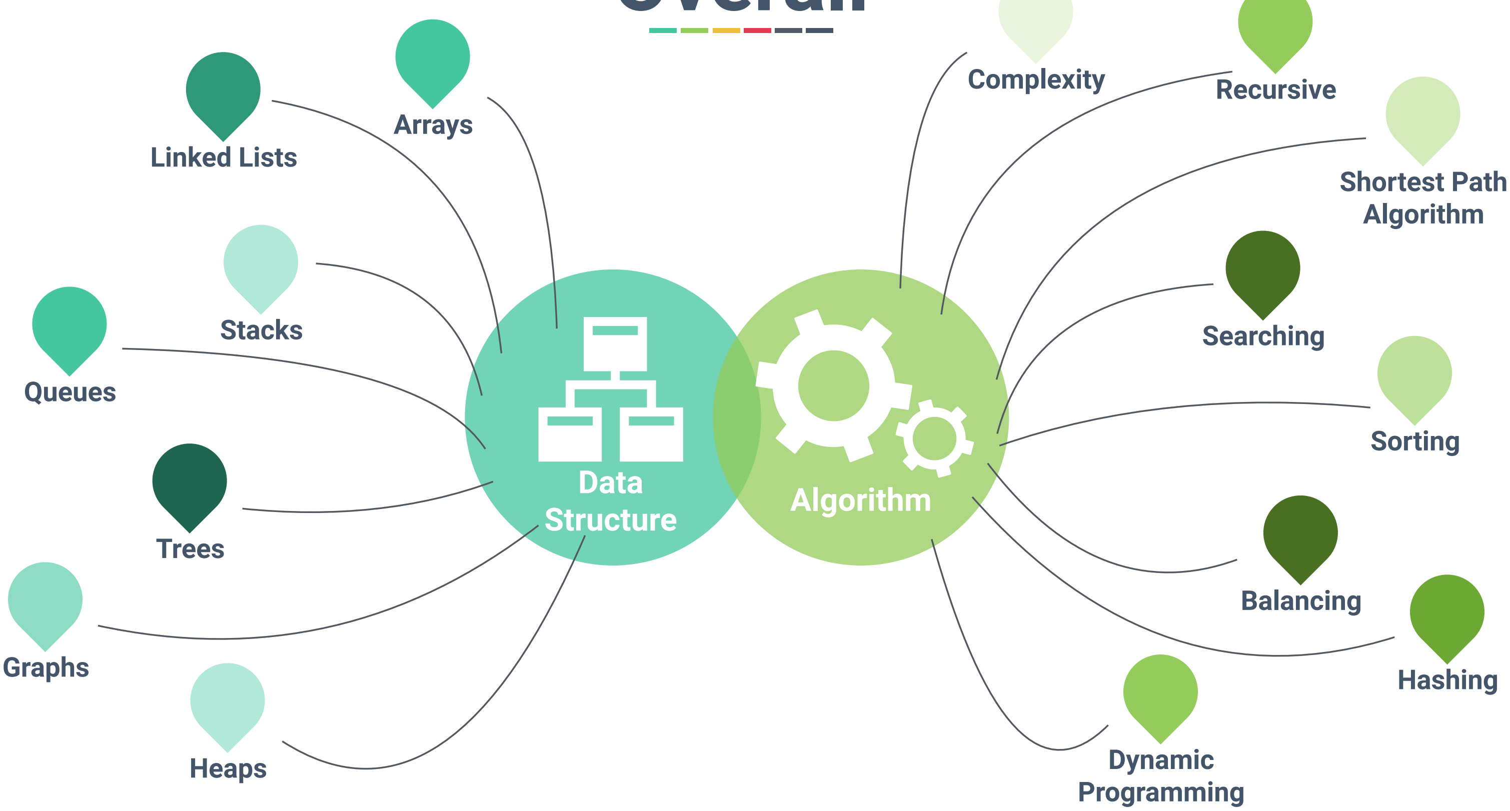
Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Introduction to Data Structures and Algorithms

Dr. Sirasit Lochanachit

Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Overall



Overall

- Data Structures
- Algorithms

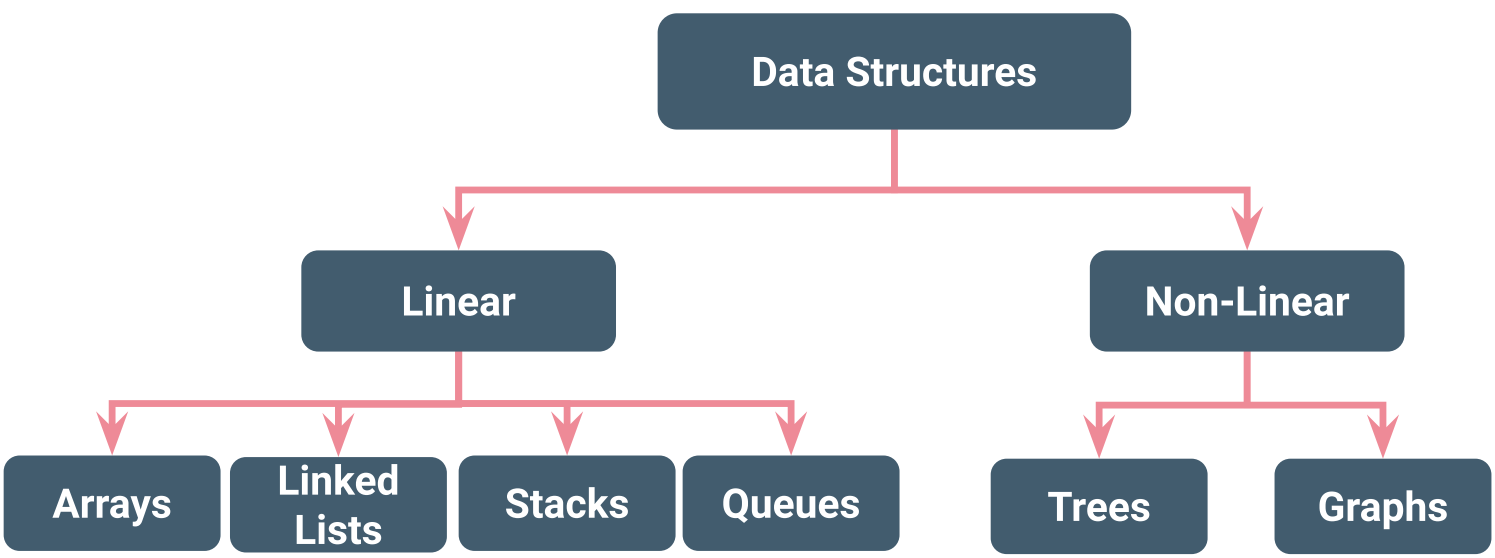
What is “Data Structure” ?

- Way to store and organise data
- Enable efficient access and modification of data
- Designed for a specific algorithm
  - Strengths and limitations
  - Time and space complexity

Abstract Data Type

- A data type, where only behavior is defined but not implementation.
- Similar to a black box where input and output are known, but not how.
- Examples: Array, List, Map, Queue, Set, and etc.

