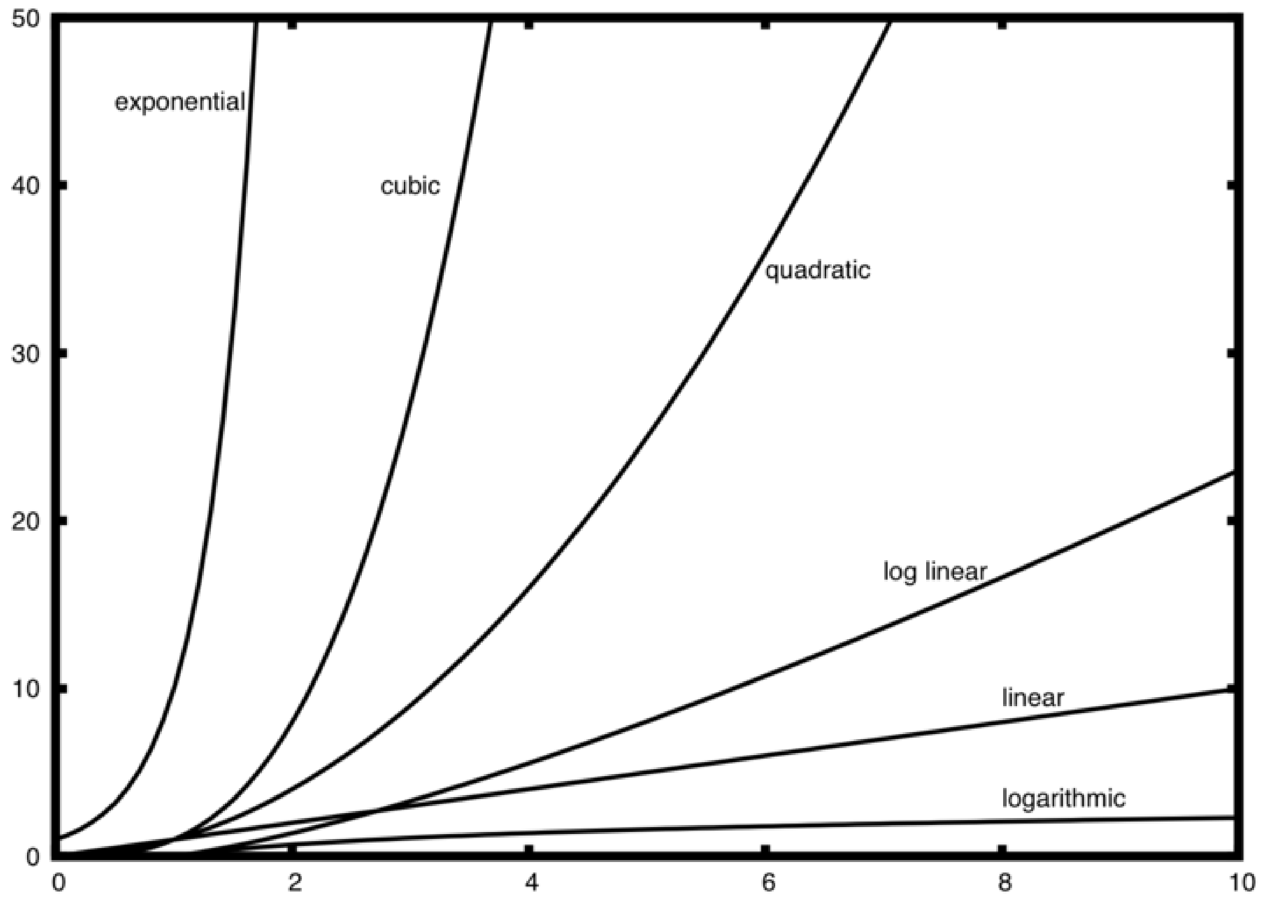
AI VIET NAM
@aivietnam.edu.vn

CONTENT

AI VIET NAM
@aivietnam.edu.vn

CONTENT

(1) – Big O – Algorithm Analysis



Computational complexity

Steps to calculate computational complexity

Python code → **Characterize Function** → **Asymptotic Notation**

```
n = 10
i = 0
total = 0
while i < n:
    total = total + i
    i = i + 1
    j = 0
    while j < n:
        total = total + j
        j = j + 1
print(total)
```

```
495
```

1 → $T(n) = 5n^2 + 6n + 4$

2 → $O(n^2)$

Computational complexity

Steps to calculate computational complexity

Python code → ① **Characterize Function** → ② **Asymptotic Notation**

```python
n = 10
i = 0
total = 0
while i < n:
    total = total + i
    i = i + 1
    j = 0
    while j < n:
        total = total + j
        j = j + 1
print(total)
```

```
495
```

$T(n) = 5n^2 + 6n + 4$ → $O(n^2)$

- Primitive Operations
- Important Function

Computational complexity

Steps to calculate computational complexity

Python code | **Characterize Function** | **Asymptotic Notation**

```
n = 10
i = 0
total = 0
while i < n:
    total = total + i
    i = i + 1
    j = 0
    while j < n:
        total = total + j
        j = j + 1
print(total)
```

```
495
```

1 → $T(n) = 5n^2 + 6n + 4$ 2 → $O(n^2)$

Primitive Operations

Important Function

Computational complexity

Primitive Operations

- Assigning an identifier to an object
- Performing an arithmetic operation
- Comparing two numbers

```
number_a = 10
number_b = number_a
number_a + number_b
```

```
20
```

```
add_number = number_a + number_b
add_number
```

```
20
```

```
number_a > number_b
```

```
False
```

```
compare_number = number_a > number_b
compare_number
```

```
False
```

Computational complexity

Primitive Operations

- Accessing a single element of list by index
- Calling a function
- Returning from a function

```
list_number = [1, 2, 3]
list_number[0]
```

```
1
```

```
def sum_list(list_number):
    total = sum(list_number)
    return total

sum_list(list_number)
```

```
6
```

Computational complexity

Counting Primitive Operations

- The number of primitive operations an algorithm performs will be proportional to the actual running time

Measuring Operations as a Function of Input Size

- To capture the order of growth of an algorithm's running time
- A function $f(n)$: characterizes the number of primitive operations that are performed as a function of the input size n
- Common functions: $f(n) = c$, $log_b n$, n, $nlogn$, …

AI VIET NAM
@aivietnam.edu.vn

Simple algorithm analysis

O(n)

“Big-Oh” Notation

The number of primitive operations that are performed

Simple algorithm analysis

Counting Primitive Operations

```
[1]  number_a = 10
     number_b = number_a
     number_a + number_b
```

```
20
```

```
[6]  number_a = 10
     number_b = number_a
     add_number = number_a + number_b
     add_number
```

```
20
```

```
number_a = 10
number_b = number_a
number_a > number_b
```

```
False
```

AI VIET NAM
@aivietnam.edu.vn

Simple algorithm analysis

Counting Primitive Operations

```
[1] number_a = 10
    number_b = number_a
    number_a + number_b
```

```
20
```

3 operations → O(1)

```
[6] number_a = 10
    number_b = number_a
    add_number = number_a + number_b
    add_number
```

```
20
```

5 operations → O(1)

```
number_a = 10
number_b = number_a
number_a > number_b
```

```
False
```

3 operations → O(1)

AI VIET NAM
@aivietnam.edu.vn

Simple algorithm analysis

Counting Primitive Operations

```
number_a = 10
number_b = number_a
number_a > number_b
```

```
False
```

```
[7] number_a = 10
    number_b = number_a
    number_c = number_a + number_b
    number_d = number_a * number_b
    number_c >= number_d
```

```
False
```

AI VIET NAM
@aivietnam.edu.vn

Simple algorithm analysis

Counting Primitive Operations

```
number_a = 10
number_b = number_a
number_a > number_b
```

```
False
```

3 operations

O(1)

```
[7] number_a = 10
    number_b = number_a
    number_c = number_a + number_b
    number_d = number_a * number_b
    number_c >= number_d
```

```
False
```

7 operations

O(1)

AI VIET NAM
@aivietnam.edu.vn

Simple algorithm analysis

Counting Primitive Operations

```python
def compute_rectangle_area(height=0, width=0):
    '''
    This function aims to compute area for a rectangle.

    height -- the height of the rectangle
    width -- the width of the rectangle

    This function returns the area of the rectangle
    '''
    area = height*width
    return area
```

```python
area1 = compute_rectangle_area(5, 6)
print('area 1: ', area1)

area2 = compute_rectangle_area(height=5, width=6)
print('area 2: ', area2)

area3 = compute_rectangle_area(width=6, height=5)
print('area 3: ', area3)

area4 = compute_rectangle_area(width=6, height=5)
print('area 4: ', area4)

area5 = compute_rectangle_area()
print('area 5: ', area5)
```

```
area 1:  30
area 2:  30
area 3:  30
area 4:  30
area 5:  0
```

Simple algorithm analysis

Counting Primitive Operations

```python
def compute_rectangle_area(height=0, width=0):
    '''
    This function aims to compute area for a rectangle.

    height -- the height of the rectangle
    width -- the width of the rectangle

    This function returns the area of the rectangle
    '''
    area = height*width
    return area
```

```python
area1 = compute_rectangle_area(5, 6)
print('area 1: ', area1)

area2 = compute_rectangle_area(height=5, width=6)
print('area 2: ', area2)

area3 = compute_rectangle_area(width=6, height=5)
print('area 3: ', area3)

area4 = compute_rectangle_area(width=6, height=5)
print('area 4: ', area4)

area5 = compute_rectangle_area()
print('area 5: ', area5)
```

9 op

10 op

10 op

10 op

8 op

```
area 1:  30
area 2:  30
area 3:  30
area 4:  30
area 5:  0
```

O(1)

Simple algorithm analysis

Time Complexity of Condition

```
[10] num = 3
     if num > 0:
         print(num, "is a positive number.")
     print("This is always printed.")

     num = -1
     if num > 0:
         print(num, "is a positive number.")
     print("This is also always printed.")
```

```
3 is a positive number.
This is always printed.
This is also always printed.
```

```
def find_max_number(number_a, number_b):
    number_max = 0
    if number_a > number_b:
        number_max = number_a
    else:
        number_max = number_b
    return number_max

print(find_max_number(5, 7))


number_a = -5
number_b = 8
print(find_max_number(number_a, number_b))
```

```
7
8
```

Simple algorithm analysis

Time Complexity of Condition

```
[10] num = 3
     if num > 0:
         print(num, "is a positive number.")
     print("This is always printed.")

     num = -1
     if num > 0:
         print(num, "is a positive number.")
     print("This is also always printed.")
```

```
3 is a positive number.
This is always printed.
This is also always printed.
```

8 op

O(1)

```
def find_max_number(number_a, number_b):
    number_max = 0
    if number_a > number_b:
        number_max = number_a
    else:
        number_max = number_b
    return number_max

print(find_max_number(5, 7))

number_a = -5
number_b = 8
print(find_max_number(number_a, number_b))
```

8 op

10 op

```
7
8
```

AI VIET NAM
@aivietnam.edu.vn

Simple algorithm analysis

Time Complexity of Condition

Input: a and b

if $a = 0$ then $y = b^2$

if $a = K$ then $y = \sqrt{b}$

```
[15] import math

     def function_3(a, b):
         result = 0

         if a>0:
             result = b*b
         elif a<0:
             result = math.sqrt(b)
         return result

     print(function_3(2, 4))


     print(function_3(-2, 4))
```

```
16
2.0
```

```
import math

def function_3(a, b):
    result = 0

    if a>0:
        result = b*b
        result = result + a
    elif a<0:
        result = math.sqrt(b)
    return result

print(function_3(2, 4))


print(function_3(-2, 4))
```

```
18
2.0
```

Simple algorithm analysis

Time Complexity of Condition

Input: a and b

if $a = 0$ then $y = b^2$

if $a = K$ then $y = \sqrt{b}$

```python
[15] import math

def function_3(a, b):
    result = 0

    if a>0:
        result = b*b
    elif a<0:
        result = math.sqrt(b)
    return result

print(function_3(2, 4))

print(function_3(-2, 4))
```

9 op

9 op

```
16
2.0
```

O(1)

```python
import math

def function_3(a, b):
    result = 0

    if a>0:
        result = b*b
        result = result + a
    elif a<0:
        result = math.sqrt(b)
    return result

print(function_3(2, 4))

print(function_3(-2, 4))
```

9 op

9 op

```
18
2.0
```

Simple algorithm analysis

Time Complexity of For Loops

n = number of iterations * static statements

```
[18] n = 5
     for i in range(n):
         print(i)
```

```
0
1
2
3
4
```

```
n = 5
total = 0
for i in range(n):
    total = total + i
    print(total)
```

```
0
1
3
6
10
```

AI VIET NAM
@aivietnam.edu.vn

Simple algorithm analysis

Time Complexity of For Loops

n = number of iterations * static statements

```
[18] n = 5
     for i in range(n):
         print(i)
```

```
0
1
2
3
4
```

16 op

3n + 1 op

O(n)

```
n = 5
total = 0
for i in range(n):
    total = total + i
    print(total)
```

```
0
1
3
6
10
```

27 op

5n + 2 op

O(n)

