Lecture 1 - Introduction

CPE112 - Programming with Data Structures

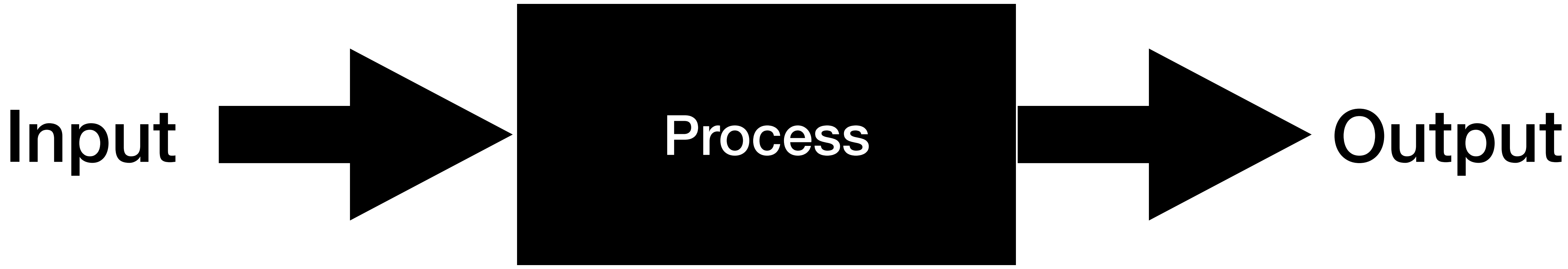
19 January 2024

Dr. Piyanit Wepulanon
Dr. Taweechai Nuntawisuttiwong

Department of Computer Engineering
KMUTT

What to learn?

Programming with Data Structures



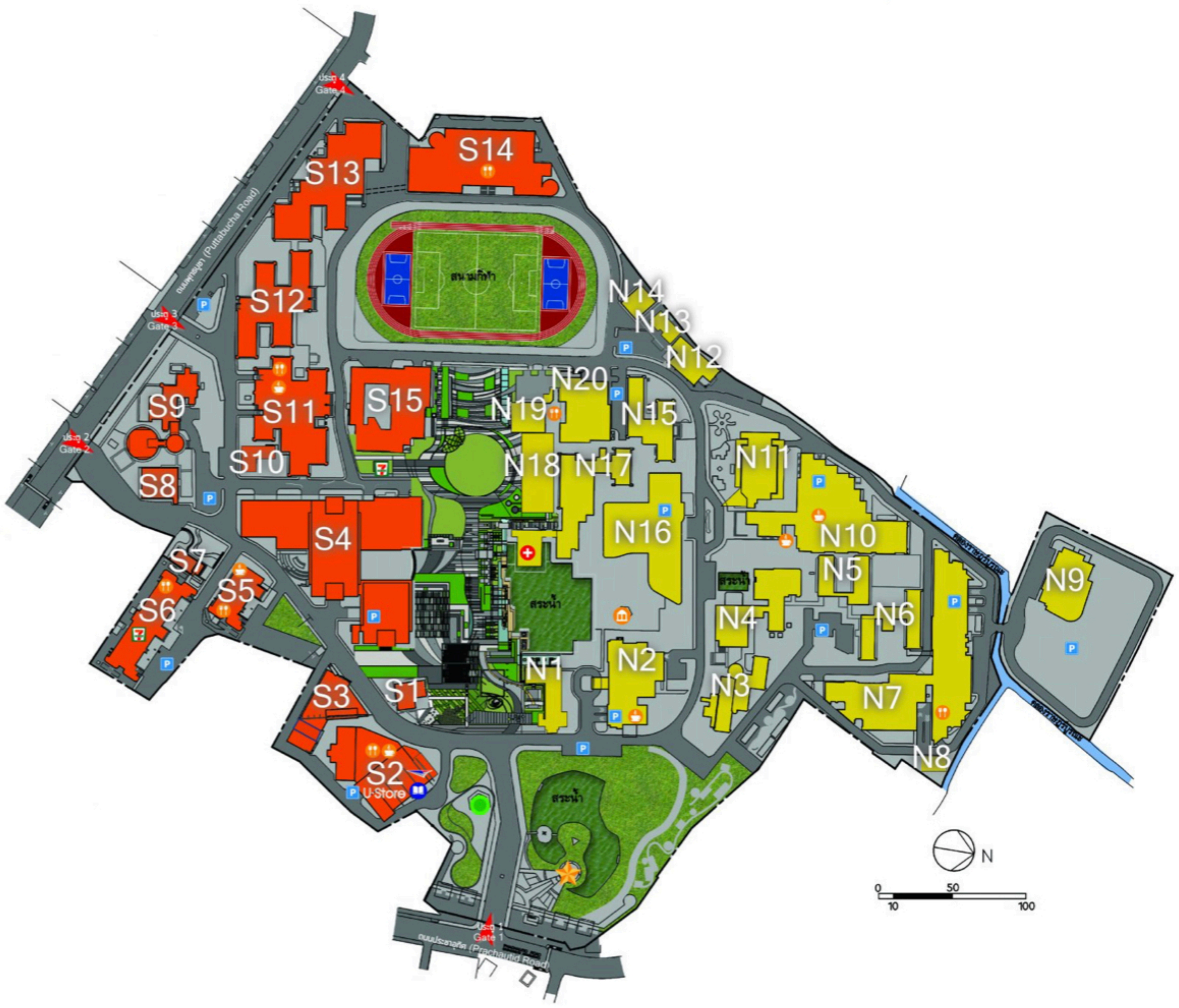
- Programs manipulate data to solve problems
- Ways to organize & store information
  - Insert/delete/sort numbers: array
  - Student information: structure