Type of Data Structure



Check out for a comprehensive list of data structures at
https://en.wikipedia.org/wiki/List_of_data_structures

What is “Algorithm” ?

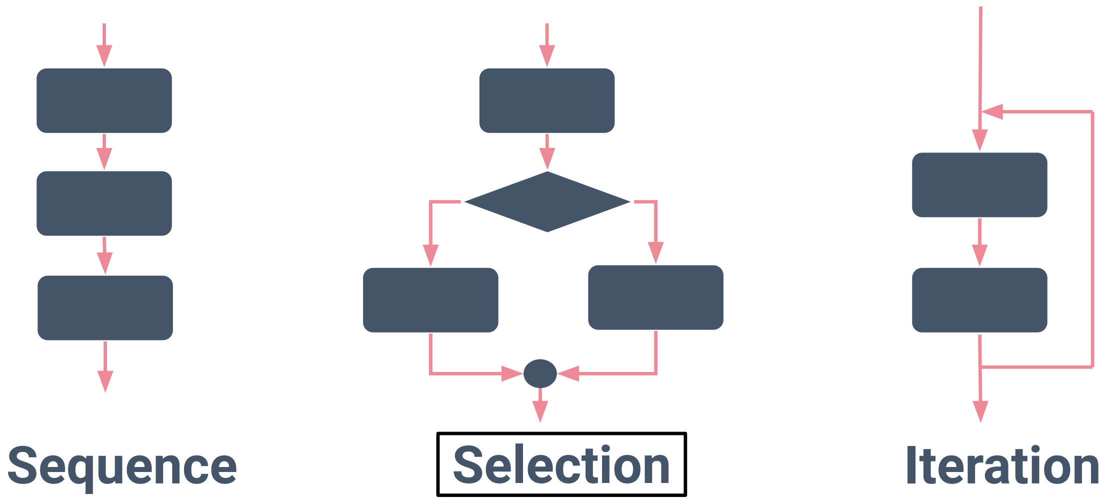
- Well-defined procedure or set of instructions to accomplish a task
- Sequence of steps that transform input into output
- Tool to solve a well-specified computational problem

Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

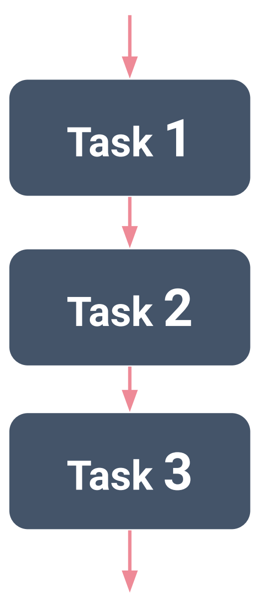
Example: Sorting numbers

1. Input:
    - A sequence of n numbers: *<a1, a2, …, an>*
    - <31, 41, 59, 26, 41>
2. Sorting Algorithms
3. Output:
    - A permutation (reordering) *<a’1, a’2, …, a’n>* of input sequence such that *a’1 <= a’2 <= … <= a’n*
    - <26, 31, 41, 41, 59>

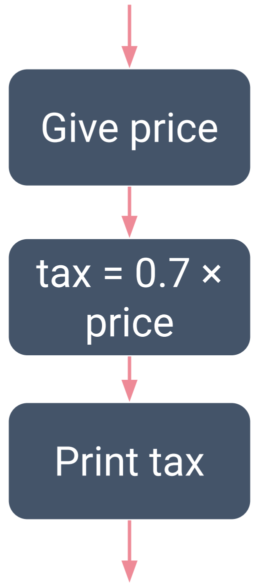
Control Structure



Sequence



Sequence

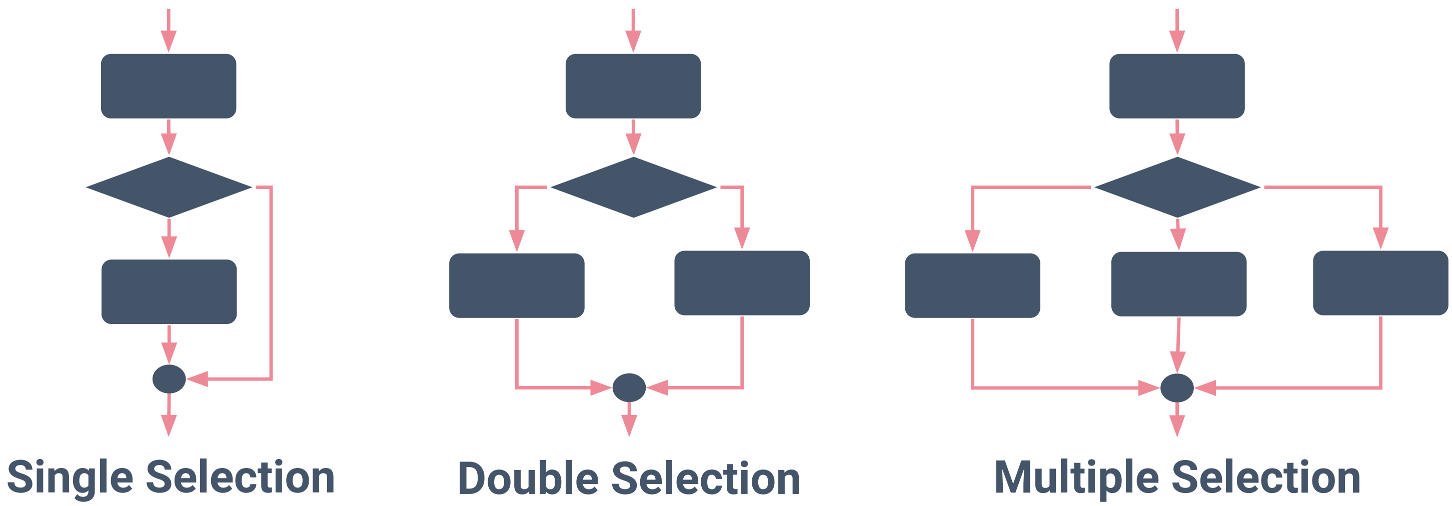


Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Exercise: Sequence

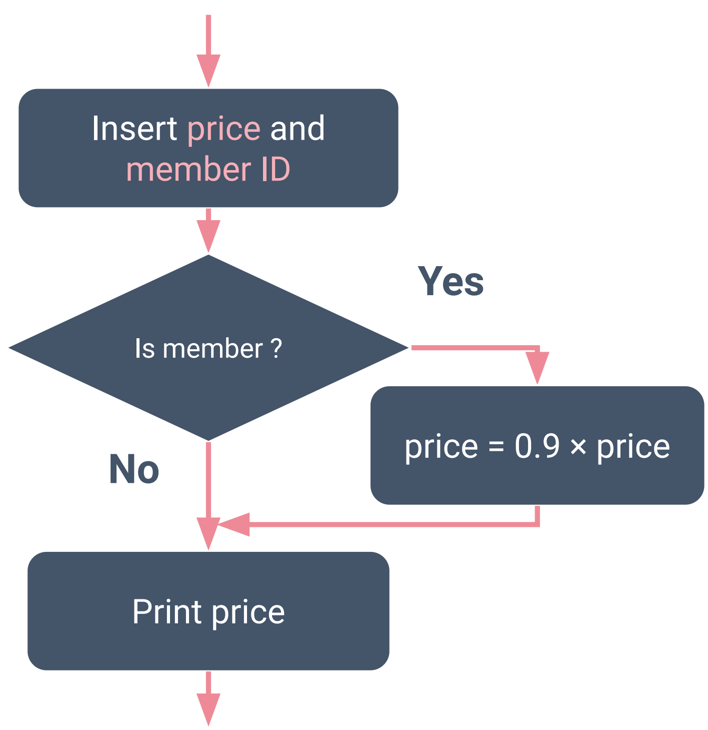
Write a flowchart that converts USD currency to Thai baht. At the end of the flowchart, print the Thai baht. (Exchange rate USD to Thai baht is 32.00)