Simple algorithm analysis

Time Complexity of For Loops

```
[29] x = 10
     y = 20
     n = 10
     for i in range(n):
         z = x + y
         t = x - y
```

```
[23] x = 10
     y = 20
     n = 10
     for i in range(n*n):
         z = x + y
         t = x - y
```

```
x = 10
y = 20
n = 10
for i in range(2*n):
    z = x + y
    t = x - y
```

```
[30] x = 10
     y = 20
     n = 10
     for i in range(n):
         for j in range(5):
             z = x + y
             t = x - y
```

```
[33] x = 10
     y = 20
     n = 10
     for i in range(n):
         x = x + i
         y = y + i
         for j in range(5):
             z = x + y
             t = x - y
```

```
[32] x = 10
     y = 20
     n = 10
     for i in range(n):
         x = x + i
         y = y + i
         for j in range(n):
             z = x + y
             t = x - y
```

AI VIET NAM
@aivietnam.edu.vn

Simple algorithm analysis

Time Complexity of For Loops

```
[29] x = 10
     y = 20
     n = 10
     for i in range(n):
         z = x + y
         t = x - y
```

63 op
$6n + 3$ op
$O(n)$

```
[23] x = 10
     y = 20
     n = 10
     for i in range(n*n):
         z = x + y
         t = x - y
```

603
$6n^2 + 3$ op
$O(n^2)$

```
x = 10
y = 20
n = 10
for i in range(2*n):
    z = x + y
    t = x - y
```

123
$12n + 3$ op
$O(n)$

```
[30] x = 10
     y = 20
     n = 10
     for i in range(n):
         for j in range(5):
             z = x + y
             t = x - y
```

323 op
$32n + 3$ op
$O(n)$

```
[33] x = 10
     y = 20
     n = 10
     for i in range(n):
         x = x + i
         y = y + i
         for j in range(5):
             z = x + y
             t = x - y
```

363 op
$36n + 3$ op
$O(n)$

```
[32] x = 10
     y = 20
     n = 10
     for i in range(n):
         x = x + i
         y = y + i
         for j in range(n):
             z = x + y
             t = x - y
```

663 op
$n(6n+6) + 3$ op
$O(n^2)$

Simple algorithm analysis

Time Complexity of While Loops

```
[46] n = 10
     i = 0
     while i < n:
         i = i + 1
```

```
[47] n = 10
     i = 5
     total = 0
     while i < n:
         total = total + i
         i = i + 1
```

```
n = 10
i = 0
total = 0
while i < n:
    total = total + i
    i = i + 1
    j = 0
    while j < n:
        total = total + j
        j = j + 1
print(total)
```

```
495
```

AI VIET NAM
@aivietnam.edu.vn

Simple algorithm analysis

Time Complexity of While Loops

```
[46] n = 10
     i = 0
     while i < n:
         i = i + 1
```

32 op
$3n + 2$ op
$O(n)$

```
[47] n = 10
     i = 5
     total = 0
     while i < n:
         total = total + i
         i = i + 1
```

53 op
$5n + 3$ op
$O(n)$

```
n = 10
i = 0
total = 0
while i < n:
    total = total + i
    i = i + 1
    j = 0
    while j < n:
        total = total + j
        j = j + 1
print(total)
```

```
495
```

554 op
$n(5n + 6) + 4$ op
$O(n^2)$

Simple algorithm analysis

Other example

```
[49] n = 10
     for i in range(n):
         for j in range(n):
             for k in range(n):
                 print(i*j*k)
```

```
[48] def is_even(value):
         result = value % 2 == 0
         return result

     n = 10
     total = 0
     for i in range(n):
         if is_even(i):
             print(i)
         else:
             total = total + i
             print(total)
```

```
def reverse_number(n):
    r_num = 0
    while n > 0:
        reminder = n % 10
        r_num = (r_num * 10) + reminder
        n = n // 10 #//: chia làm tròn xuống
    return r_num

print(reverse_number(12))

print(reverse_number(n=123456789))
```

```
21
987654321
```

AI VIET NAM
@aivietnam.edu.vn

Simple algorithm analysis

Other example

```python
def factorial_fcn(x):
    res = 1
    for i in range(x):
      res *= (i+1)
    return res

def approx_cos(x, n):
    cos_approx = 0
    for i in range(n+1):
        coef = (-1)**i
        num = x**(2*i)
        denom = factorial_fcn(2*i)
        cos_approx += ( coef ) * ((num)/(denom))
    return cos_approx

approx_cos(x=3.14, n=10)
```

```
-0.9999987316527259
```

```python
def factorial_fcn(x):
    res = 1
    for i in range(x):
      res *= (i+1)
    return res

def approx_sin(x, n):
    sin_approx = 0
    for i in range(n+1):
        coef = (-1)**i
        num = x**(2*i+1)
        denom = factorial_fcn(2*i+1)
        sin_approx += ( coef ) * ((num)/(denom))

    return sin_approx

approx_sin(x=3.14, n=10)
```

```
0.0015926529267151343
```

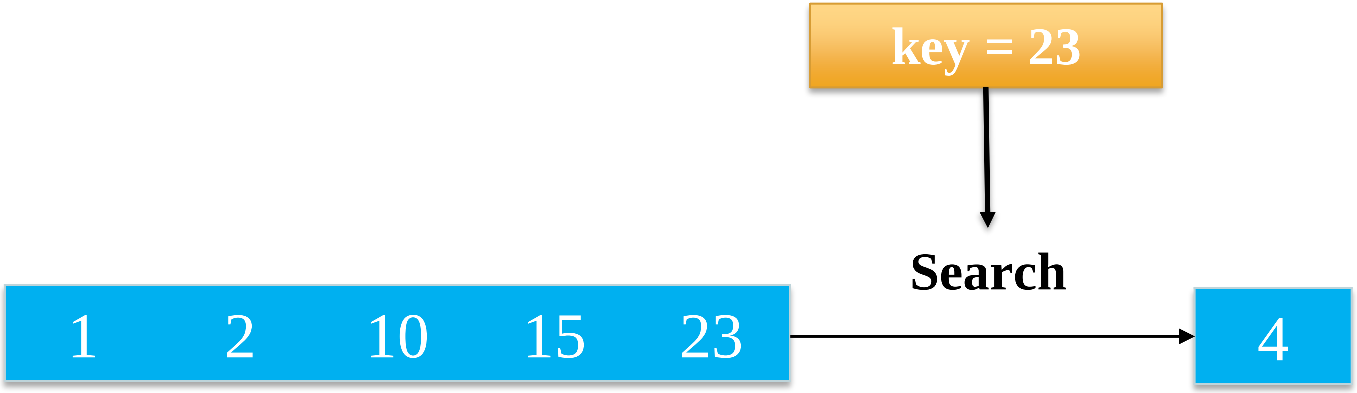
Objectives

Algorithm design and analysis

Example: Searching problem

Input: a sorted sequence of n number <a_1, a_2,…, a_n>, key

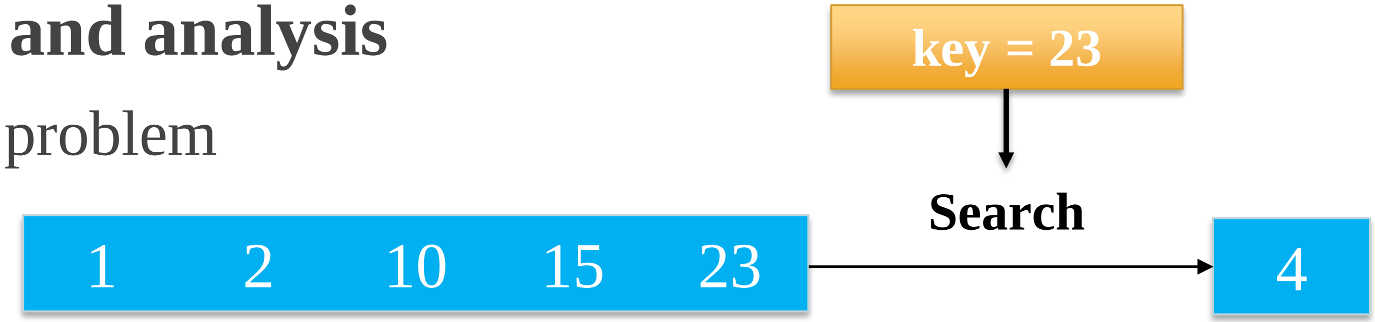
Output: index of key in the sequence if exist, -1 if not exist



Objectives

Algorithm design and analysis

Example: Searching problem



Pseudo code

LINEAR-SEARCH(arr, key)

```
for idx = 0 to (arr.length-1)
    element = arr[i]
    // Check key
    if element == key
        return idx
    return
        -1
```

Python

```python
[15] def linear_search(arr, key):

         n = len(arr)
         for idx in range(n):

             element = arr[idx]
             if element == key:
                 return idx

         return -1

     arr = [1, 2, 10, 15, 23]
     key = 23
     linear_search(arr, key)

     4
```

Objectives

Different searching algorithms

Linear search

```
[15] def linear_search(arr, key):

         n = len(arr)
         for idx in range(n):

             element = arr[idx]
             if element == key:
                 return idx

         return -1

     arr = [1, 2, 10, 15, 23]
     key = 23
     linear_search(arr, key)
```

4

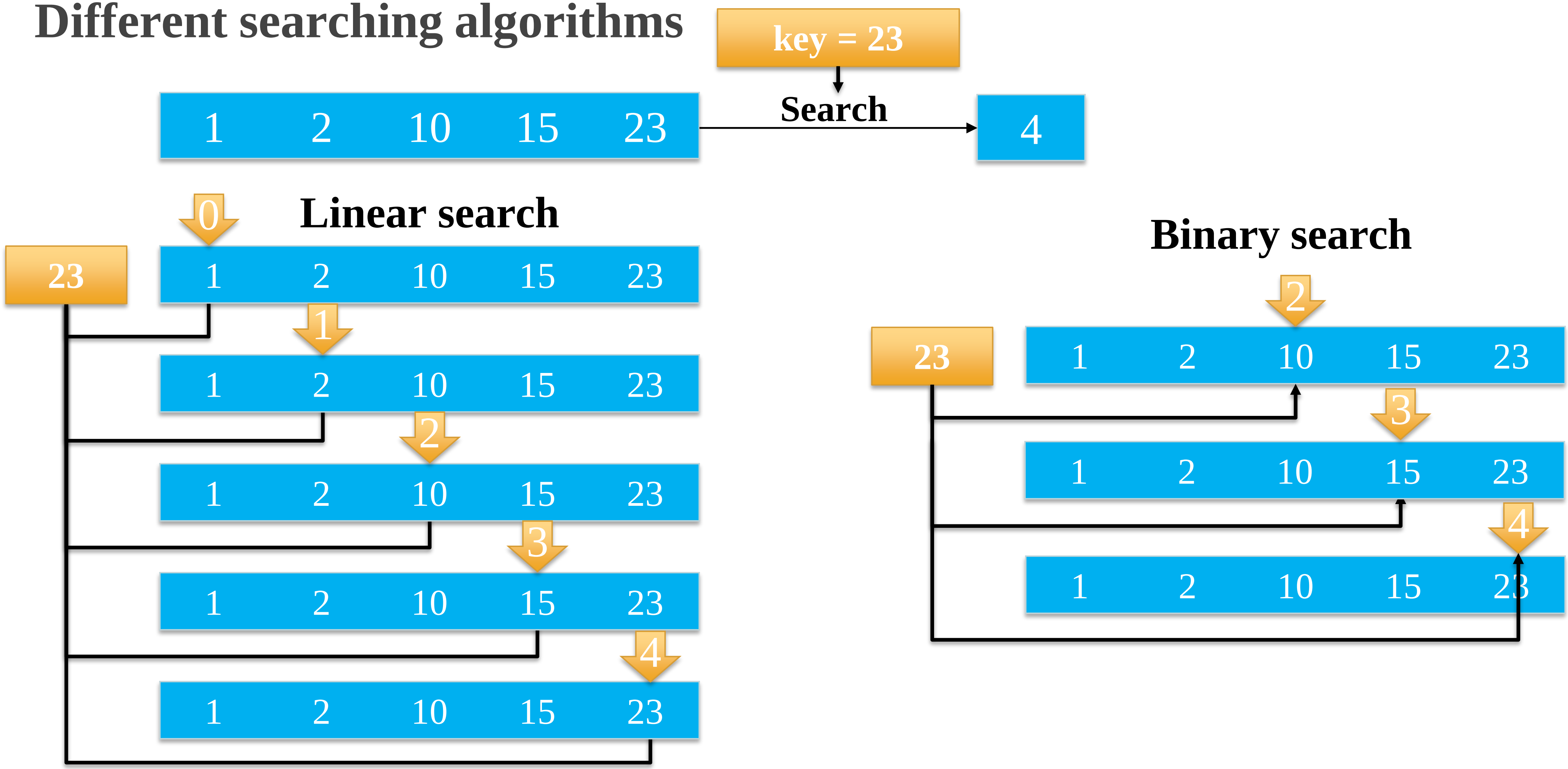
Binary search

```
def binary_search(array, key):
    low = 0
    high = len(arr) - 1

    while low <= high:

        mid = low + (high - low)//2

        if arr[mid] == key:
            return mid

        elif arr[mid] < key:
            low = mid + 1

        else:
            high = mid - 1

    return -1


arr = [1, 2, 10, 15, 23]
key = 23

binary_search(arr, key)
```

4

Objectives

Different searching algorithms



AI VIET NAM
@aivietnam.edu.vn

Objectives

Different searching algorithms

key = 23

1 2 10 15 23 → Search → 4

Linear search

O(n)

```
[15] def linear_search(arr, key):

         n = len(arr)
         for idx in range(n):

             element = arr[idx]
             if element == key:
                 return idx

         return -1

     arr = [1, 2, 10, 15, 23]
     key = 23
     linear_search(arr, key)
```