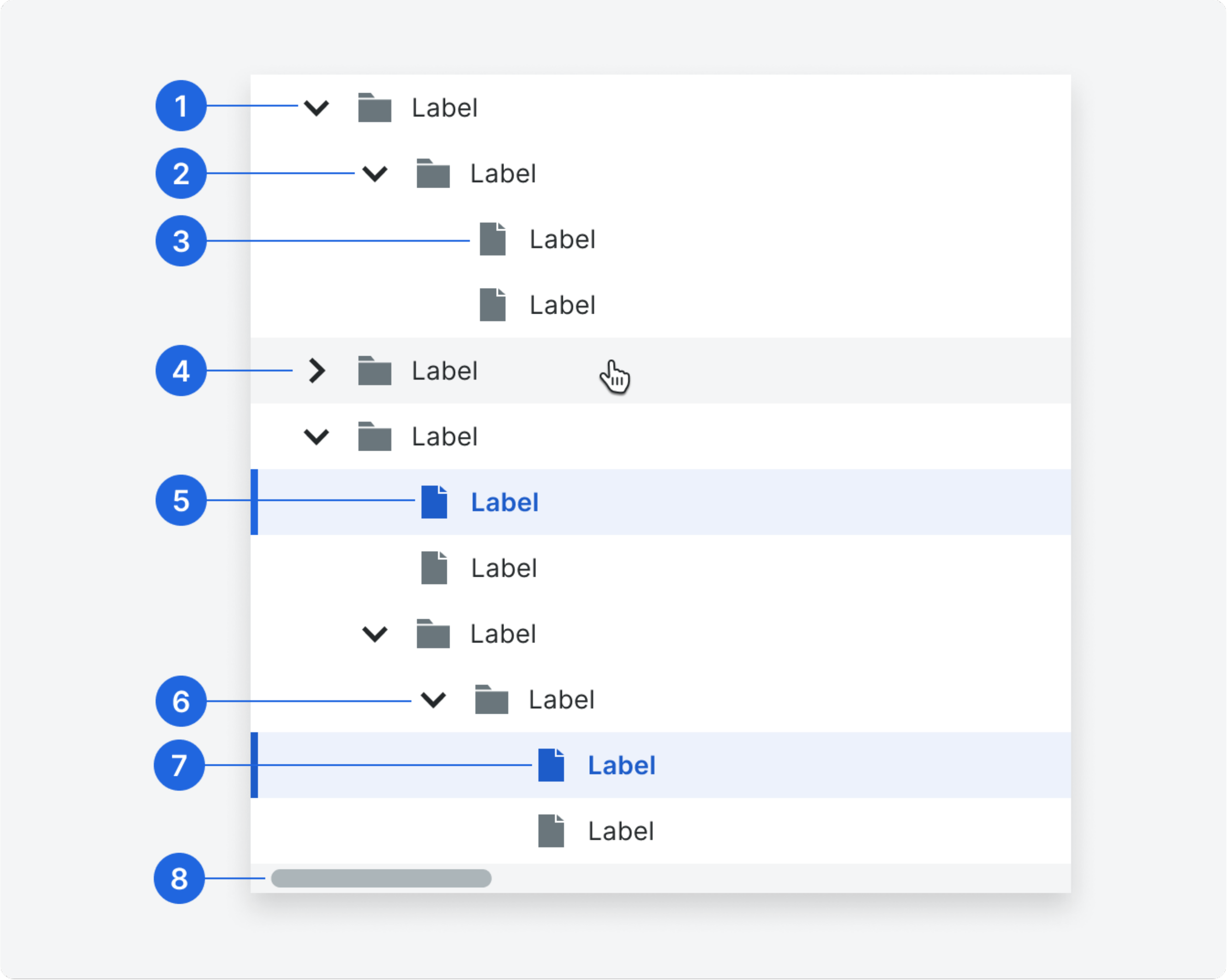
Score
67
78
87
90
82
65
68

Name	Midterm	Final
Alice	7.5	67
Bob	8.2	78
Carol	6.0	87
Dave	9.1	90
Eve	8.4	82
Felix	6.5	65
George	6.8	68

What to learn?