Selection

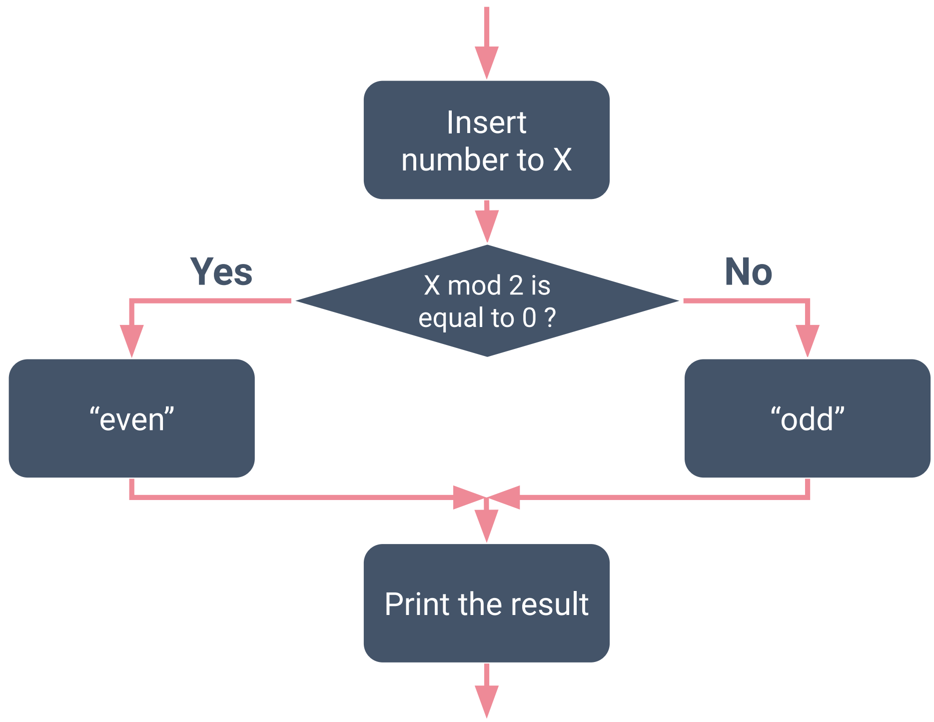


Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Selection

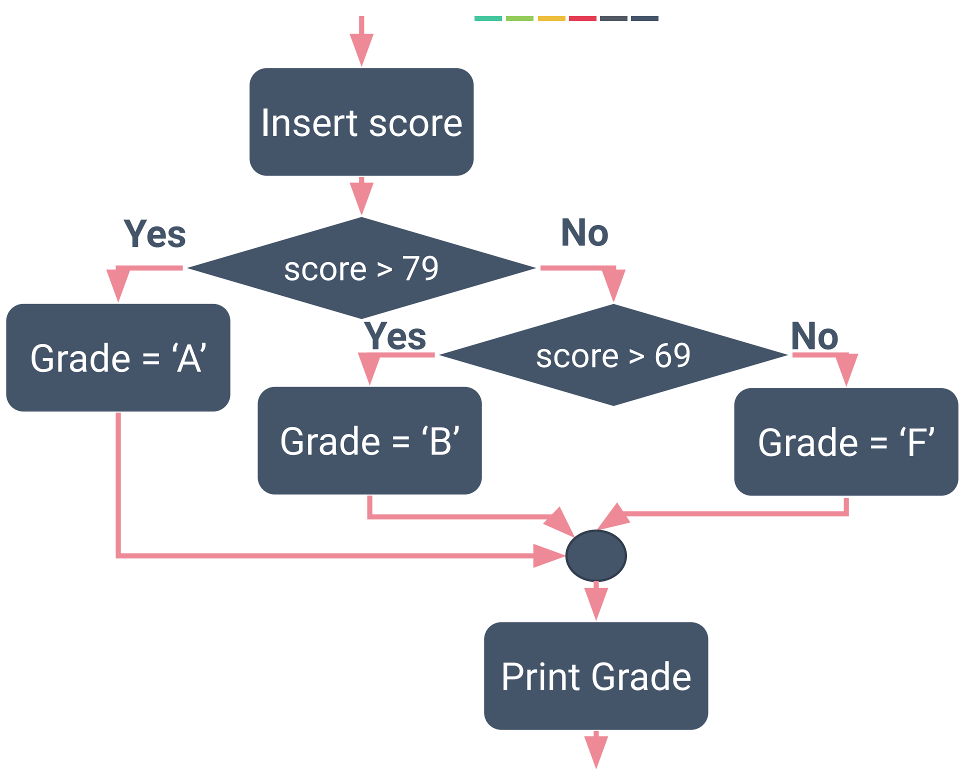


Selection



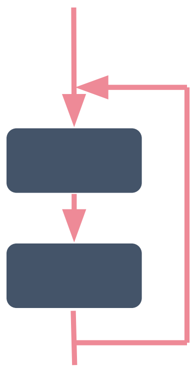
Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Selection

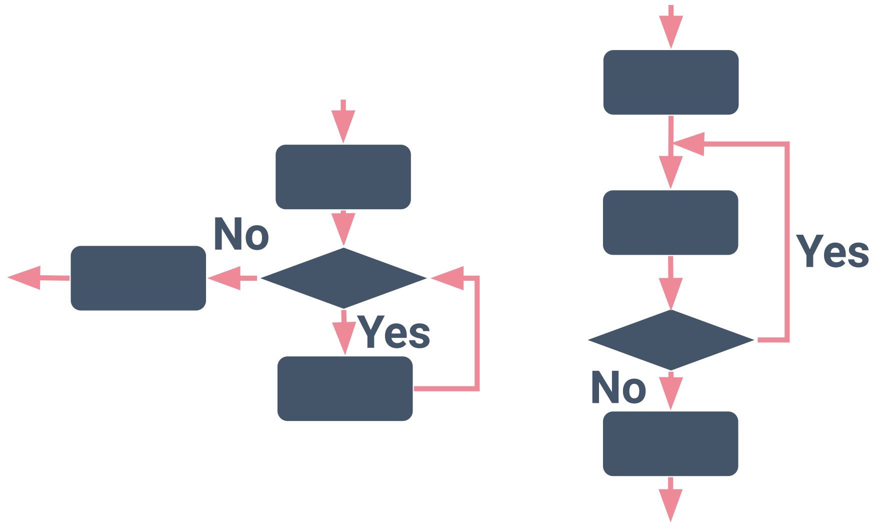


Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Iteration



Infinite Loop



Finite Loop

Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Exercise: Control Flow

Write a flowchart describing the logic of factorial function.