4

Binary search

O(logn)

```
def binary_search(array, key):
    low = 0
    high = len(arr) - 1

    while low <= high:

        mid = low + (high - low)//2

        if arr[mid] == key:
            return mid

        elif arr[mid] < key:
            low = mid + 1

        else:
            high = mid - 1

    return -1


arr = [1, 2, 10, 15, 23]
key = 23

binary_search(arr, key)
```

4

Objectives

Different searching algorithms

- Suppose $n=10^{10}$ numbers:
  - Linear search: $c_1 n$
  - Binary search: $c_2(logn)$

Case 1: The same programmer ($c_1=c_2=1$), computer (1 billion/second)

Linear search	Binary search
$1*(10^{10})$ instructions/10^9 instructions per second = 10 seconds	$1*\log(10^{10})$ instructions/10^9 instructions per second $\approx 3.3*10^{-8}$ seconds

Objectives

Different searching algorithms

- Suppose $n=10^{10}$ numbers:
  - Linear search: $c_1 n$
  - Binary search: $c_2(\log n)$

Case 2: Best programmer (c_1=1), bad programmer (c_2=50)

The same computer (1 billion/second)

	Linear search	Binary search
Best programmer	$1*(10^{10}) / 10^9 = 10$ seconds	$1*\log(10^{10}) / 10^9 \approx 3.3*10^{-8}$ seconds
Bad programmer	$50*(10^{10}) / 10^9 = 500$ seconds	$50*\log(10^{10}) / 10^9 \approx 1.65*10^{-6}$ seconds

Objectives

Different searching algorithms

- Suppose $n=10^{10}$ numbers:
  - Linear search: $c_1 n$
  - Binary search: $c_2(\log n)$

Case 3: The same programmer ($c_1=c_2=1$), language (python),

Computer A (1 billion/second), computer B (10 million/second)

	Linear search	Binary search
Computer A	$1*(10^{10}) / 10^9 = 10$ seconds	$1*\log(10^{10}) / 10^9 \approx 3.3*10^{-8}$ seconds
Computer B	$1*(10^{10}) / 10^7 = 10^3$ seconds	$1*\log(10^{10}) / 10^7 \approx 3.3*10^{-6}$ seconds

Objectives

Different searching algorithms

- Suppose $n=10^{10}$ numbers:
  - Linear search: $c_1 n$
  - Binary search: $c_2(logn)$

Case 4: <A>: best programmer ($c_1=1$), computer A (1 billion/second)

<B>: bad programmer ($c_2=50$), computer B (10 million/second)

	Linear search	Binary search
<A>	$1*(10^{10})\ /10^9 = 10$ seconds	$1*\log(10^{10})\ /10^9 \approx 3.3*10^{-8}$ seconds
<B>	$50*(10^{10})\ /10^7 = 5*10^4$ seconds	$50*\log(10^{10})\ /10^7 \approx 1.66.10^{-4}$ seconds

=> Binary search on <B> is 60.000 times faster than binary search on A

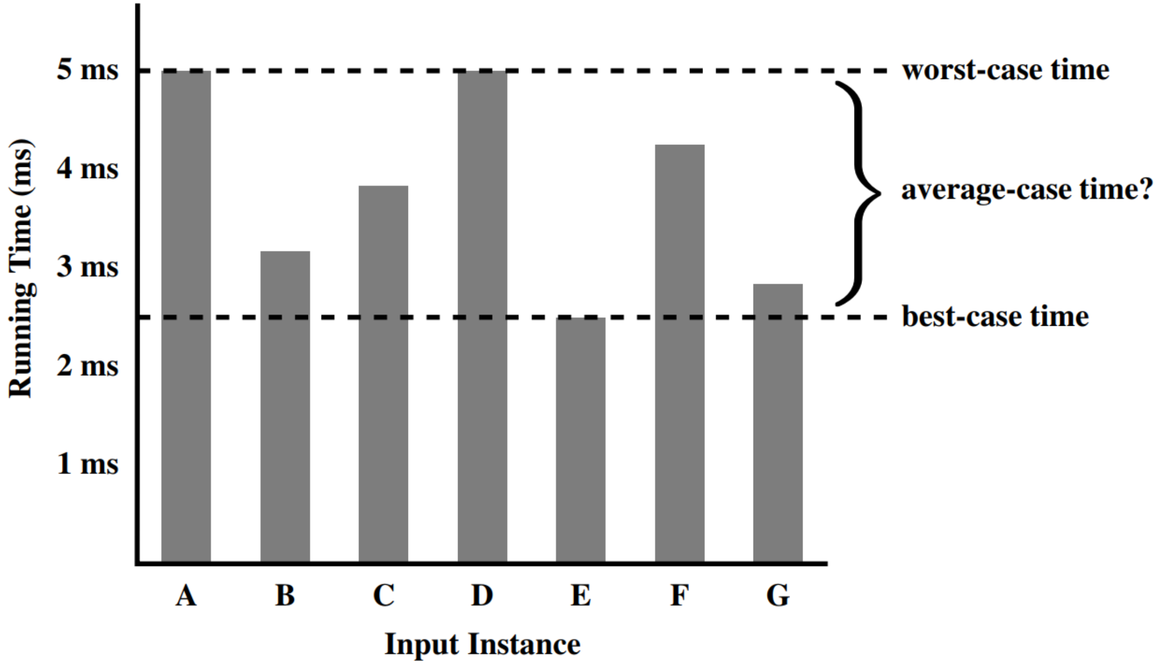
AI VIET NAM
@aivietnam.edu.vn

Computational complexity

Running time

Takes into account all possible inputs

- Worst case, best case, average case
- For some algorithms, worst case occurs often, average case is often roughly as bad as the worst case. So generally, worse case running time



AI VIET NAM
@aivietnam.edu.vn

Computational complexity

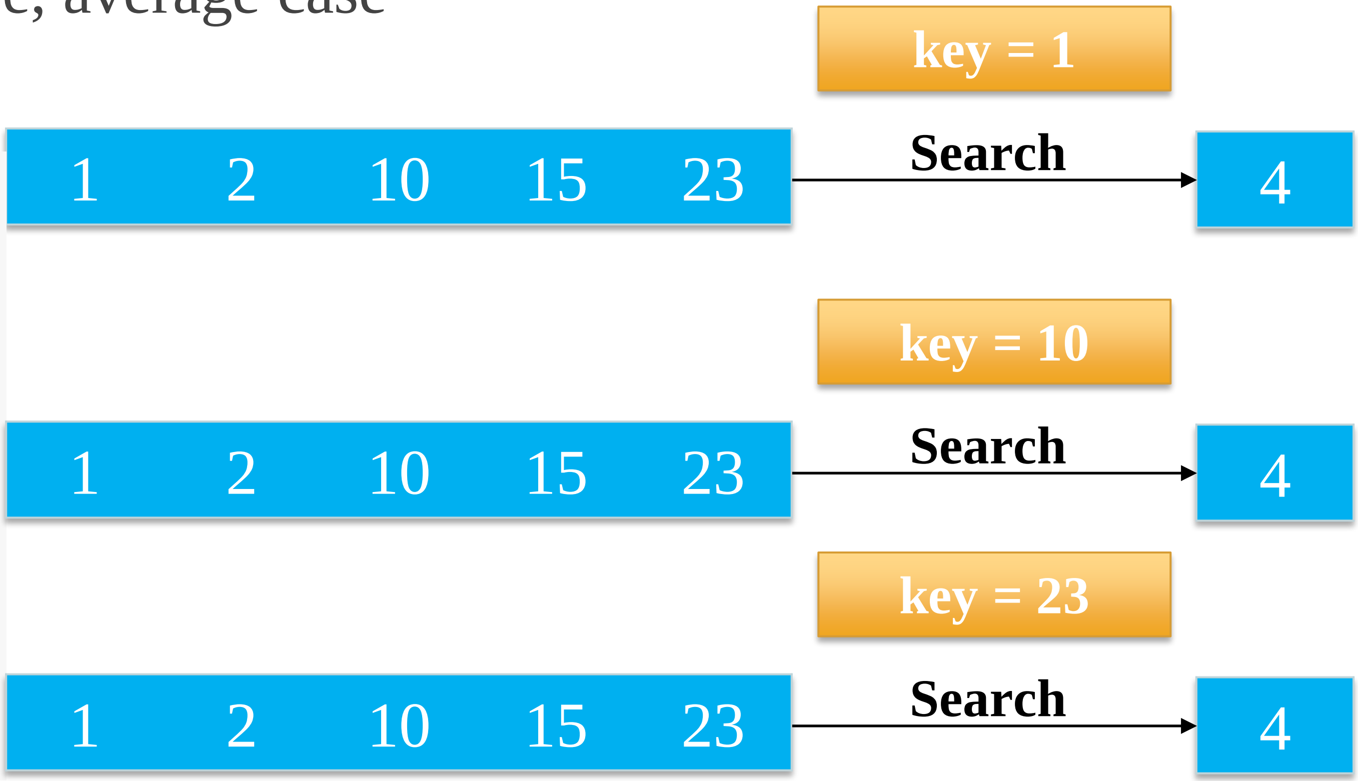
Running time

Takes into account all possible inputs

➢ Worst case, best case, average case

Linear search

```
[15] def linear_search(arr, key):

         n = len(arr)
         for idx in range(n):

             element = arr[idx]
             if element == key:
                 return idx

         return -1

     arr = [1, 2, 10, 15, 23]
     key = 23
     linear_search(arr, key)

     4
```



AI VIET NAM
@aivietnam.edu.vn

Computational complexity

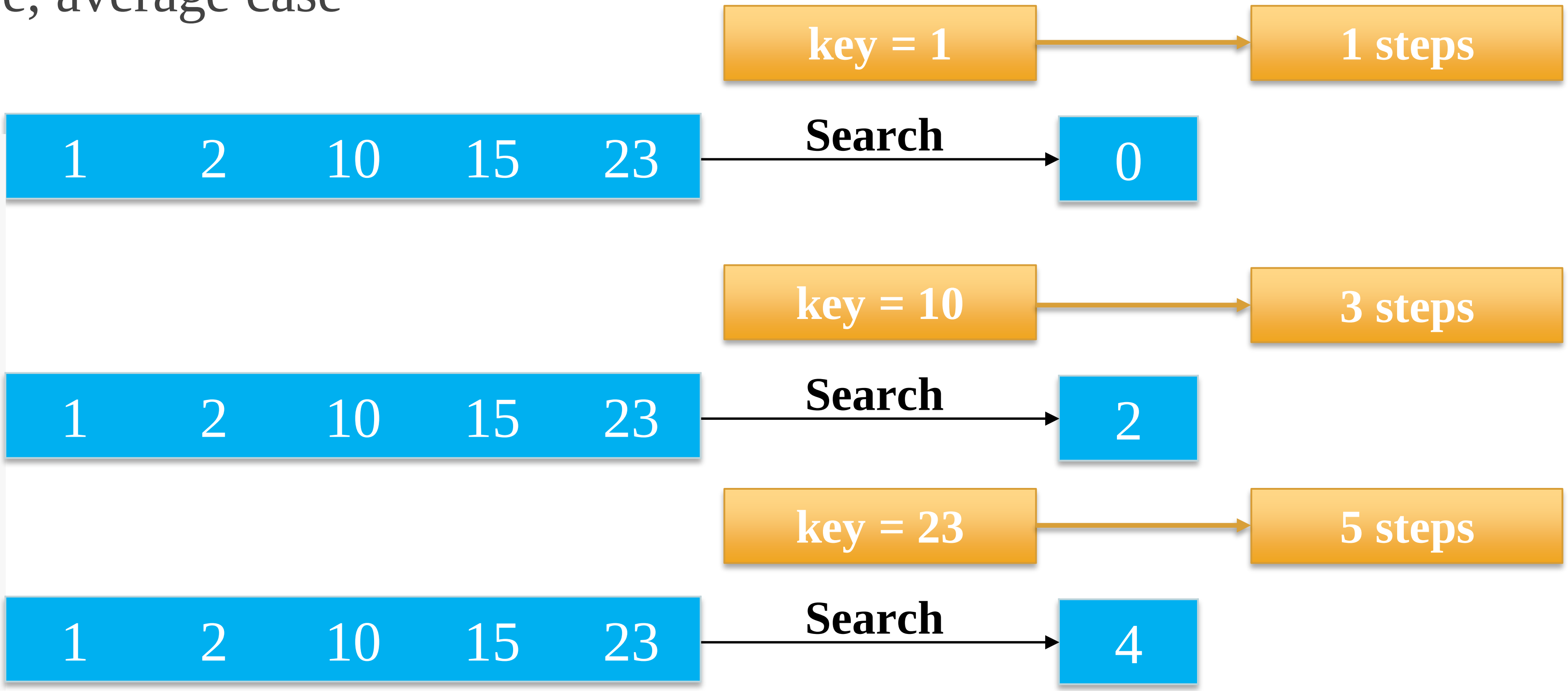
Running time

Takes into account all possible inputs

- Worst case, best case, average case

Linear search

```
[15] def linear_search(arr, key):

         n = len(arr)
         for idx in range(n):

             element = arr[idx]
             if element == key:
                 return idx

         return -1

     arr = [1, 2, 10, 15, 23]
     key = 23
     linear_search(arr, key)
```