Programming with Data Structures



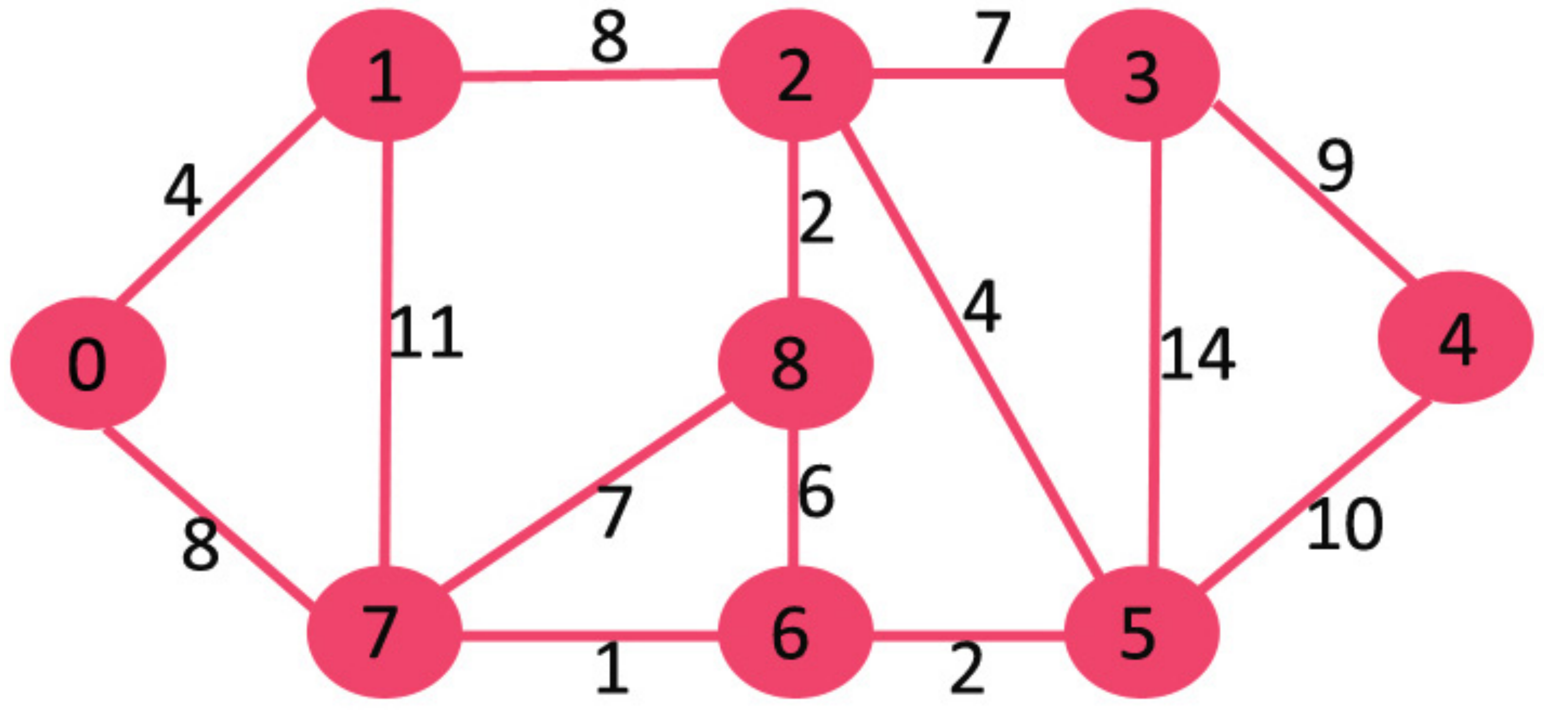


1. Expanded Folder (1st level)
2. Expanded Folder (2nd level)
3. Unselected File (2nd level)
4. Collapsed Folder (1st level) / Hover
5. Selected File (1st level)
6. Expanded Folder (3rd level)
7. Selected File (3rd level)
8. Horizontal Scrollbar