Hint: The factorial n!, is the product of all positive integers less than or equal to n.

For example: 5! = 5 * 4 * 3 * 2 * 1 = 120

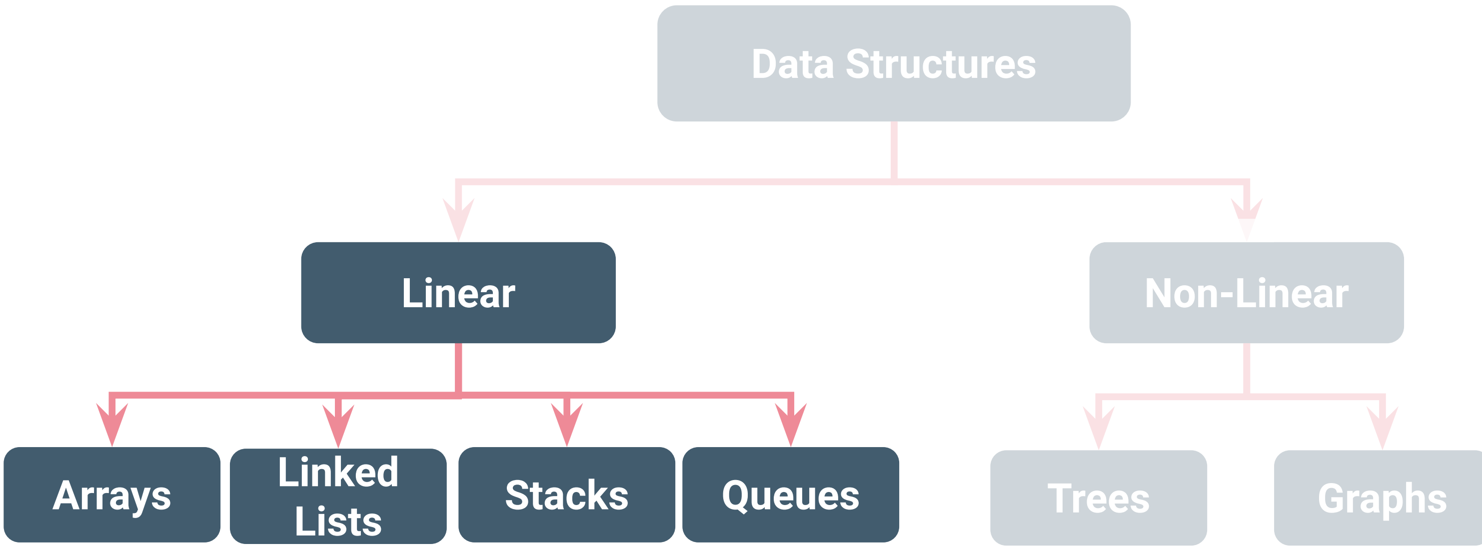
What kind of problems are solved by algorithms?

- Human Genome Project
  - identifying all genes of human beings
- Internet: Routing, searches, and security
  - Shortest path, search engines, encrypted communication
- E-commerce
  - Ads, recommendations, authentications
- Commercial enterprises
  - Resource allocation:
    - crew assignment on flights, package delivery route

Data Structures and Algorithms

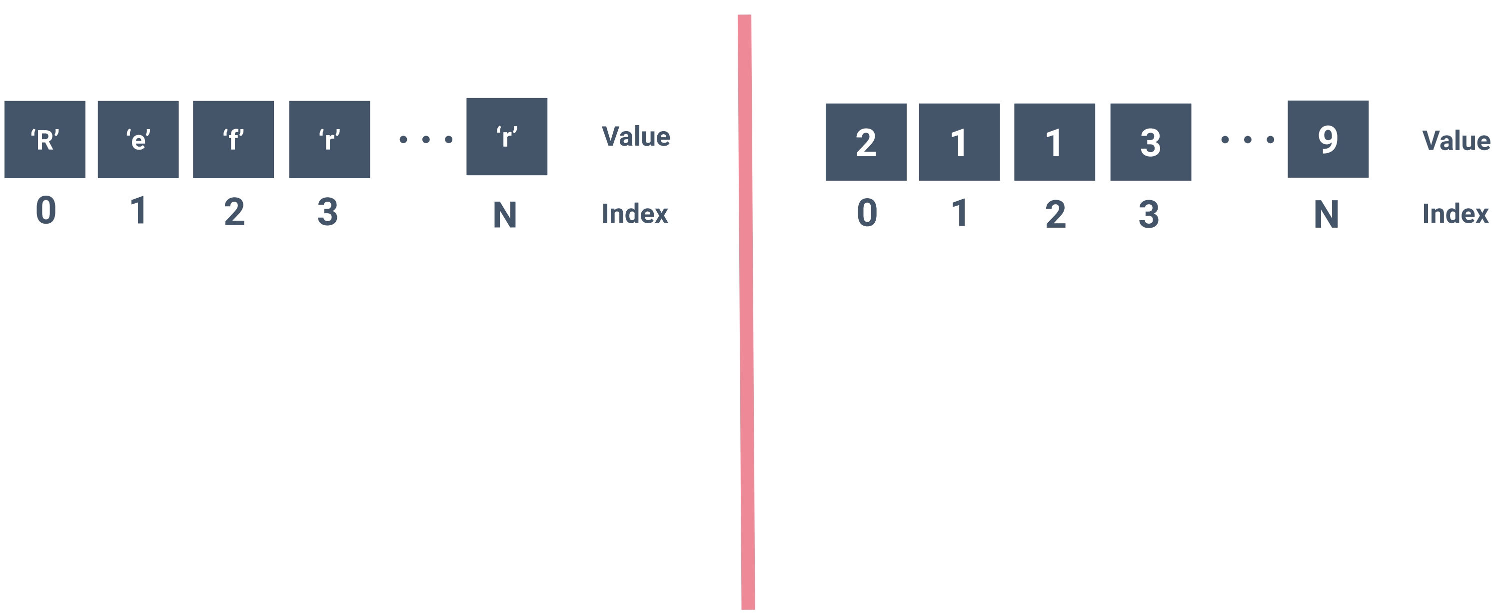
- Data Structures
  - Way to store and organise data
- Algorithms
  - Sequence of steps performed on data structures to perform a task