Computational complexity

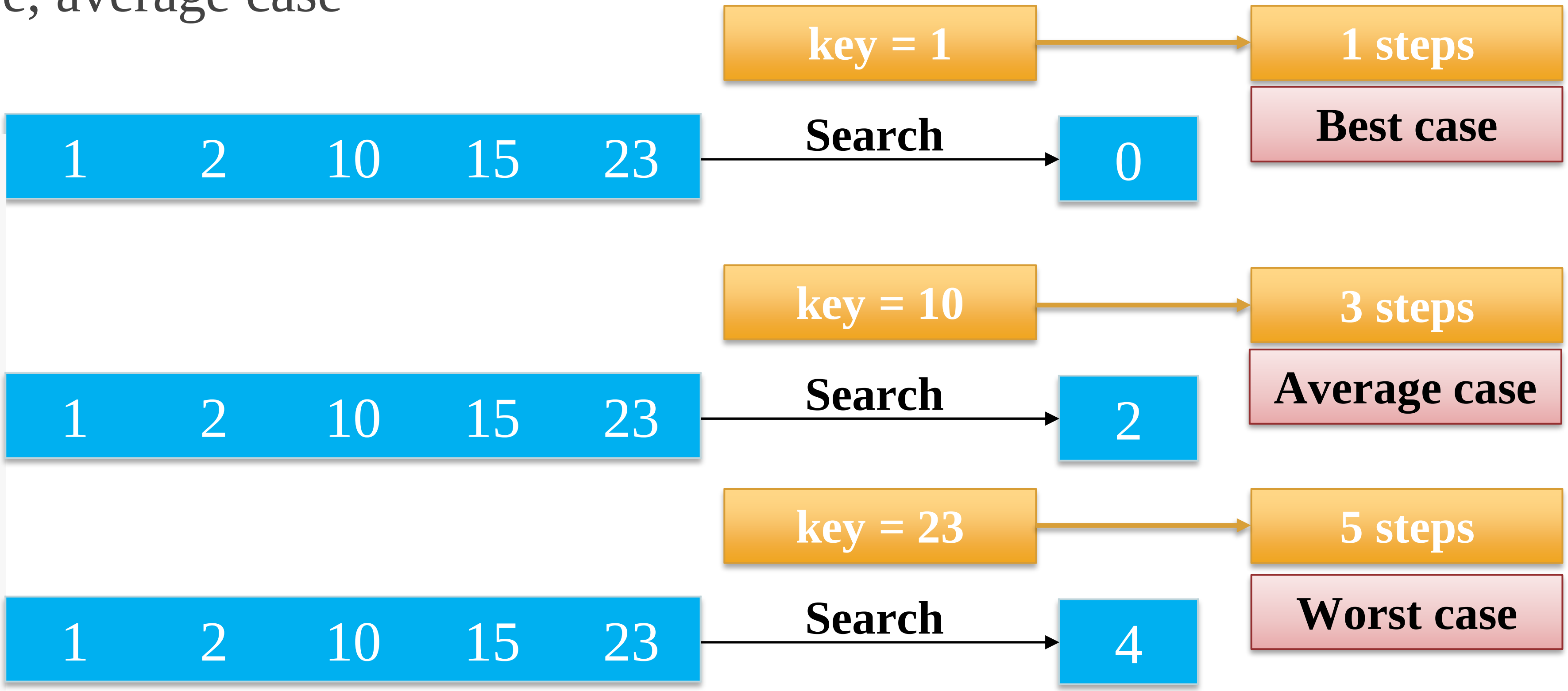
Running time

Takes into account all possible inputs

➢ Worst case, best case, average case

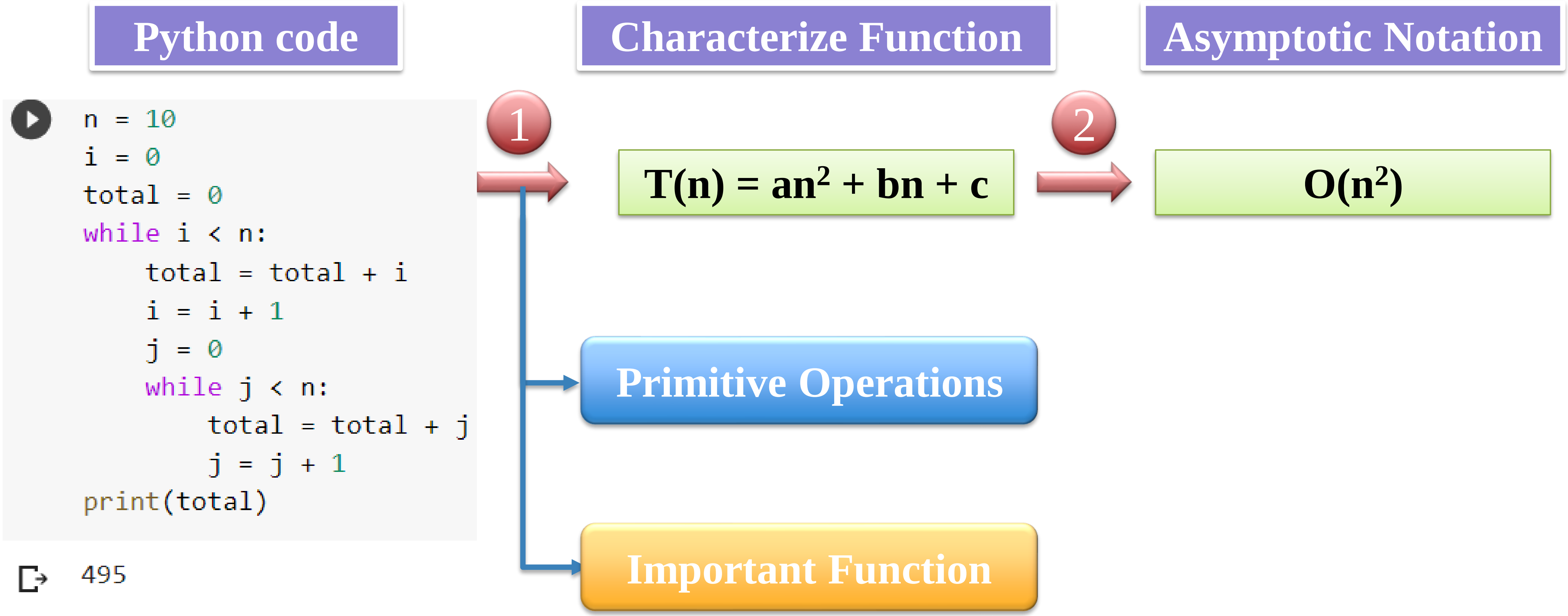
Linear search

```
[15] def linear_search(arr, key):

         n = len(arr)
         for idx in range(n):

             element = arr[idx]
             if element == key:
                 return idx

         return -1

     arr = [1, 2, 10, 15, 23]
     key = 23
     linear_search(arr, key)

     4
```



Computational complexity

Steps to calculate computational complexity



Computational complexity

The seven most important functions

- The constant function: $f(n) = c$
  - Any argument n, $f(n)$ assigns the value c.
  - $c = 5, 10, 2^{10},\ldots$
  - Basic operation: comparing two numbers,…

```python
def find_max_number(number_a, number_b):
    number_max = 0
    if number_a > number_b:
        number_max = number_a
    else:
        number_max = number_b
    return number_max
```

Computational complexity

The seven most important functions

- The logarithm function:

  $f(n) = \log_b n \quad$ if and only if $b^x = n, \quad b > 1$

  - b: base of the logarithm (computer science is 2)
  - $\log_b 1 = 0$
  - For any real numbers a>0, b>1, c>0, n:

$\log_b ac = \log_b a + \log_b c \qquad \log_b a/c = \log_b a - \log_b c$

$\log_b a^n = n\log_b a \qquad b^{\log_d a} = a^{\log_d b}$

$\log_b a = \log_d a / \log_d b$

```python
def binary_search(array, key):
    low = 0
    high = len(arr) - 1

    while low <= high:

        mid = low + (high - low)//2

        if arr[mid] == key:
            return mid

        elif arr[mid] < key:
            low = mid + 1

        else:
            high = mid - 1

    return -1


arr = [1, 2, 10, 15, 23]
key = 23

binary_search(arr, key)
```

```
4
```

Computational complexity

The seven most important functions

- The linear function: $f(n) = n$
  - The best running time to achieve for any algorithm
  - Comparing a number x to each element of a sequence of size n
- The N-Log-N function: $f(n) = n\log n$
  - The fastest possible algorithms for sorting

```
[29] x = 10
     y = 20
     n = 10
     for i in range(n):
         z = x + y
         t = x - y
```

```
[47] n = 10
     i = 5
     total = 0
     while i < n:
         total = total + i
         i = i + 1
```

Computational complexity

The seven most important functions

- The Quadratic function: $f(n) = n^2$
  - Appears often in algorithm analysis: nested loops
  - For any integer $n \geq 1$:

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + (n-2) + (n-1) + n = \frac{n(n+1)}{2}$$

```
[32] x = 10
     y = 20
     n = 10
     for i in range(n):
         x = x + i
         y = y + i
         for j in range(n):
             z = x + y
             t = x - y
```

```
n = 10
i = 0
total = 0
while i < n:
    total = total + i
    i = i + 1
    j = 0
    while j < n:
        total = total + j
        j = j + 1
```

Computational complexity

The seven most important functions

- The Cubic function: $f(n) = n^3$
  - Polynomials:

$$f(n) = a_0 + a_1 n + a_2 n^2 + \cdots + a_d n^d = \sum_{i=0}^{d} a_i n^i$$

$a_0, a_1, a_2, \ldots$: constants => coefficients, $a_d \neq 0$. d: highest power => degree

```
[49] n = 10
     for i in range(n):
         for j in range(n):
             for k in range(n):
                 print(i*j*k)
```

Computational complexity

The seven most important functions

- The Exponential function: $f(n) = b^n$
  - b: positive constant => base
  - n => exponent

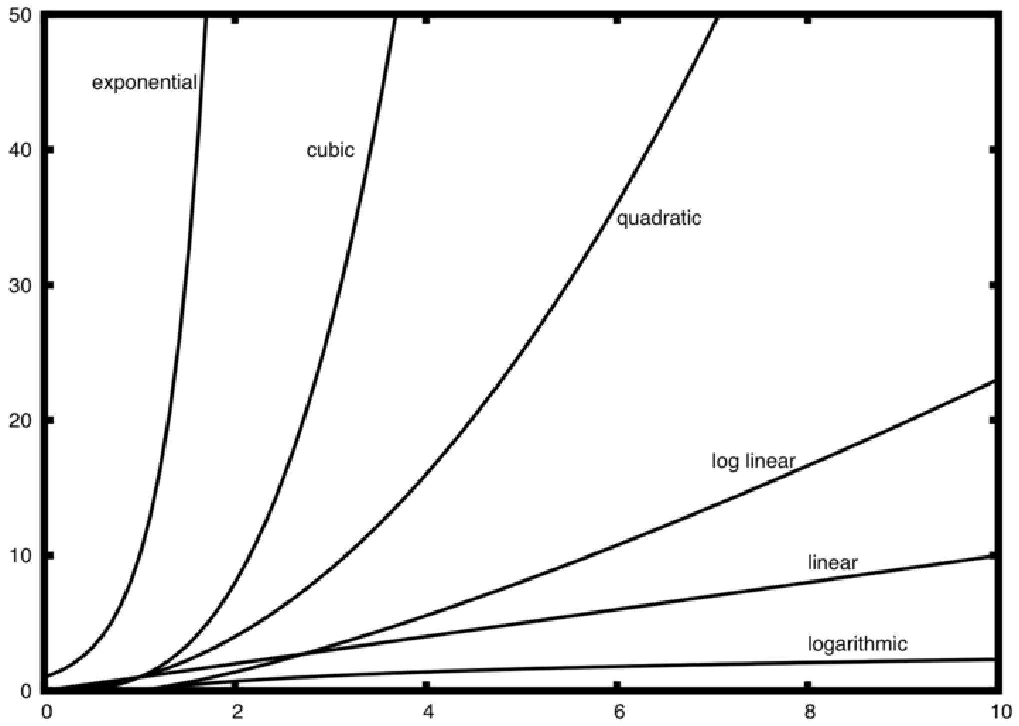
For any $n \geq 0$, $a > 0$, $a \neq 1$: $\sum_{i=0}^{n} a^i = 1 + a + a^2 + \ldots + a^n = \frac{a^{n+1}-1}{a-1}$

Computational complexity

The seven most important functions

Comparing growth rates (the order of growth)

constant	logarithm	linear	n-log-n	quadratic	cubic	exponential
1 (c)	logn	n	nlogn	n^2	n^3	a^n