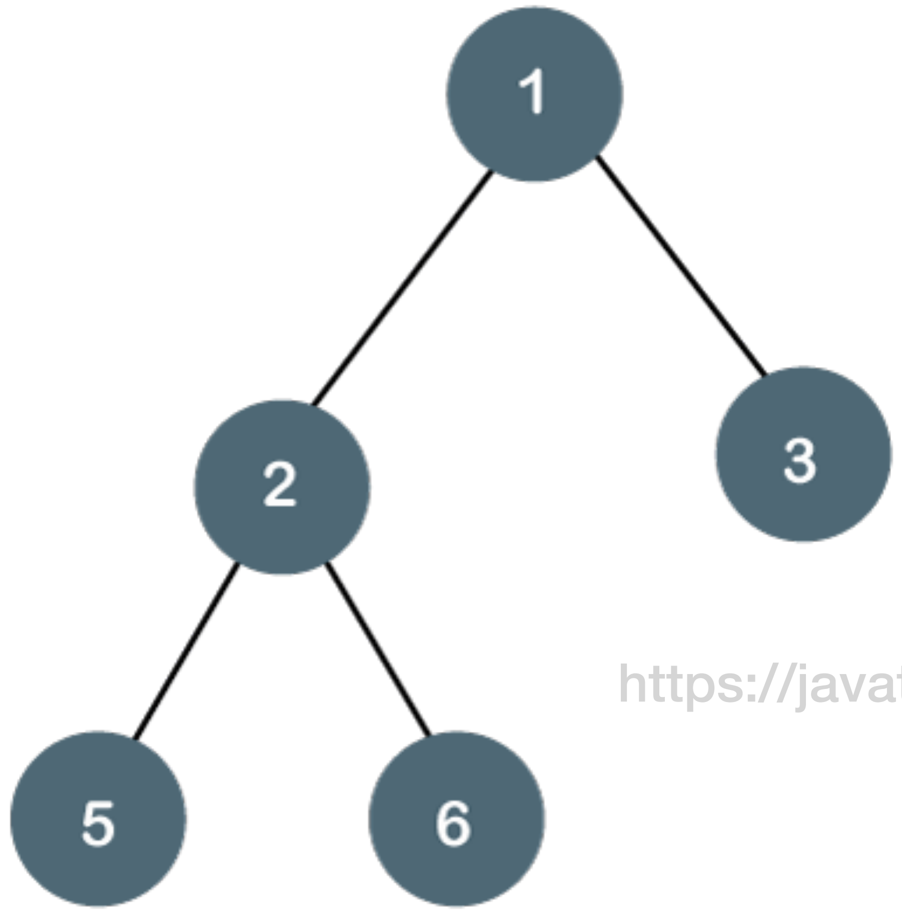
https://design.procore.com

What to learn?

Programming with Data Structures



https://geeksforgeeks.org



https://javatpoint.com

- Important **data structures** - strategies for organizing information in a program

- Important **algorithms** - well-known methods for accomplishing particular kinds of tasks

- **Efficiency** of different data structure algorithms

Data Structure & Algorithm

Google Coding interview question via phone

Find the first recurring character

“ABCA” -> A
“BCABA” -> B
“ABC” -> NULL

Data Structure & Algorithm

Google Coding interview question via phone

Find the first recurring character

“ABCA” -> A
“BCABA” -> B
“ABC” -> NULL

Your algorithm will be evaluated.

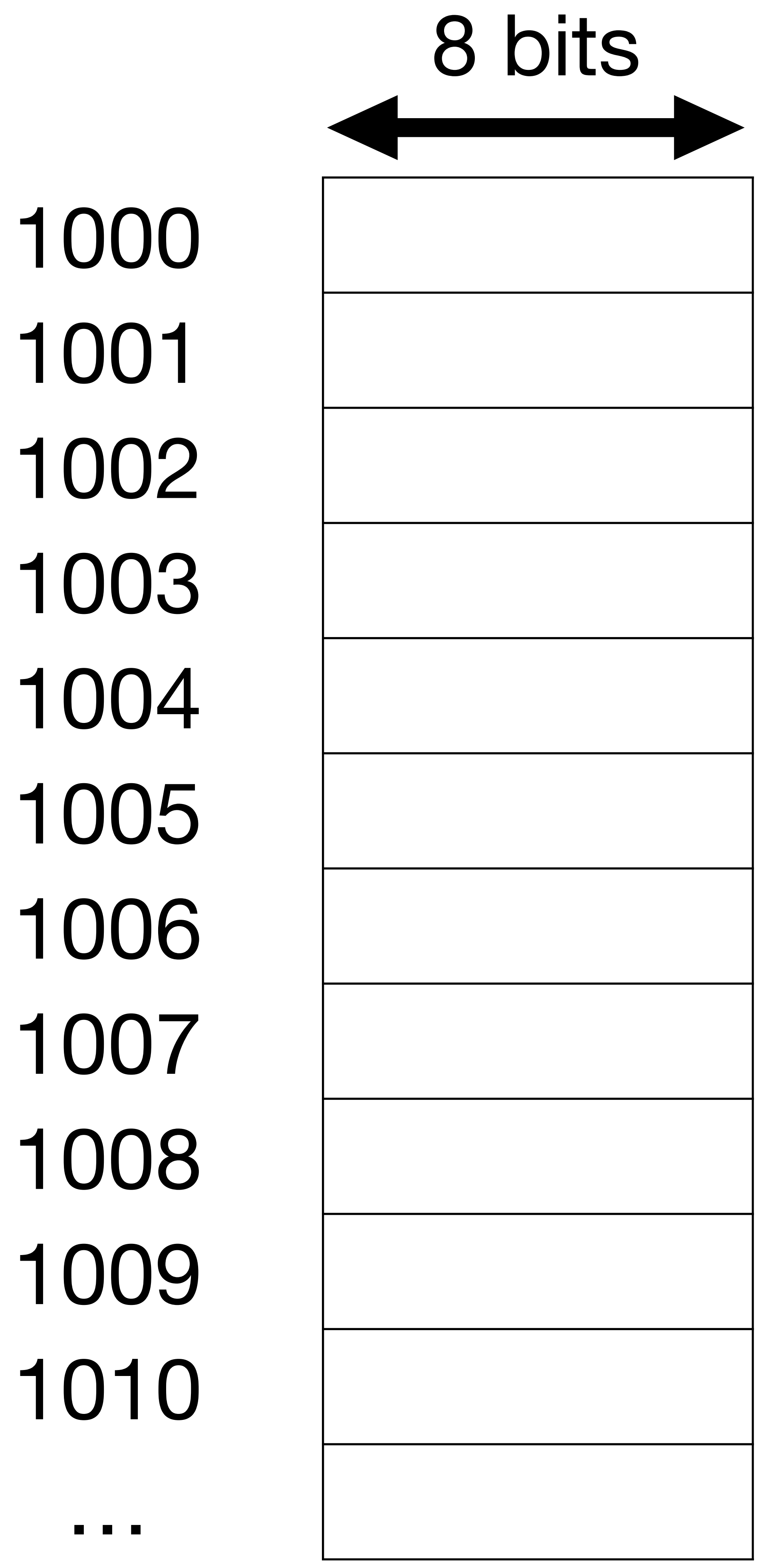
Correctness (work well)
Efficiency (minimum time & memory)
Style (easy to understand & modify)