Type of Data Structure

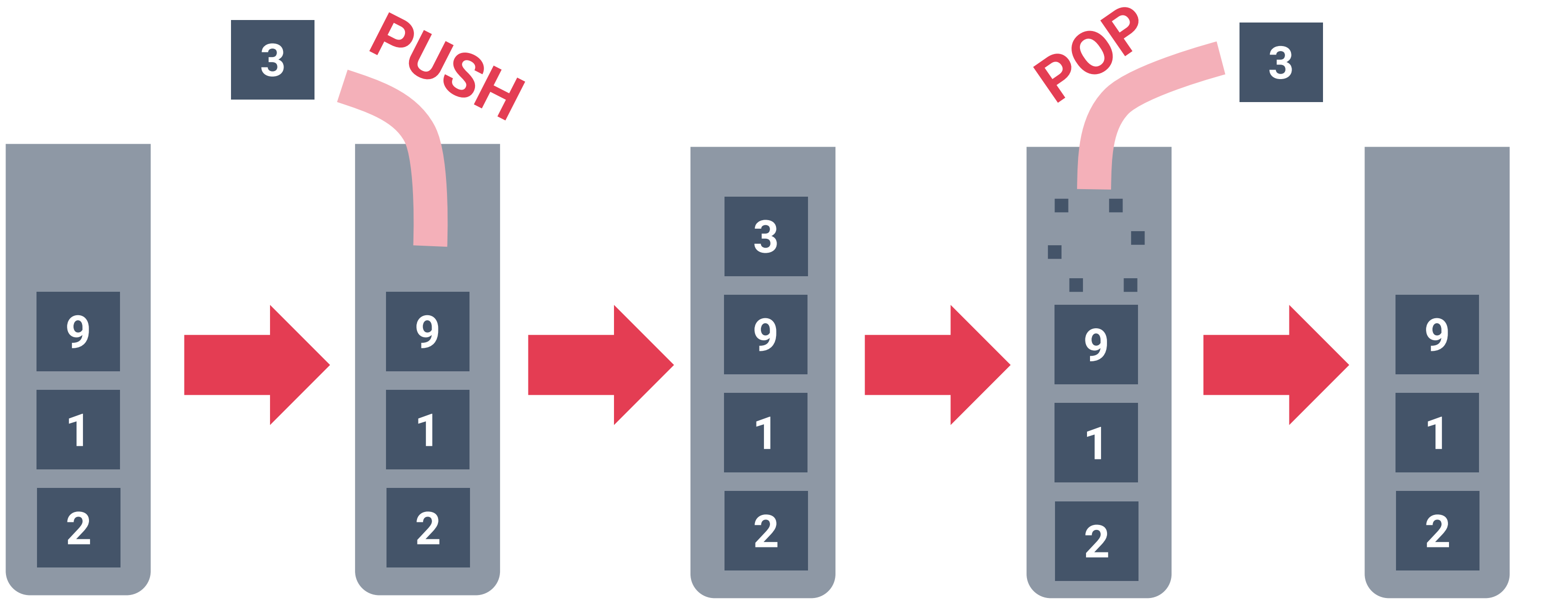


Check out for a comprehensive list of data structures at https://en.wikipedia.org/wiki/List_of_data_structures