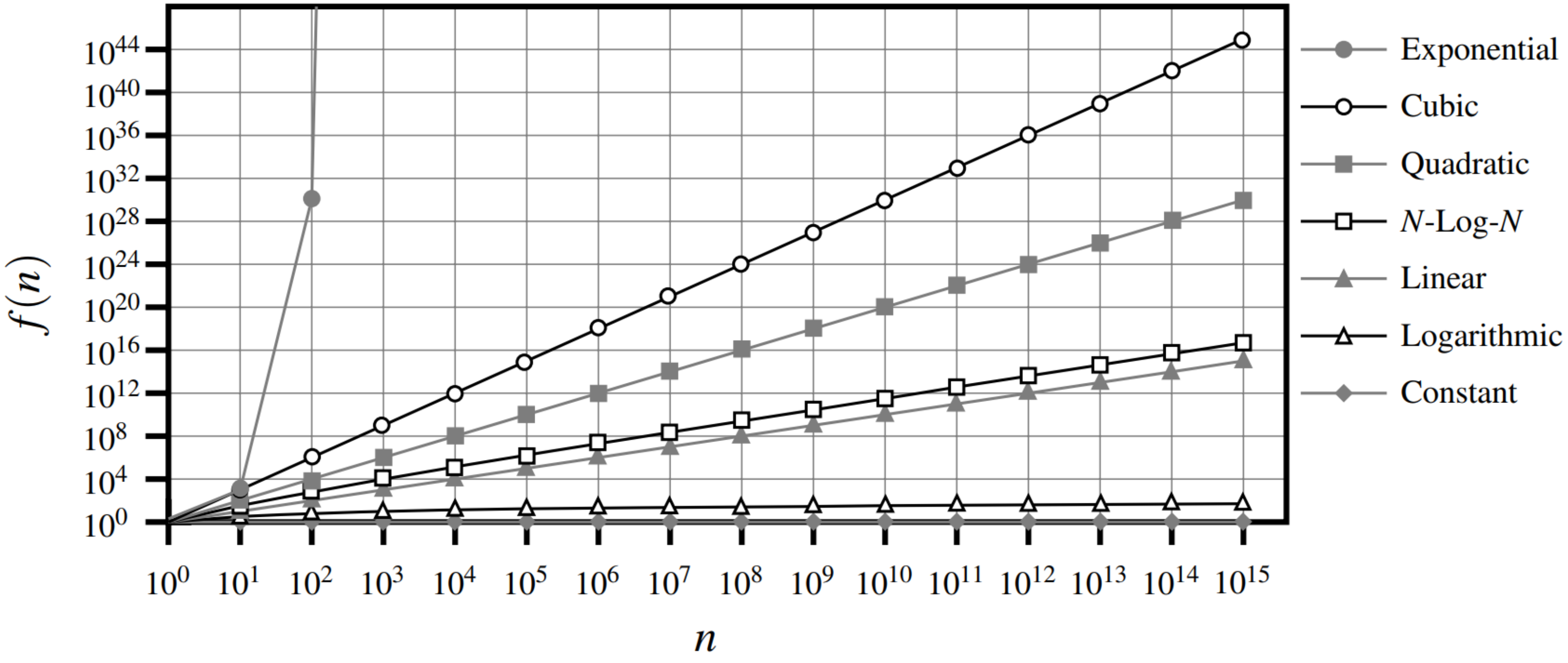
AI VIET NAM
@aivietnam.edu.vn

Computational complexity

The seven most important functions

Comparing growth rates (the order of growth)

constant	logarithm	linear	n-log-n	quadratic	cubic	exponential
1 (c)	logn	n	nlogn	n^2	n^3	a^n



Computational complexity

Steps to calculate computational complexity

Python code → 1 → Characterize Function

```
[6]  S = [1, 2, 3]
     n = len(S)
     for i in range(n):
         for j in range(n):
             total = S[i] + S[j]
             print(total)
         print('- - - -')
```

Computational complexity

Steps to calculate computational complexity

Python code → (1) → Characterize Function

$T(n) = 7n^2 + 4n + 3$

```
[6]  S = [1, 2, 3]
     n = len(S)
     for i in range(n):
         for j in range(n):
             total = S[i] + S[j]
             print(total)
         print('- - - -')
```

- `S = [1, 2, 3]`, `n = len(S)`: 2 op
- `for i in range(n):`: 2 op
- `for j in range(n):`: 2 op
- `total = S[i] + S[j]`: 4 op
- `print(total)`: 2 op
- inner loop: 7n + 1 op
- `print('- - - -')`: 1 op
- outer loop: (7n + 4)n + 1 op

AI VIET NAM
@aivietnam.edu.vn

Computational complexity

Steps to calculate computational complexity

Python code → (1) → Characterize Function

$T(n) = an^2 + bn + c$

```
[6]  S = [1, 2, 3]
     n = len(S)
     for i in range(n):
         for j in range(n):
             total = S[i] + S[j]
             print(total)
         print('- - - -')
```

cost	times

Computational complexity

Steps to calculate computational complexity

Python code → (1) → Characterize Function

$T(n) = an^2 + bn + c$

```
[6]  S = [1, 2, 3]
     n = len(S)
     for i in range(n):
         for j in range(n):
             total = S[i] + S[j]
             print(total)
         print('- - - -')
```

cost	times
c_0	1
c_1	1
c_2	n+1
c_3	$n*(n+1) = n^2+n$
c_4	$n*n = n^2$
c_5	$n*n = n^2$
c_6	n

$$T(n) = (c_3+c_4+c_5)n^2 + (c_2+c_3+c_6)n + (c_0+c_1+c_2)$$

Computational complexity

Steps to calculate computational complexity

Python code

```
[6]  S = [1, 2, 3]
     n = len(S)
     for i in range(n):
         for j in range(n):
             total = S[i] + S[j]
             print(total)
         print('- - - -')
```

1

Characterize Function

$T(n) = an^2 + bn + c$

Primitive Operations

Important Function

2

Asymptotic Notation

$O(n^2)$

Asymptotic Analysis

AI VIET NAM
@aivietnam.edu.vn

Asymptotic Analysis

"Big-Oh" Notation

- If f(n) and g(n): two non-negative functions of non-negative arguments

  f(n) is O(g(n)) if there is a real constant c>0 and an integer constant $n_0 \geq 1$:

  $$f(n) \leq cg(n), \text{ for } n \geq n_0$$

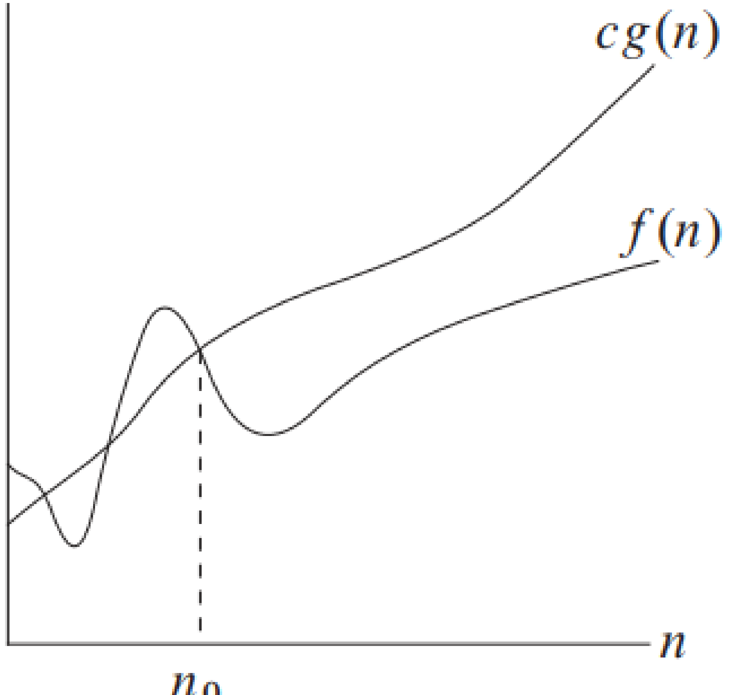
  "less-than-or-equal-to"

- **Example**: f(n) = 8n + 5 is O(n)

  Find c>0 and $n_0 \geq 1$

  $8n + 5 \leq cn$, for every integer $n \geq n_0$

  => A possible choice is c = 13 and $n_0 = 1$



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

Asymptotic Analysis

“Big-Oh” Notation

- Some properties of the Big-Oh Notation
  - Lower order items are ignored, just keep the highest order item
  - The constant coefficients are ignored
  - The rate (/order) of growth possesses the highest significance
- **Example**:

  $f(n) = 5n^4 + 3n^3 + 2n^2 + 4n + 1$ is $O(n^4)$

  $f(n) = 5n^4 + 3n^3 + 2n^2 + 4n + 1 \le (5+3+2+4+1)n^4 = cn^4$

  $\Rightarrow c = 15, n \ge n_0 = 1$

  $f(n)$ is a polynomial of degree d, $a_d > 0 \Rightarrow f(n)$ is $O(n^d)$

Computational complexity

"Big-Oh" Notation

Some rules:

$f(n) = 7n + 7$, $g(n) = 3n\log n$

- $O(c.f(n)) = O(f(n))$

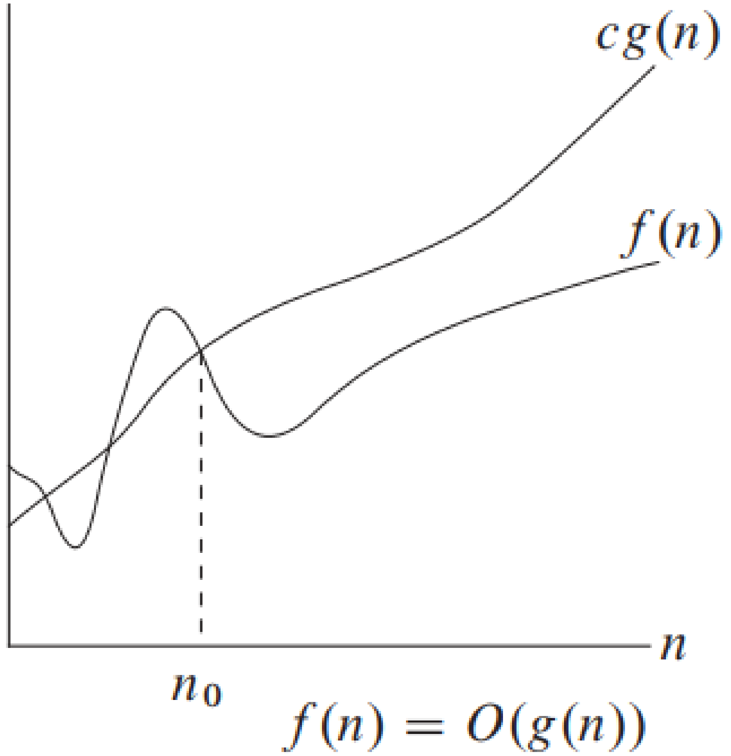
  => $f(n)$ is $O(n)$

- $O(f(n) + g(n)) = O(\max(f(n), g(n)))$

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

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

  => $T(n)$ is $O(n^2\log n)$



Asymptotic Analysis

"Big-Oh" Notation

- If f(n) and g(n): two non-negative functions of non-negative arguments

  f(n) is O(g(n)) if there is a real constant c>0 and an integer constant $n_0 \geq 1$:

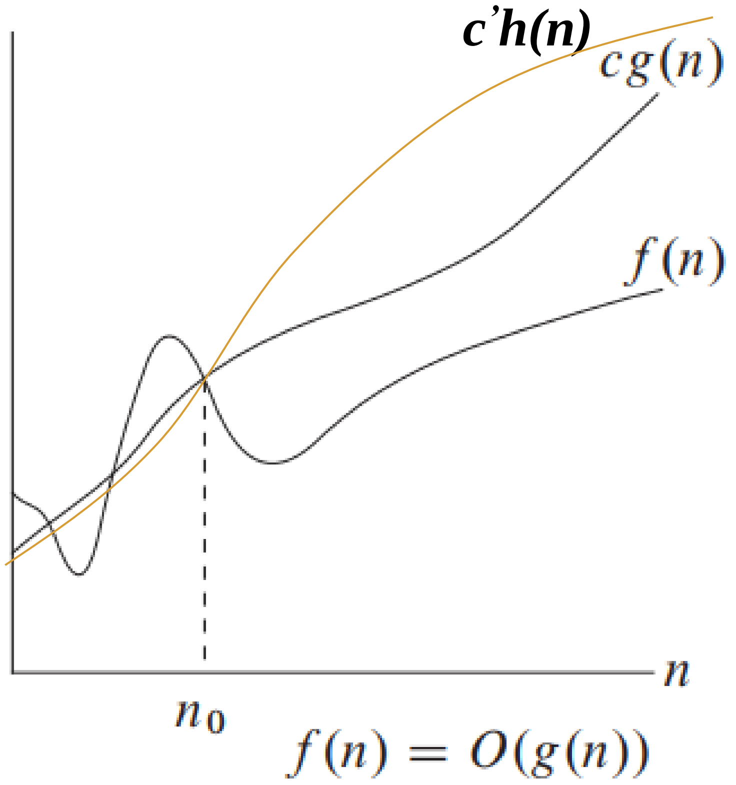
  $$f(n) \leq cg(n), \text{ for } n \geq n_0$$

- **Example**: $f(n) = 8n^3 + 5n$

$f(n) = 13n^3$
=> f(n) is $O(n^3)$

$h(n) = 13n^5$
=> f(n) is $O(n^5)$

TRUE???
f(n) is ???