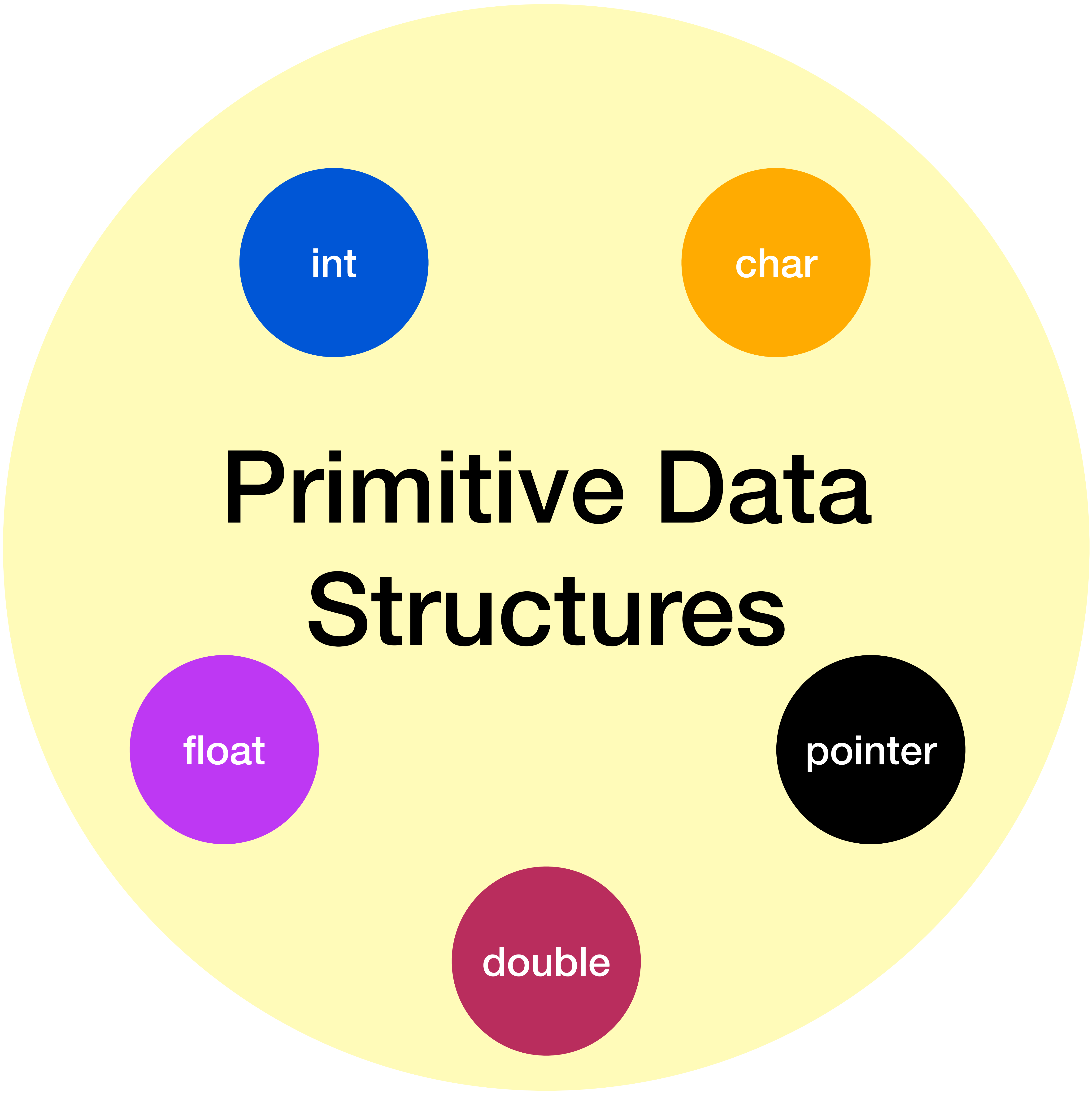
Major Contents

- Introduction
- Pointer
- Array
- Structure
- Lists
- Stack & Queue
- Tree
- Heaps
- Graph
- Hashing
- …