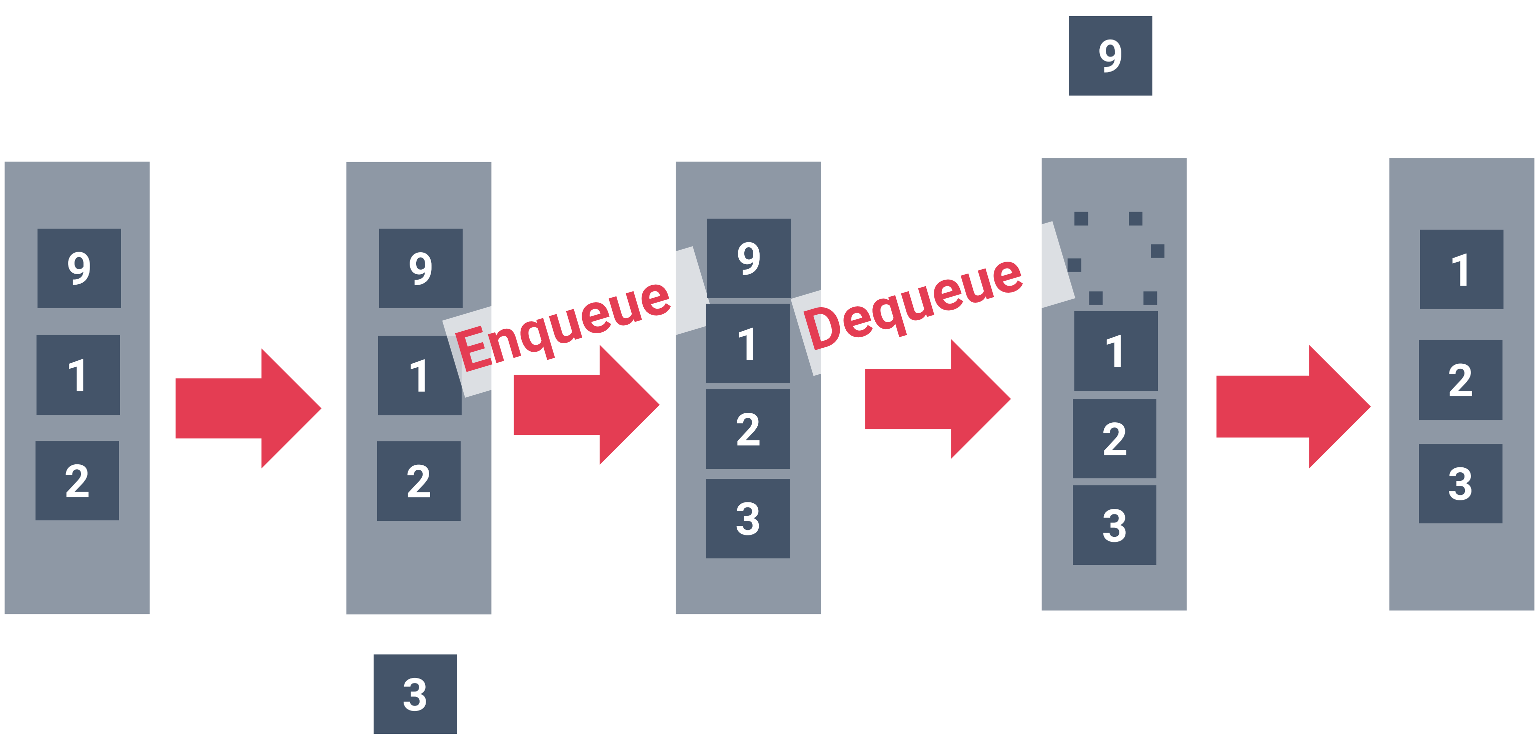
Arrays



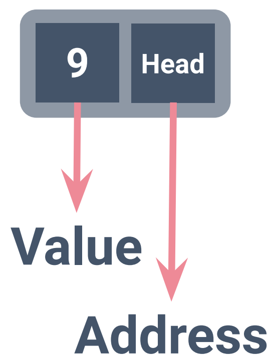
Stacks



Queues

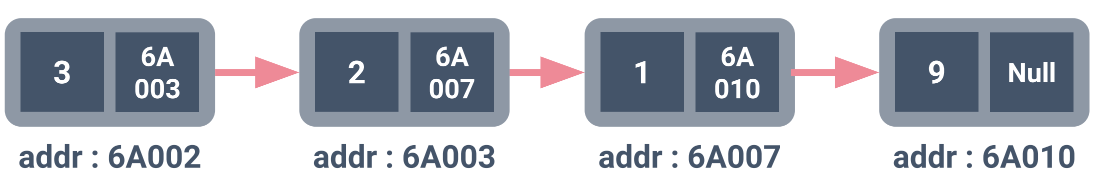


Linked Lists



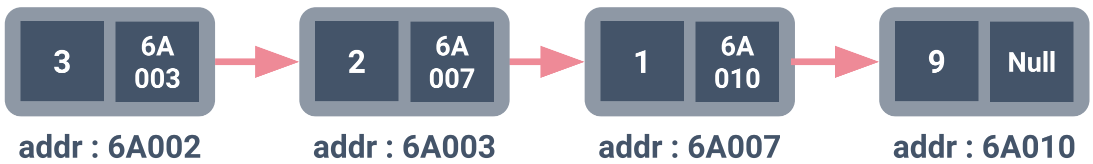
Linked Lists

- **Linked Lists**



Linked Lists VS Arrays

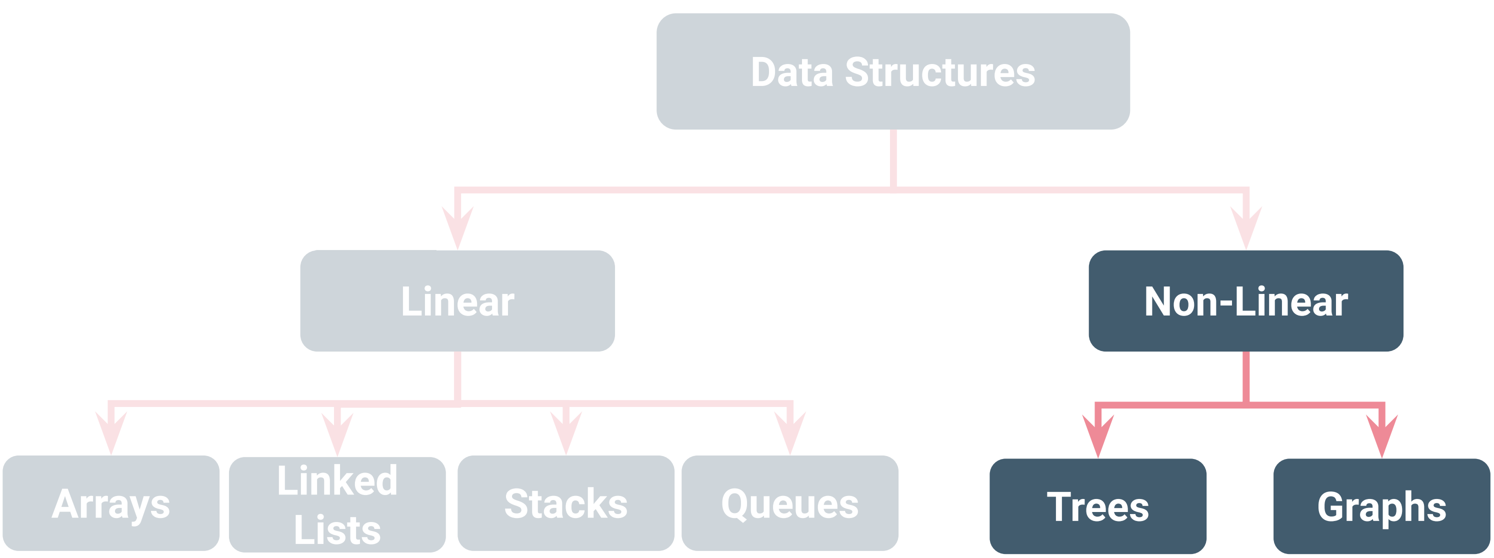
- **Linked Lists**



- **Arrays**

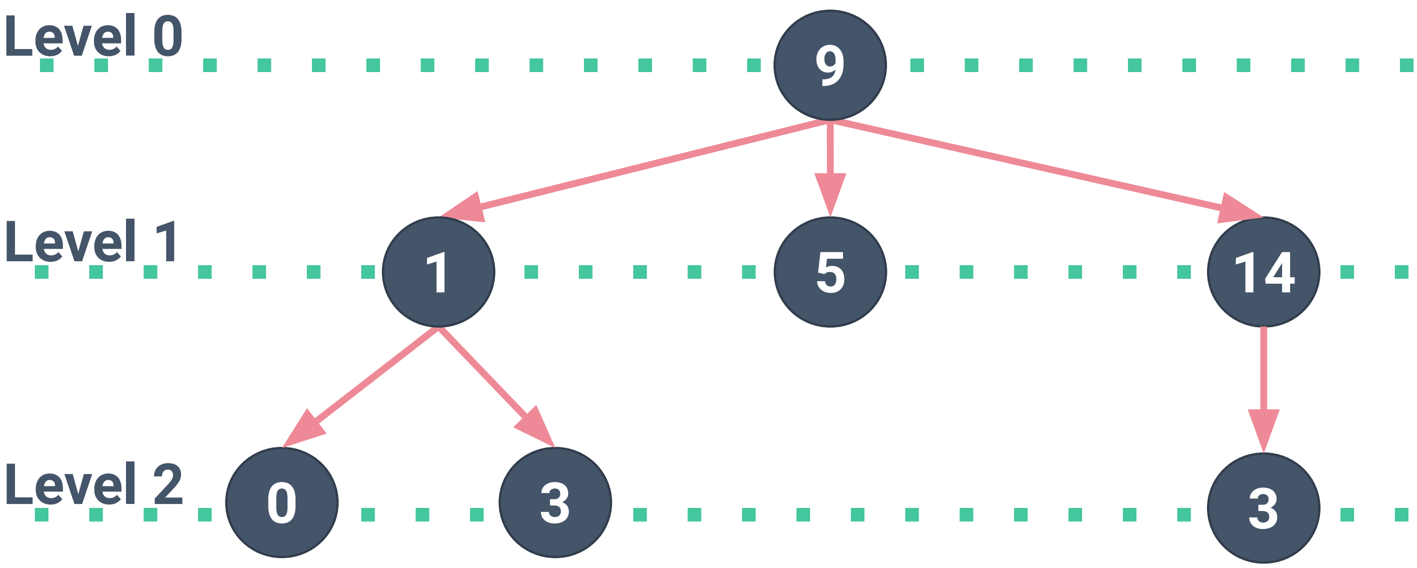
Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Type of Data Structure