Asymptotic Analysis

"Big-Oh" Notation

- If f(n) and g(n): two non-negative functions of non-negative arguments

  f(n) is O(g(n)) if there is a real constant c>0 and an integer constant $n_0 \geq 1$:

  $$f(n) \leq cg(n), \text{ for } n \geq n_0$$

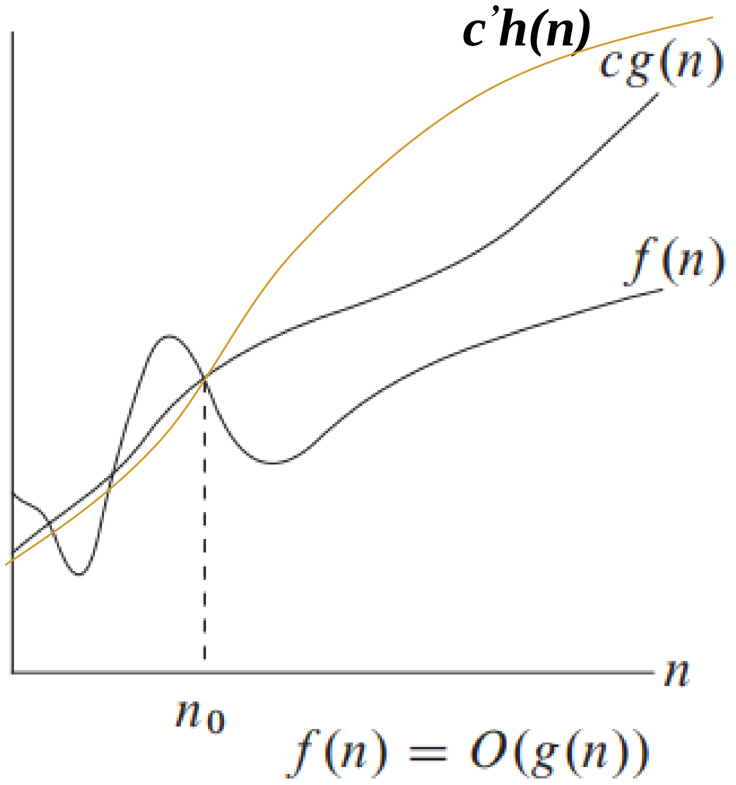
- **Example**: $f(n) = 8n^3 + 5n$

Simplest Terms

$f(n) = 13n^3$
$\Rightarrow f(n)$ is $O(n^3)$

~~$h(n) = 13n^5$~~
~~$\Rightarrow f(n)$ is $O(n^5)$~~

TRUE???
f(n) is ???



AI VIET NAM
@aivietnam.edu.vn

Asymptotic Analysis

"Big-Omega" Notation

- If f(n) and g(n): two non-negative functions of non-negative arguments

  f(n) is $\Omega(g(n))$ if there is a real constant $c>0$ and an integer constant $n_0 \geq 1$:

  $$f(n) \geq cg(n), \text{ for } n \geq n_0$$

  "greater-than-or-equal-to"

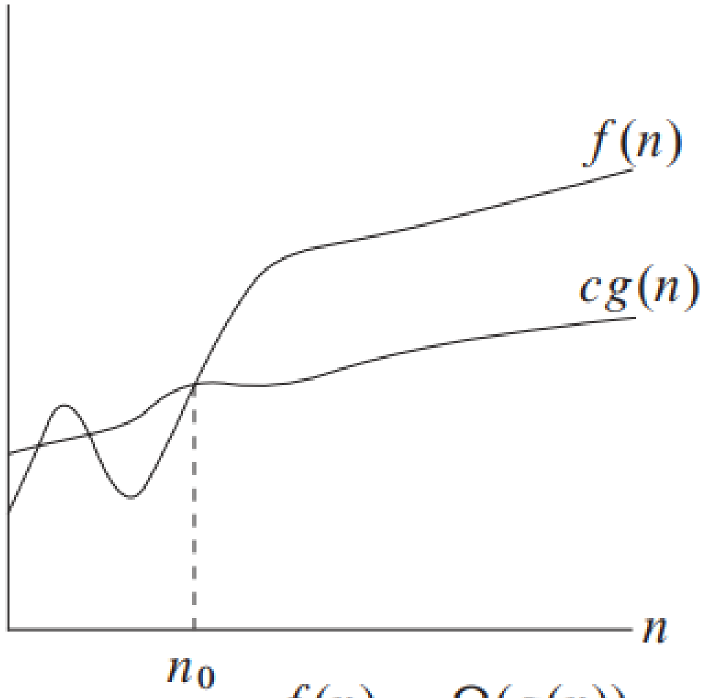
- **Example**: $f(n) = 3n\log n - 2n$ is $\Omega(n\log n)$

  Find $c>0$ and $n_0 \geq 1$:

  $3n\log n - 2n = n\log n + 2n(n\log n - 1)$

  $\geq n\log n$, for every integer $n \geq n_0 = 2$

  $\Rightarrow c = 1, n_0 = 2.$



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

AI VIET NAM
@aivietnam.edu.vn

Asymptotic Analysis

"Big-Theta" Notation

- If f(n) and g(n): two non-negative functions of non-negative arguments

  f(n) is $\Theta(g(n))$ if there is a real constant c_1, c_2>0 and an integer constant $n_0 \geq 1$:

  $$c_1 g(n) \leq f(n) \leq c_2 g(n), \text{ for } n \geq n_0$$

  f(n) is $\Theta(g(n))$ if f(n) is $O(g(n))$ and f(n) is $\Omega(g(n))$

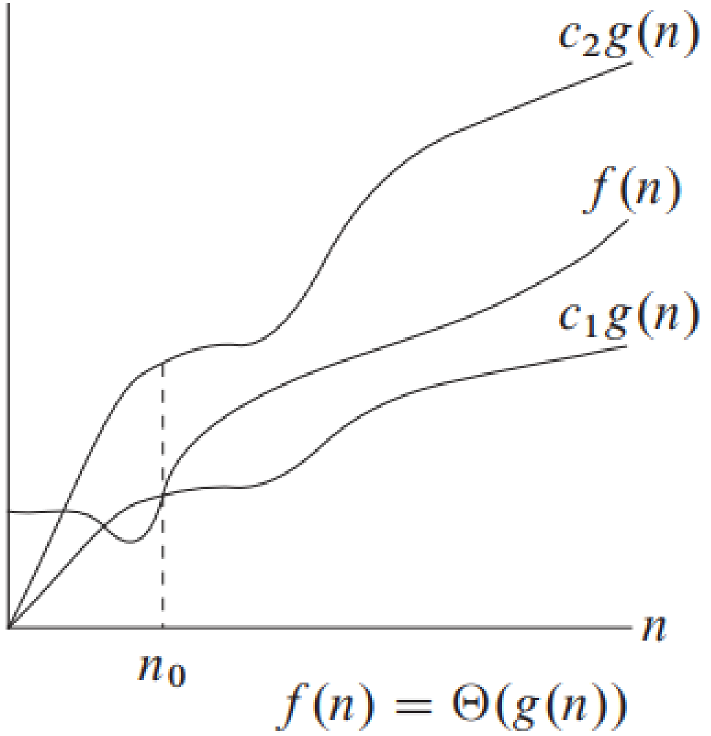
- **Example**: $f(n) = 3n\log n + 4n + 5\log n$ is $\Theta(n\log n)$

  Find c_1, c_2 >0 and $n_0 \geq 1$

  $3n\log n \leq 3n\log n + 4n + 5\log n \leq (3+4+5)n\log n$

  => $3n\log n \leq 3n\log n + 4n + 5\log n \leq 12n\log n$

  => c_1=3, c_2=12 and n_0=2



Computational complexity

Steps to calculate computational complexity

Python code → (1) → **Characterize Function** → (2) → **Asymptotic Notation**

```
[6] S = [1, 2, 3]
    n = len(S)
    for i in range(n):
        for j in range(n):
            total = S[i] + S[j]
            print(total)
        print('- - - -')
```

$$T(n) = an^2 + bn + c$$

$$O(n^2)$$

$$T(n) = (c_3+c_4+c_5)n^2 + (c_2+c_3+c_6)n + (c_0+c_1+c_2)$$
$$\leq (c_0+c_1+2c_2+2c_3+c_4+c_5+c_6)n^2$$
$$= c'n^2$$

For $c' = c_0+c_1+2c_2+2c_3+c_4+c_5+c_6$, $n_0=1$

Exercise

Question 1

a) $(n+1)^5$ is $O(n^5)$

b) 2^{n+1} is $O(2^n)$

c) $5n^2 + 3n\log n + 2n + 5$ is $O(n^2)$

d) $3\log n + 2$ is $O(\log n)$

e) $3n\log n - 2n$ is $\Omega(n\log n)$

f) n^2 is $\Omega(n\log n)$

g) $3n\log n + 4n + 5\log n$ is $\Theta(n\log n)$

Exercise

Question 1

a) $(n+1)^5$ is $O(n^5)$

b) 2^{n+1} is $O(2^n)$

c) $5n^2 + 3n\log n + 2n + 5$ is $O(n^2)$

d) $3\log n + 2$ is $O(\log n)$

e) $3n\log n – 2n$ is $\Omega(n\log n)$

f) n^2 is $\Omega(n\log n)$

g) $3n\log n + 4n + 5\log n$ is $\Theta(n\log n)$

Justification

a) (n+1)5 is O(n5)

$(n+1)^5 = n^5+5n^4+10n^3+10n^2+5n+1$

$\leq (1+5+10+10+5+1)n^5 = cn^5$

$\Rightarrow c=32,\ n_0=1$

b) 2^{n+1} is $O(2^n)$

$2^{n+1} = 2^n.2 = c2^n$

$\Rightarrow c=2,\ n_0=1$

c) $5n^2+3n\log n+2n+5$ is $O(n^2)$

$\leq (5+3+2+5)n^2 = cn^2$

$\Rightarrow c=15,\ n_0=1$

AI VIET NAM
@aivietnam.edu.vn

Exercise

Question 1

a) $(n+1)^5$ is $O(n^5)$

b) 2^{n+1} is $O(2^n)$

c) $5n^2 + 3n\log n + 2n + 5$ is $O(n^2)$

d) $3\log n + 2$ is $O(\log n)$

e) $3n\log n – 2n$ is $\Omega(n\log n)$

f) n^2 is $\Omega(n\log n)$

g) $3n\log n + 4n + 5\log n$ is $\Theta(n\log n)$

Justification

d) $3\log n + 2$ is $O(\log n)$

$3\log n + 2 = 3\log n + 2\log 1$

$\leq (3+2)\log n = c\log n$

$\Rightarrow c=5, n_0=2$

e) $3n\log n – 2n$ is $\Omega(n\log n)$

$3n\log n – 2n = n\log n + 2n(\log n - 1)$

$\geq n\log n$

$\Rightarrow c=2, n_0=1$

f) n^2 is $\Omega(n\log n)$

$n^2 = n.n \geq n\log n \Rightarrow c=1, n_0=1$

Exercise

Question 1

a) $(n+1)^5$ is $O(n^5)$

b) 2^{n+1} is $O(2^n)$

c) $5n^2 + 3n\log n + 2n + 5$ is $O(n^2)$

d) $3\log n + 2$ is $O(\log n)$

e) $3n\log n - 2n$ is $\Omega(n\log n)$

f) n^2 is $\Omega(n\log n)$

g) $3n\log n + 4n + 5\log n$ is $\Theta(n\log n)$

Justification

g) $3n\log n + 4n + 5\log n$ is $\Theta(n\log n)$

$3n\log n \le 3n\log n + 4n + 5\log n \le (3+4+5)n\log n$

$3n\log n \le 3n\log n + 4n + 5\log n \le 12n\log n$

$\Rightarrow c_1=3,\ c_2=12,\ n_0=2$

Exercise

Question 2

a) $4n\log n + 2n$	2^{10}	$2^{\log n}$
b) $n^2 + 10$	n^3	$n\log n$
c) $4^{\log n}$	$4n$	$n^{1/\log n}$

AI VIET NAM
@aivietnam.edu.vn

Exercise

Question 2

a) $4n\log n + 2n$ 2^{10} $2^{\log n}$

$\Rightarrow$ $4n\log n + 2n$ is $O(n\log n)$ 2^{10} is $O(1)$ $2^{\log n}=n$ is $O(n)$

$\Rightarrow$ $4n\log n + 2n \leq 2^{\log n} \leq 2^{10}$

b) $n^2 + 10$ n^3 $n\log n$

$\Rightarrow$ $n^2 + 10$ is $O(n^2)$ n^3 is $O(n^3)$ $n\log n$ is $O(n\log n)$

$\Rightarrow$ $n\log n \leq n^2 + 10 \leq n^3$

c) $4^{\log n}$ $4n$ $n^{1/\log n}$

$\Rightarrow$ $4^{\log n}$ is $O(n^2)$ $4n$ is $O(n)$ $n^{1/\log n}$ is $O(1)$

$\Rightarrow$ $n^{1/\log n} \leq 4n \leq 4^{\log n}$

Exercise

Question 3

a)

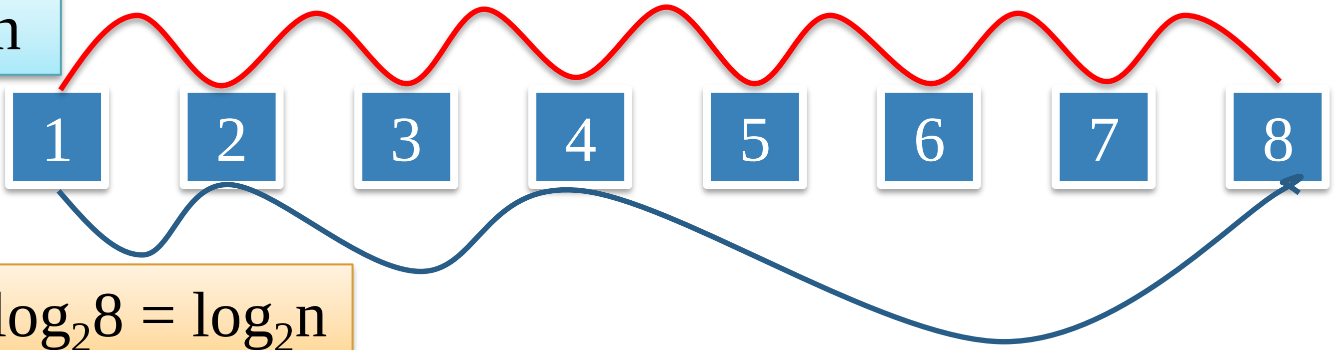
```
def step_example1(n):
    i = 1
    count = 0
    while i < n:
        print(i)
        i *= 2
        count += 1
    return count
```