Review

Memory





Review

Pointer

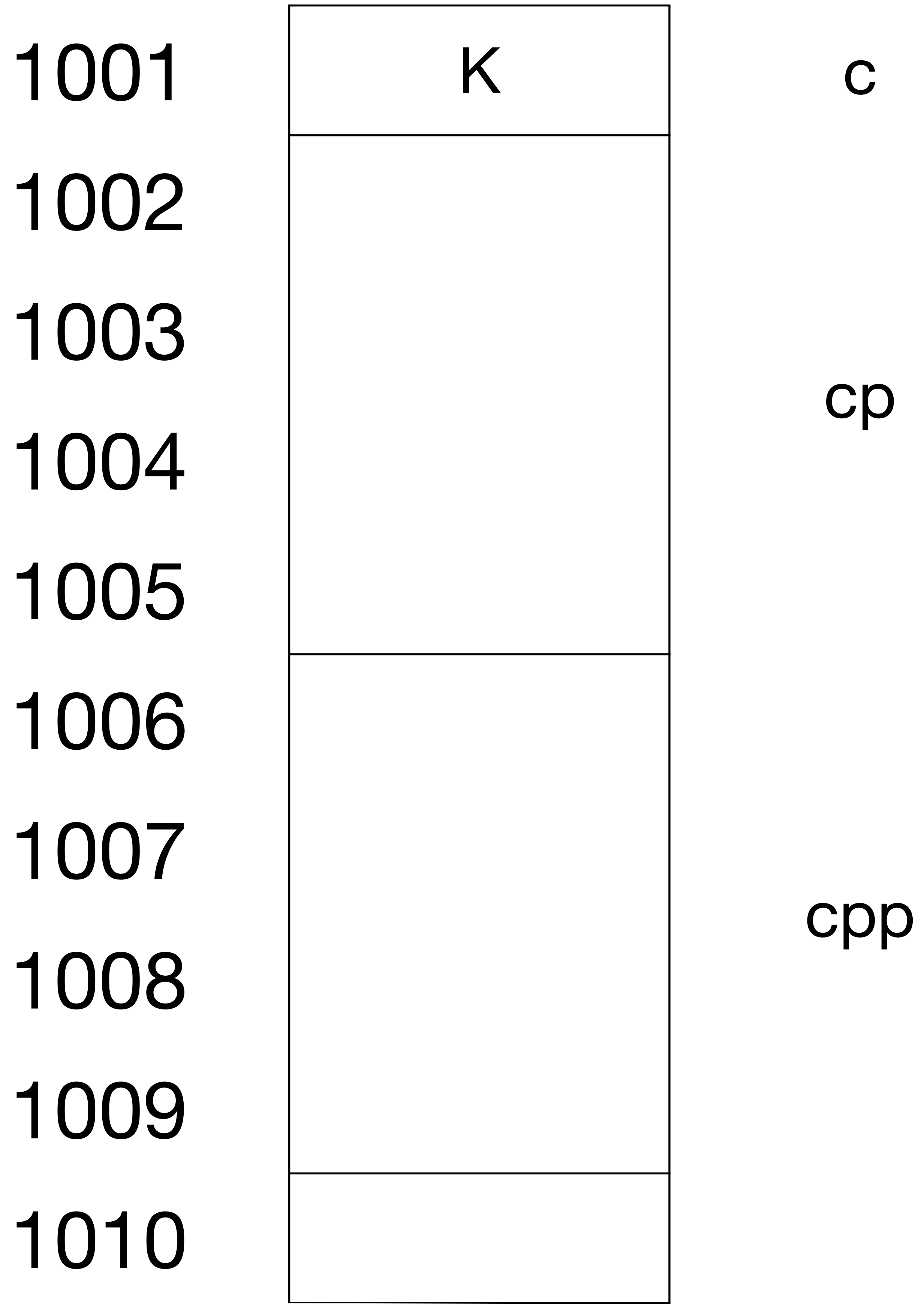
- A variable type - store the address of some value