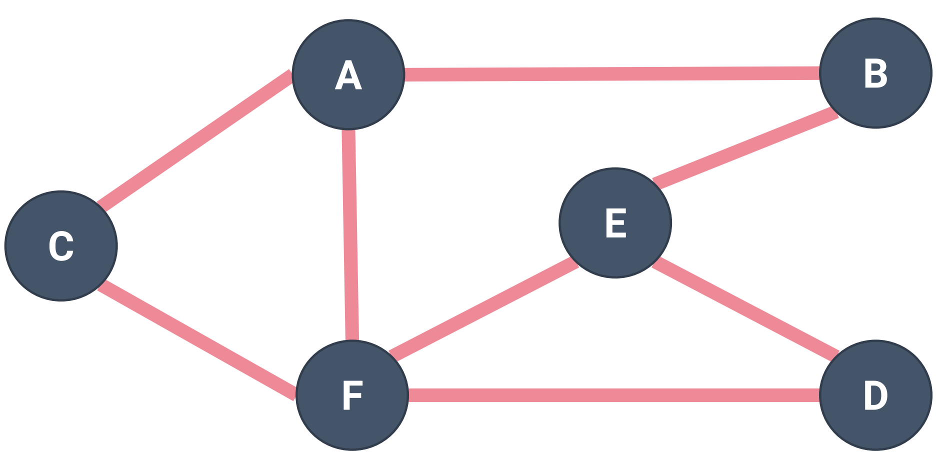


Check out for a comprehensive list of data structures at
https://en.wikipedia.org/wiki/List_of_data_structures

Trees



Graphs



Algorithm Analysis

- If computers were infinitely fast, **any correct method** for solving a problem would do.
- Computing time and space in memory are a limited resource.
- The solution should run as fast as possible.
- For the same problem, it can be solved by different algorithms but they are often differ in their efficiency.
- **Algorithms that are efficient in terms of time or space are preferred.**

Algorithm Analysis

- How do we measure algorithm **efficiency** or **performance**?
  - Use running time as an indicator.

Benchmark Analysis

- Example: Summation of *n* integers
  - Time required for the iterative solution seems to increase as we increase the value of *n*.
- Running time depends on many factors
  - Hardware (CPU, RAM, etc.)
  - Software (OS, Programming language, etc.)
- Need an alternative way for analysing algorithms with respect to running time

Algorithm Analysis

- How do we measure algorithm **efficiency** or **performance**?
  - Use running time as an indicator.

- Running time can be expressed as the number of operations or steps executed.
  - theSum = 0 -> 1 step/time
  - for i in range(1,n+1):
    theSum = theSum + i -> n steps/times

Time complexity
T(n) = 1+n

Algorithm Analysis

- The parameter ***n*** is often referred as "**size of the problem**" or **input size**
- The running time of an algorithm or data structure operation **increases** with the input size.
- Running time as a function of the input size - f(n)
- As *n* gets larger, the constant will become less significant.
- Order of magnitude/growth describes the running time that is most important.

Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Asymptotic Notation

- Example:
  - Suppose an algorithm runs on an input size n.
  - Times required to execute is $T(n) = 2n^2 + n + 1$
  - The $\mathbf{n^2}$ terms become larger when **n** gets larger.
  - The running time of this algorithm grows as $\mathbf{n^2}$.
- Asymptotic notation represents algorithm's complexity
  - Ignores constant factors and slower growing terms.
  - Focus on the main components that affect the growth.
  - Big-O notation

Big-O notation

- **Big-*O* notation (Order)**
  - Example:
    - Given T(n) = 1+n, then T(n) = O(n)
    - Given T(n) = $2n^2$ + n + 1, then T(n) = O(n^2)

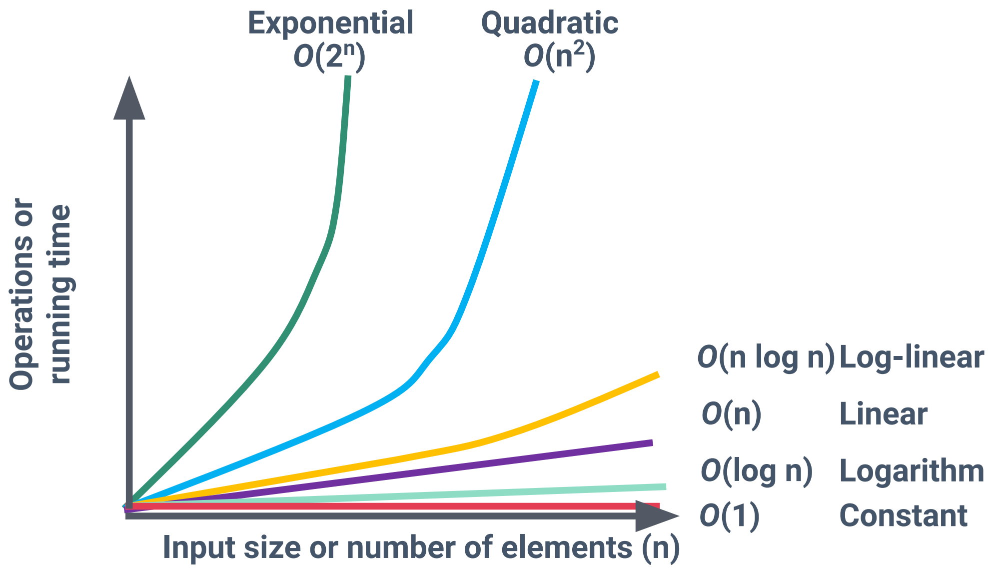
- **O(n) means time complexity will never exceed n.**

Big-O notation

- **Big-*O* notation (Order)**
  - Common f(n):
    - O(1): constant
    - O(log n): logarithm
    - O(n): Linear
    - O(n log n): Log linear
    - O(n^2): Quadratic
    - O(2^n): Exponential
    - O(n!): Factorial

Time Complexity

In general, the standard functions of input size n are shown in figure.

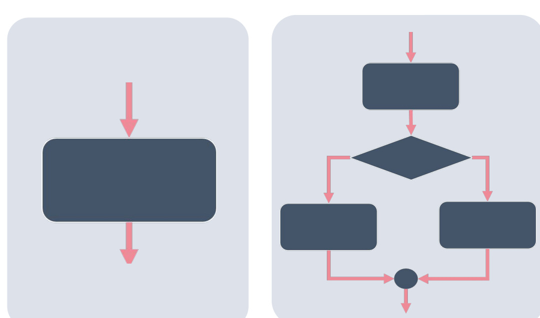


Calculating Time Complexity

Mathematic

$f(n) = 3$

Computer



- Simple statements (read, write, assign)
- Simple operations (+ - * / == > >= < <=)