Exercise

Question 3

8 steps = n

n = 8



3 steps = $\log_2 8 = \log_2 n$

a)

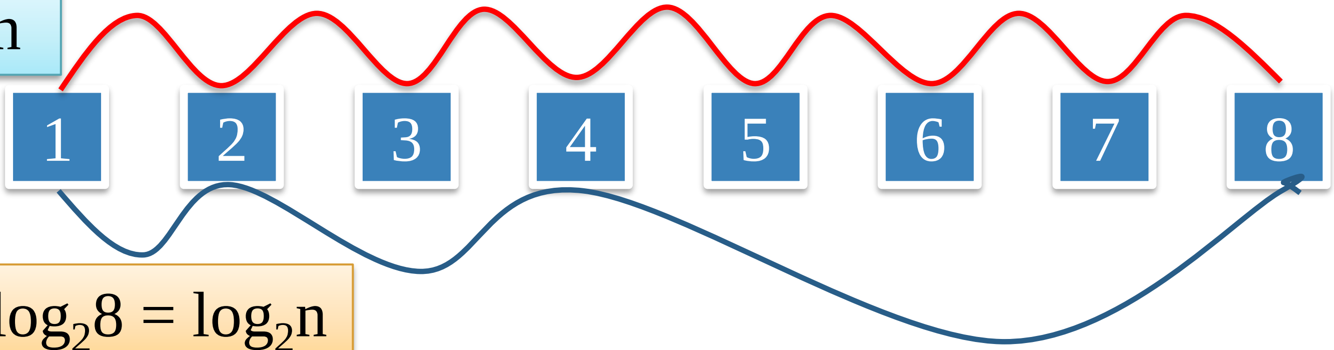
```
def step_example1(n):
    i = 1
    count = 0
    while i < n:
        print(i)
        i *= 2
        count += 1
    return count
```

Exercise

Question 3

8 steps = n

n = 8



3 steps = $\log_2 8 = \log_2 n$

a)

```python
def step_example1(n):
    i = 1
    count = 0
    while i < n:
        print(i)
        i *= 2
        count += 1
    return count
```

cost	times
c_0	1
c_1	1
c_2	$\log_2 n$
c_3	$\log_2 n$
c_4	$\log_2 n$
c_5	$\log_2 n$
c_6	1

$T(n) = (c_2+c_3+c_4+c_5)\log n + (c_0+c_1+c_6)$ is $O(\log n)$

AI VIET NAM
@aivietnam.edu.vn

Exercise

Question 3

a)

```
def step_example1(n):
    i = 1
    count = 0
    while i < n:
        print(i)
        i *= 2
        count += 1
    return count
```

$T(n) = (c_2+c_3+c_4+c_5)\log_2 n + (c_0+c_1+c_6)$

is $O(\log_2 n)$

```
def step_example2(n):
    i = 1
    count = 0
    while i < n:
        print(i)
        i *= 3
        count += 1
    return count
```

$T(n) = (c_2+c_3+c_4+c_5)\log_3 n + (c_0+c_1+c_6)$

is $O(\log_3 n)$

AI VIET NAM
@aivietnam.edu.vn

Exercise

Question 3

b)

```
def sum_example1(S):
    n = len(S)
    total = 0
    for i in range(n):
        total += S[i]
    return total
```

cost	times
c_0	1
c_1	1
c_2	n+1
c_3	n
c_4	1

```
def sum_example2(S):
    n = len(S)
    total = 0
    for i in range(0, n, 2):
        total += S[i]
    return total
```

T(n) is O(n)

Exercise

Question 3

b)

```
def sum_example1(S):
    n = len(S)
    total = 0
    for i in range(n):
        total += S[i]
    return total
```

cost	times
c_0	
c_1	
c_2	
c_3	
c_4	

```
def sum_example2(S):
    n = len(S)
    total = 0
    for i in range(0, n, 2):
        total += S[i]
    return total
```

T(n) is O(n)

AI VIET NAM
@aivietnam.edu.vn

Exercise

Question 3

b)

```
def sum_example1(S):
    n = len(S)
    total = 0
    for i in range(n):
        total += S[i]
    return total
```

cost	times
c_0	1
c_1	1
c_2	n/2+1
c_3	n/2
c_4	1

```
def sum_example2(S):
    n = len(S)
    total = 0
    for i in range(0, n, 2):
        total += S[i]
    return total
```

T(n) is O(n)

T(n) is O(n)

Exercise

Question 3

cost	times
c_0	1
c_1	1
c_2	n+1
c_3	$\sum_{j=1}^{n+1} j + 1$
c_4	$\sum_{j=1}^{n+1} j$
c_5	1

```
def sum_example3(S):
    n = len(S)
    total = 0
    for i in range(n):
        for j in range(1+i):
            total += S[j]
    return total
```

```
def sum_example4(S):
    n = len(S)
    prefix = 0
    total = 0
    for i in range(n):
        prefix += S[i]
        total += prefix
    return total
```

T(n) is O(n^2)

Exercise

Question 3

cost	times
c_0	1
c_1	1
c_2	n+1
c_3	$\sum_{j=1}^{n+1} j + 1$
c_4	$\sum_{j=1}^{n+1} j$
c_5	1

```
def sum_example3(S):
    n = len(S)
    total = 0
    for i in range(n)
        for j in rang
            total +=
    return total
```

cost	times
c_0	1
c_1	1
c_2	1
c_3	n+1
c_4	n
c_5	n
c_6	1

```
def sum_example4(S):
    n = len(S)
    prefix = 0
    total = 0
    for i in range(n):
        prefix += S[i]
        total += prefix
    return total
```

T(n) is O(n^2)

T(n) is O(n)

Exercise

Question 3

c)

```
def uniq_example1(S):
    n = len(S)
    for i in range(n):
        for j in range(i+1, n):
            if S[i] == S[j]:
                return False
    return True
```

```
def uniq_example2(S):
    n = len(S)
    S_temp = sorted(S)
    for i in range(n-1):
        if S_temp[i] == S_temp[i+1]:
            return False
    return True
```

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

T(n) is O(nlogn)

Exercise

Question 4: Running time analysis: worst, best, average case

```python
def insertion_sort(S):
    n = len(S)
    for step in range(1, n):
        key = S[step]
        i = step - 1
        while i >= 0 and key < S[i]:
            S[i + 1] = S[i]
            i = i - 1
        S[i + 1] = key
    return S
```

Exercise

Question 4: Running time analysis: worst, best, average case

Example

23	1	10	5	2

```
def insertion_sort(S):
    n = len(S)
    for step in range(1, n):
        key = S[step]
        i = step - 1
        while i >= 0 and key < S[i]:
            S[i + 1] = S[i]
            i = i - 1
        S[i + 1] = key
    return S
```

step = 1
key = 1

23	1	10	5	2

Exercise

Question 4: Running time analysis: worst, best, average case

Example

23	1	10	5	2

```python
def insertion_sort(S):
    n = len(S)
    for step in range(1, n):
        key = S[step]
        i = step - 1
        while i >= 0 and key < S[i]:
            S[i + 1] = S[i]
            i = i - 1
        S[i + 1] = key
    return S
```

step = 1
key = 1

23	1	10	5	2

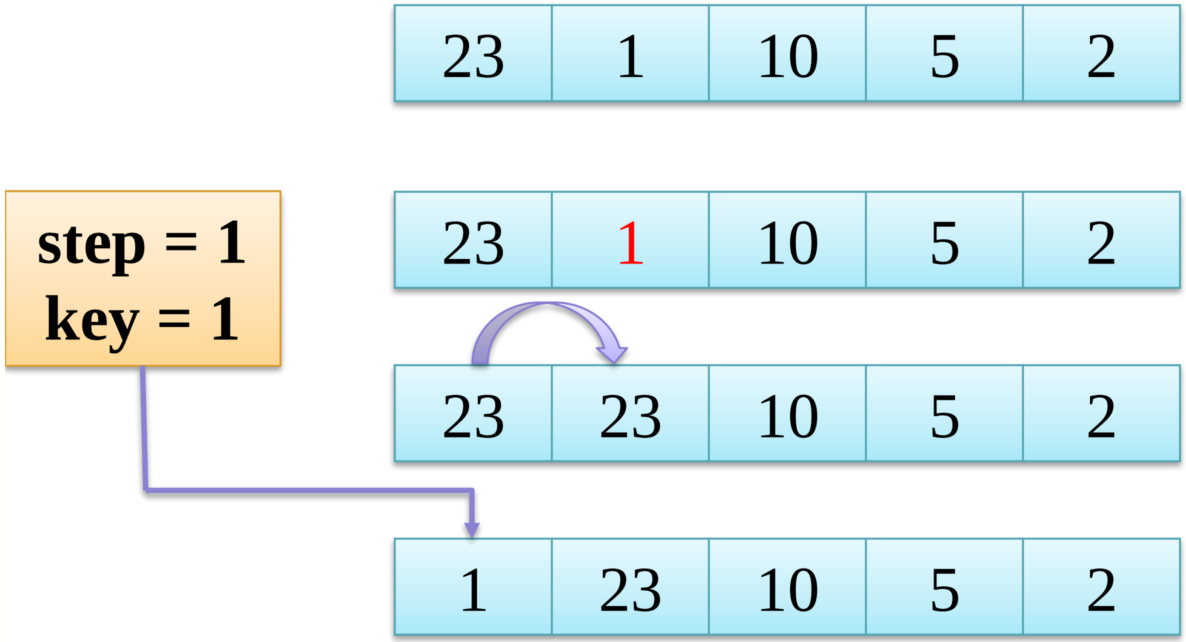
23	23	10	5	2

Exercise

Question 4: Running time analysis: worst, best, average case

Example

```python
def insertion_sort(S):
    n = len(S)
    for step in range(1, n):
        key = S[step]
        i = step - 1
        while i >= 0 and key < S[i]:
            S[i + 1] = S[i]
            i = i - 1
        S[i + 1] = key
    return S
```

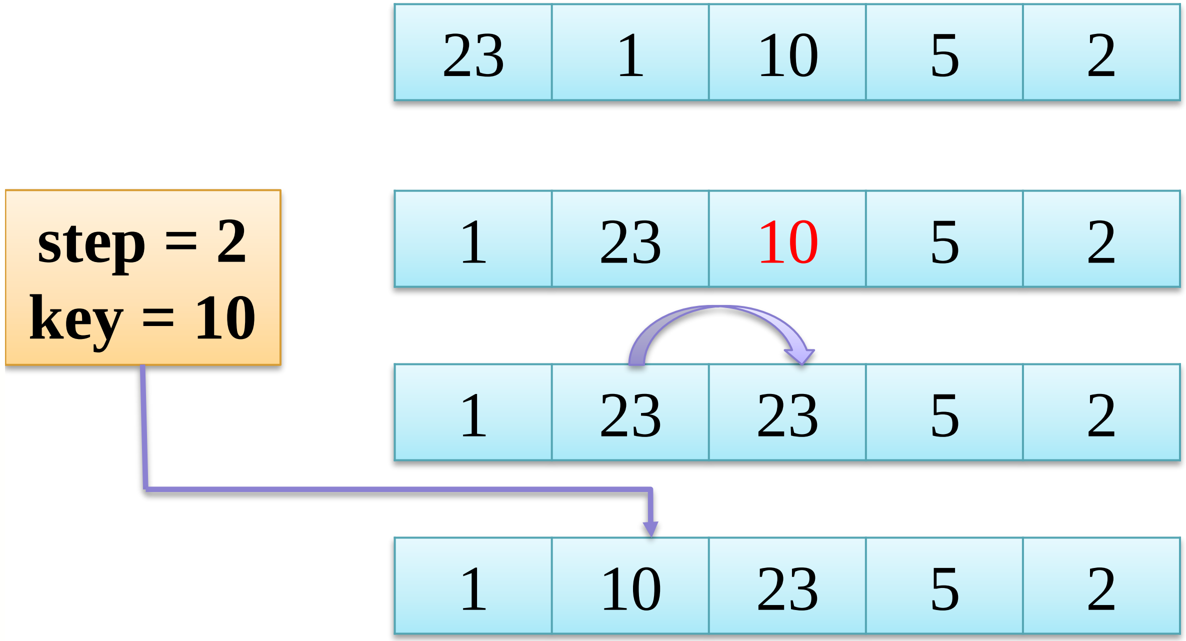


Exercise

Question 4: Running time analysis: worst, best, average case

Example

```python
def insertion_sort(S):
    n = len(S)
    for step in range(1, n):
        key = S[step]
        i = step - 1
        while i >= 0 and key < S[i]:
            S[i + 1] = S[i]
            i = i - 1
        S[i + 1] = key
    return S
```