& - Get address

**** - Look into address***

```
data-type *pointer-variable-name;
```

```
char c='K';
char *cp=&c;
char **cpp=&cp;
```



1001	K	c
1002		cp
1003		
1004		
1005		
1006		cpp
1007		
1008		
1009		
1010		

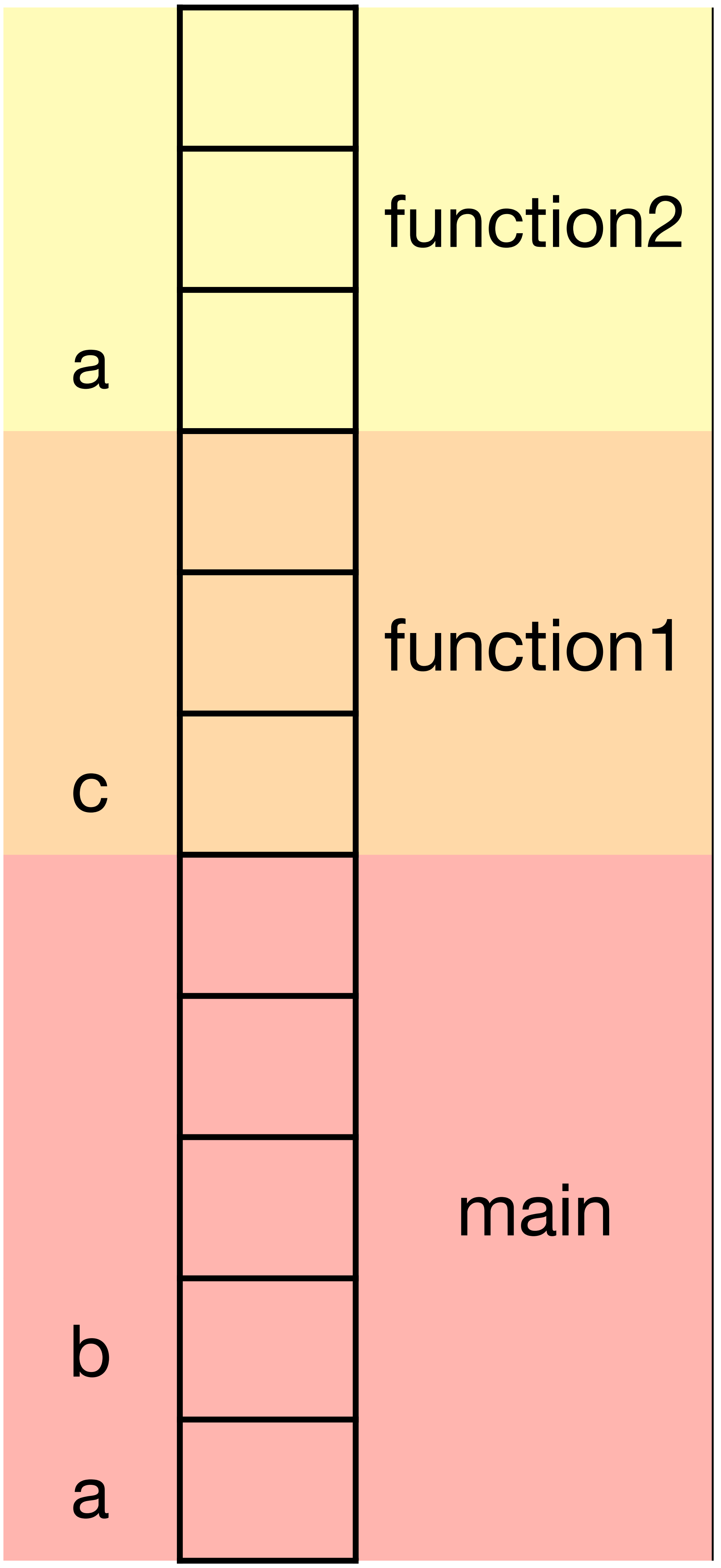
Review

Types of Memory - Stack

```
void function2()
{
  int a;
}

void function1()
{
  int c;
  function2();
}

int main ()
{
  int a,b;
  function1();
  return 0;
}
```