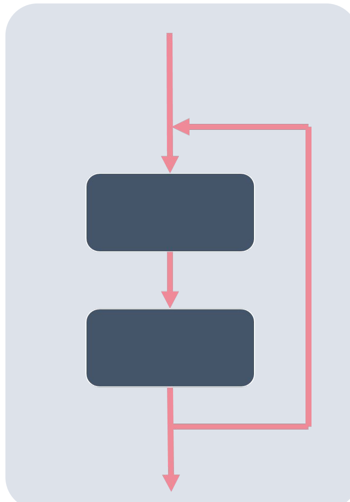
Big-O

$O(1)$

Constant

Calculating Time Complexity

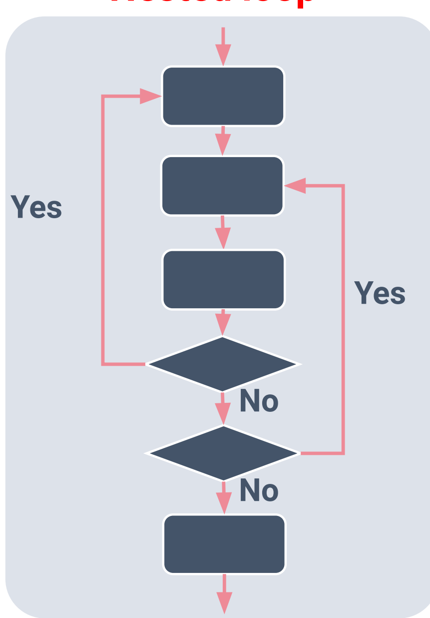
Mathematic	Computer	Big-O
$f(n) = n-3$	Linear loop	$O(n)$ → Linear



Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Calculating Time Complexity

Mathematic	Computer	Big-O
$f(n) = n^2 + n + 7$	Nested loop	$O(n^2)$ → Quadratic



Loops

	Python Code	Time complexity
Linear	for i in range (0,n): do something	T(n) = n, O(n)
Linear	for i in range (0,n,2): do something	T(n) = n/2, O(n)

Nested	for i in range (0,n):
	for j in range (0,n):
	do something
		T(n) = n^2, O(n^2)

Loops

	Python Code	n=1000	Time complexity
Logarithmic	i = 1 while i < n: do something i *= 2	1 2 4 8 16 32 64 128 256 512	T(n) = log n, O(log n)
Logarithmic	i = n while i >= 1: do something i //= 2	1000 500 250 125 62 31 15 7 3 1	T(n) = log n, O(log n)

Loops

Python Code

Time complexity

Linear logarithmic

```
for i in range (0,n):
        j = 1
        while j < n:
                do something
                j *= 2
```

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

Loops

Python Code

Time complexity

Dependent Nested

```
for i in range (0,n):
        for j in range(0, i+1)
                do something
```

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

Number of iterations of the inner loop depends on the outer loop

For the inner loop, the number of iterations is (n+1)/2

Exercise

Time complexity

- $n^2 + 100n$
- $n^2 * n + 100n^2 \log n$
- $123 + \log 657$
- $(n + \log n)^3$
- $n(2 + \log n)$
- $1 + 2 + 3 + ... + n$

O(?)
O(?)
O(?)
O(?)
O(?)
O(?)

Exercise

Time complexity

- $n^2 + 100n$
- $n^2 * n + 100n^2 \log n$
- $123 + \log 657$
- $(n + \log n)^3$
- $n(2 + \log n)$
- $1 + 2 + 3 + ... + n$

$O(n^2)$
$O(n^3)$
$O(1)$
$O(n^3)$
$O(n \log n)$
$O(n)$

Calculating Time Complexity

Python Code: Factorial

```
def factorial1(n):
  if n <= 1:
    return 1
  else:
    fact = 1
    for i in range(2,n+1):
      fact *= i
    return fact
```

→ $T(1)$

→ $T(1)$

→ $T(n)$

→ $T(1)$

$T(n) = 2 + n + 1 = n + 3$

Complexity: $O(n)$

Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Calculating Time Complexity

Python Code: Simple nested loops

```
def simple(n):
  for i in range(n):
    for j in range(n):
      print("i: {0}, j: {1}".format(i,j))
```

→ $T(n)$

→ $T(n)$

$T(n) = n*n$

Complexity: $O(n^2)$

Calculating Time Complexity

Python Code: Element uniqueness v1

```
def unique1(s):
  for i in range(len(s)):
    for j in range(i+1, len(s)):
      if s[i] == s[j]:
        return False  # Found duplicate pair
  return True       # All elements are unique
```

→ *T*(n)

→ *T*(n)

T(n) = n(n-1)/2

Complexity: *O*(n^2)

Calculating Time Complexity

Python Code: Element uniqueness v2

```
def unique2(s):
  temp = sorted(s)  # create a sorted copy of s
  for i in range(1, len(temp)):
    if temp[i-1] == temp[i]:
      return False   # Found duplicate pair
  return True        # All elements are unique
```

→ *T*(n log n)

→ *T*(n)

T(n) = n log n + n

Complexity: *O*(n log n)

Calculating Time Complexity

Python Code: Prefix averages v1

```
def prefix_average1(s):
    n = len(s)
    a = [0] * n                  # create list of n zeros
    for i in range(n):
        total = 0                # compute each element
        for j in range(i+1):
            total += s[j]
        a[i] = total / (i+1)     # record the average
    return a
```

→ $T(1)$

→ $T(n)$

→ $T(n)$

→ $T(n)$

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

Complexity: $O(n^2)$

Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Calculating Time Complexity

Python Code: Prefix averages v2

```
def prefix_average2(s):
    n = len(s)
    a = [0] * n                  # create list of n zeros
    for i in range(n):
        a[i] = sum(s[0:i+1])/(i+1) # record the average
    return a
```

→ $T(1)$

→ $T(n)$

→ $T(n)$

→ $T(n)$

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

Complexity: $O(n^2)$

Calculating Time Complexity

Python Code: Prefix averages v3

```
def prefix_average3(s):
  n = len(s)
  a = [0] * n                # create list of n zeros
  total = 0
  for i in range(n):
    total += s[i]      # update total sum to include s[i]
    a[i] = total / (i+1) # compute average based on
current sum
  return a
```

$T(1)$
$T(n)$
$T(1)$

$T(n)$

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

v3 is asymptotically better than v2 and v1

Complexity: $O(n)$

Exercise 1

Python Code

```
def example1(s):
  n = len(s)
  total = 0
  for i in range(n):     # loop from 0 to n-1
    total += s[i]
  return total
```

Complexity: $O(?)$

Exercise 2

Python Code

```
def example2(s):
  n = len(s)
  total = 0
  for i in range(0,n,2):   # Increment of 2
    total += s[i]
  return total
```

Complexity: O(?)

Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Exercise 3

Python Code

```
def example3(s):
  n = len(s)
  total = 0
  for i in range(n):       # loop from 0 to n-1
    for k in range(1+i):   # loop from 0 to i
      total += s[k]
  return total
```

Complexity: O(?)

Exercise 4

Python Code

```
def example4(s):
  n = len(s)
  prefix = 0
  total = 0
  for i in range(n):
    prefix += s[i]
    total += prefix
  return total
```

Complexity: *O*(?)

Exercise 5

Python Code

```
# Assume that A and B have equal length of n
def example5(A,B):
  n = len(A)
  count = 0
  for i in range(n):          # loop from 0 to n-1
    total = 0
    for j in range(n):        # loop from 0 to n-1
      for k in range(1+j):    # loop from 0 to j
        total += A[k]
    if B[i] == total:
      count += 1
  return count
```

Complexity: *O*(?)