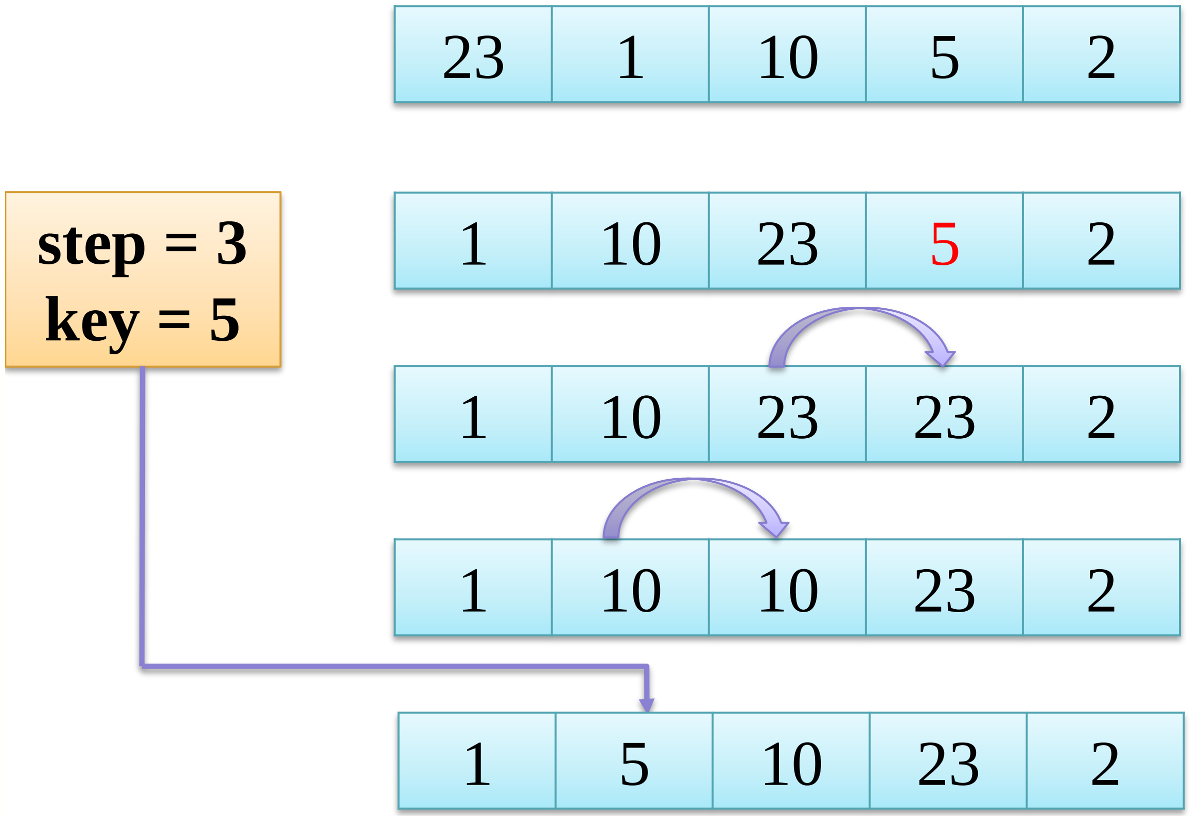


Exercise

Question 4: Running time analysis: worst, best, average case

Example

```python
def insertion_sort(S):
    n = len(S)
    for step in range(1, n):
        key = S[step]
        i = step - 1
        while i >= 0 and key < S[i]:
            S[i + 1] = S[i]
            i = i - 1
        S[i + 1] = key
    return S
```

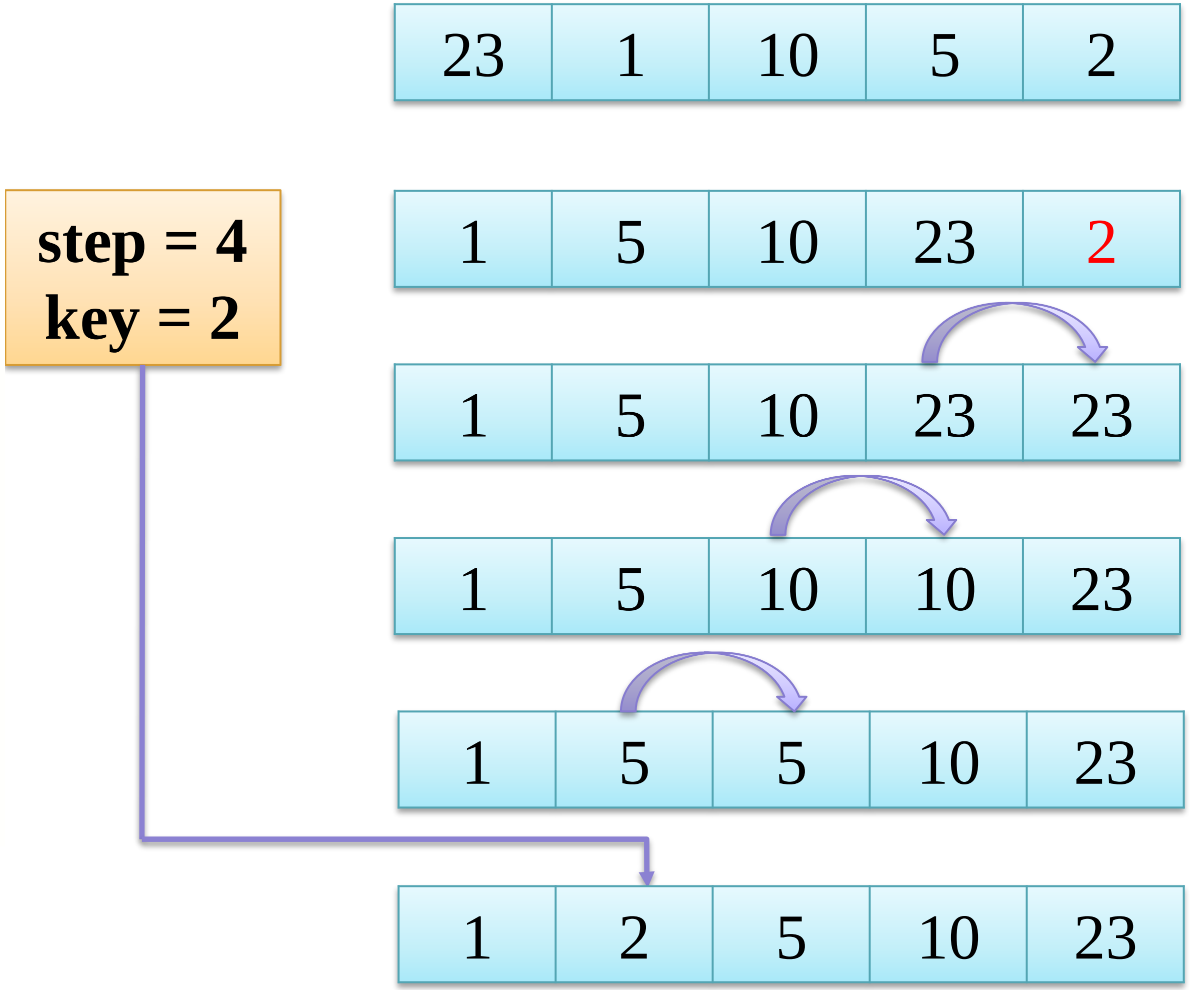


Exercise

Question 4: Running time analysis: worst, best, average case

Example

```python
def insertion_sort(S):
    n = len(S)
    for step in range(1, n):
        key = S[step]
        i = step - 1
        while i >= 0 and key < S[i]:
            S[i + 1] = S[i]
            i = i - 1
        S[i + 1] = key
    return S
```



AI VIET NAM
@aivietnam.edu.vn

Exercise

Question 4

1. def insertion_sort(s):	cost	times
2. n = len(s)	c_0	1
3. for step in range(1, n):	c_1	n
4. key = s[step]	c_2	n-1
5. i = step – 1	c_3	n-1
6. while i >= 0 and key < s[i]	c_4	$\sum_{i=1}^{n-1} t_i$
7. s[i+1] = s[i]	c_5	$\sum_{i=1}^{n-1}(t_i-1)$
8. i = i – 1	c_6	$\sum_{i=1}^{n-1}(t_i-1)$
9. s[i+1] = key	c_7	n-1
10. return s	c_8	1

t_i is the number of times while loop test in line 6 is executed for that value of i

$$T(n) = c_0 + c_1 n + c_2(n-1) + c_3(n-1) + c_4\sum_{i=1}^{n-1} t_i + c_5\sum_{i=1}^{n-1}(t_i-1) + c_6\sum_{i=1}^{n-1}(t_i-1) + c_7(n-1) + c_8$$

Exercise

Question 4

$T(n) = c_0 + c_1 n + c_2(n\text{-}1) + c_3(n\text{-}1) + c_4\sum_{i=1}^{n-1} t_i + c_5\sum_{i=1}^{n-1}(t_i\text{-}1) + c_6\sum_{i=1}^{n-1}(t_i\text{-}1) + c_7(n\text{-}1) + c_8$

- Best case: already ordered numbers

  - $t_i=1$, line 7 and 8 will be executed 0 times

  - $T(n) = c_0 + c_1 n + c_2(n\text{-}1) + c_3(n\text{-}1) + c_4(n\text{-}1) + c_7(n\text{-}1) + c_8$

    $= (c_1 + c_2 + c_3 + c_4 + c_7)n + (c_0 + c_8 - c_2 - c_3 - c_4 - c_7)$

    $= cn + c'$

    $\Rightarrow O(n)$

AI VIET NAM
@aivietnam.edu.vn

Exercise

Question 4

$T(n) = c_0 + c_1 n + c_2(n-1) + c_3(n-1) + c_4\sum_{i=1}^{n-1} t_i + c_5\sum_{i=1}^{n-1}(t_i-1) + c_6\sum_{i=1}^{n-1}(t_i-1) + c_7(n-1) + c_8$

- Worst case: reverse numbers

  - t_i=i, line 7 and 8 will be executed i times

  - $\sum_{i=1}^{n-1} t_i = \sum_{i=1}^{n-1} i =$ n(n+1)/2-1, and $\sum_{i=1}^{n-1}(t_i-1) = \sum_{i=1}^{n-1}(i-1) = n(n-1)/2$

  - $T(n) = c_0 + c_1 n + c_2(n-1) + c_3(n-1) + c_4 (n(n+1)/2-1) + c_5 n(n-1)/2$

    $+ c_6 n(n-1)/2 + c_7(n-1) + c_8$

    $= an^2 + bn + c$

    $\Rightarrow O(n^2)$

- Average case: random numbers

  - $t_i = i/2 \Rightarrow$ The same worst case: $O(n^2)$

AI VIET NAM
@aivietnam.edu.vn

Exercise

Question 5

a) $\sum_{i=1}^{n} \log i$ is $O(n\log n)$

b) $\sum_{i=1}^{n} \log i$ is $\Theta(n\log n)$

c) Let $p(n) = \sum_{i=1}^{d} a_i n^i$, where $a_d > 0$. Let k be a constant. Use the definitions of the asymptotic notation to prove the following properties:

(i) Nếu $k \geq d$, thì $p(n) = O(n^k)$

(ii) Nếu $k = d$, thì $p(n) = \Theta(n^k)$

Exercise

Question 5

a) $\sum_{i=1}^{n} \log i = \log 1 + \log 2 + \ ... + \log n = \log(1*2*\ldots*n)$

$\leq \log(n*n*\ldots*n) = n\log n$

$=> \sum_{i=1}^{n} \log i$ is $O(n\log n)$

b) $\sum_{i=1}^{n} \log i \geq \sum_{i=n/2}^{n} \log i = \log\left(\frac{n}{2}\right) + \log\left(\frac{n}{2} + 1\right) + \ ... + \log(n)$

$= \log n - \log 2 + \log(n+3) - \log 2 + \ldots + \log n$

$\geq \log n - \log 2 + \log n - \log 2 + \ldots + \log n$

$= n/2\log n - (n/2-1)\log 2$

$=> \sum_{i=1}^{n} \log i$ is $\Omega(n\log n)$

$=> \sum_{i=1}^{n} \log i$ is $\Theta(n\log n)$

Exercise

Question 5

c) $p(n) = \sum_{i=0}^{d} a_i n^i$, $a_d > 0$

(i) – $p(n)$ is $O(n^k)$, if $k \geq d$

$\sum_{i=0}^{d} a_i n^i \leq cn^k$

divide n^k => $\sum_{i=0}^{d} a_i n^{i-k} \leq c$

$k >= d$ => $i\text{-}k \leq 0$ => $n^{i-k} \leq 1$

=> Choose $c = \sum_{i=0}^{d} a_i$

c) $p(n) = \sum_{i=0}^{d} a_i n^i$, $a_d > 0$

(ii) – $p(n)$ is $\Theta(n^k)$, if $k = d$

$c_1 n^k \leq p(n) \leq c_2 n^k$

Note (i) => $c_2 = \sum_{i=0}^{d} a_i$

Find c_1

$c_1 n^k \leq \sum_{i=0}^{d} a_i n^i$

divide n^k => $c_1 \leq \sum_{i=0}^{d} a_i n^{i-k}$

$k = d$ => $i\text{-}k \leq 0$ => $n^{i-k} \leq 1$

=> Choose $c_1 = a_d$

=> $c_1 = a_d$, $c_2 = \sum_{i=0}^{d} a_i$

SUMMARY

Python code

```
[6]  S = [1, 2, 3]
     n = len(S)
     for i in range(n):
         for j in range(n):
             total = S[i] + S[j]
             print(total)
         print('- - - -')
```

Characterize Function

1(c)
logn
n
nlogn
$\mathbf{n^2}$
$\mathbf{n^3}$
$\mathbf{a^n}$

Asymptotic Notation

$\mathbf{O(n)}$
$\mathbf{\Omega(n)}$
$\mathbf{\Theta(n)}$

AI VIET NAM
@aivietnam.edu.vn

Reference

(1) Introduction to Algorithms, 3rd Edition; Thomas H.Cormen et al; 2009

(2) Data Structures & Algorithms; Michael T.Goodrich et al; 2013

(3) Algorithms, 4th; Robert Sedgewick et al; 2011

Thanks!

Any questions?