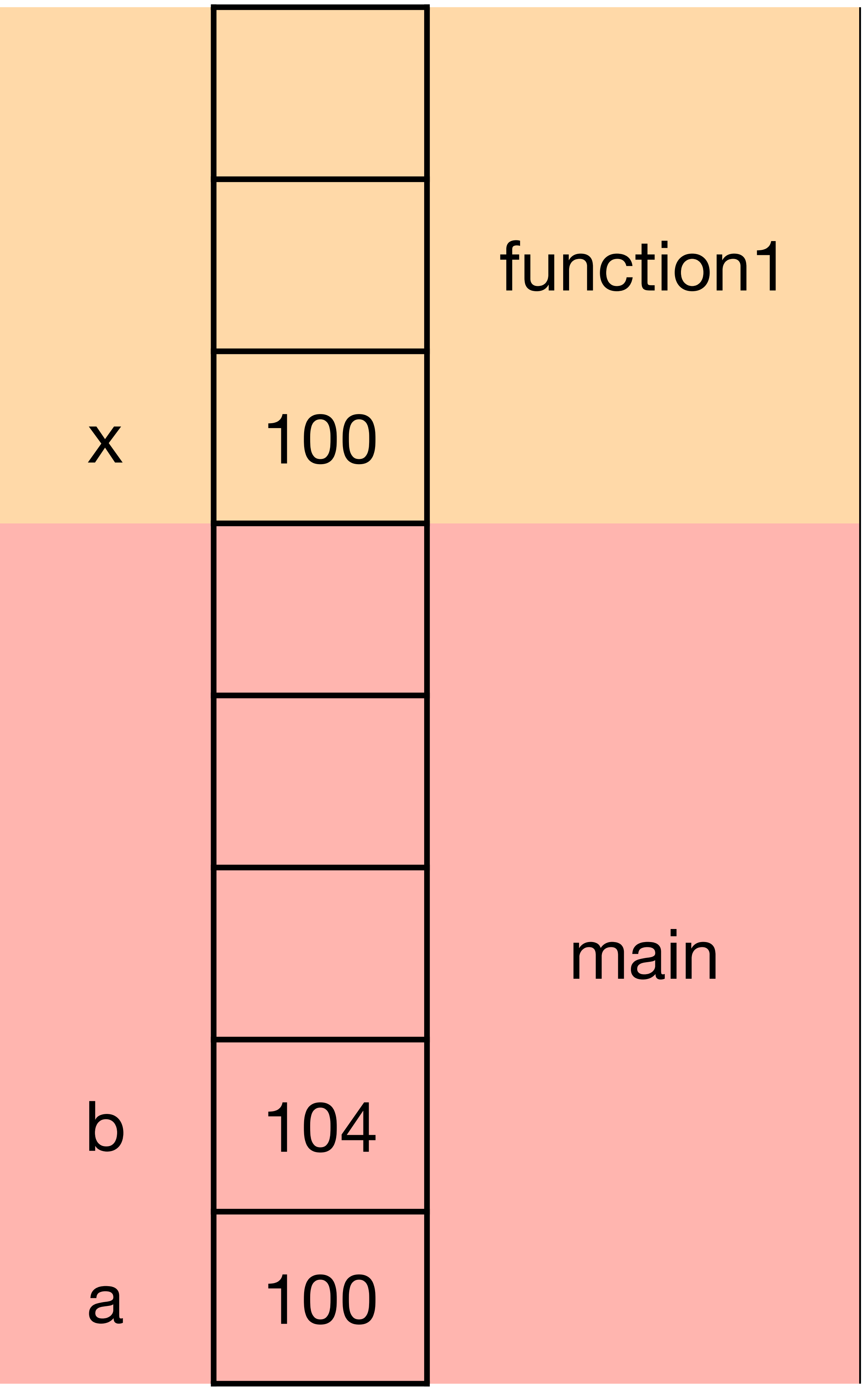


Review

Types of Memory - Heap

```
void function1(int* x)
{
  int x;
  x[0] = /*…*/;
}

int main ()
{
  int *a = malloc(4);
  int *b = malloc(6);
  function1(a);
  free(a);
  free(b);
  return 0;
}
```



function1

x 100

main

b 104

a 100

Review

Types of Memory

	Life cycle (How & when they are allocated/ deallocated)	**Scope** (Where they can be accessed)
Stack	**Allocate** - program enters the function that they are declared **Deallocate** - program leaves the function	Can be accessed and will have valid value <u>within their own block</u>
Static	**Allocate** - program starts running **Deallocate** - program exits	Can be accessed by <u>any code in C module</u>
Dynamic	**Allocate** - call calloc **Deallocate** - call free	Can be accessed and will have valid values <u>anywhere in the program, by passing around variables</u> that hold the start address

Review

Dynamic Memory Allocation

Stack Memory

1000	H
1001	i
1002	\0
1003	C
1004	P
1005	E
1006	1
1007	1
1008	2
1009	\0
1010	
…	

Example:

```
char word1[3] = “Hi”;
char word2[10] = “CPE112”
```

“Hi” -> “Hello”

Heap Memory

calloc

free

Example:

```
#define ARRAYSIZE 300
int* myValues = calloc(ARRAYSIZE,sizeof(int));
```

Address

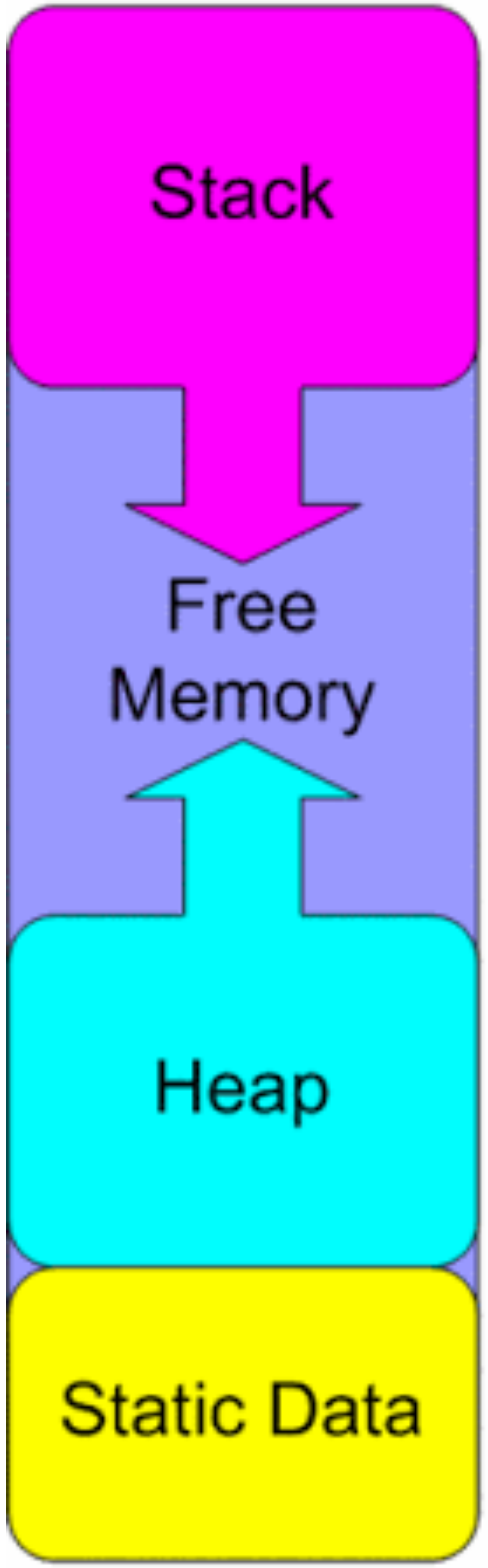
Review

Dynamic Memory Allocation

calloc

but not free

Memory crash!!!



https://esp8266.com

Review

Structure

- Create own custom data type that is a composition of multiple other data types

Name	Midterm	Final
Alice	7.5	67
Bob	8.2	78
Carol	6.0	87
Dave	9.1	90
Eve	8.4	82
Felix	6.5	65
George	6.8	68

```
typedef struct {
   char name[100];
   float midterm;
   int final;
} student_info;
```

Review

Structure

```
void selectionSort (student_info data[], int count)
{
  int i, j, minPos;
  for (i = 0, i<count-1; i++)
  {
    minPos = i;
    for (j=i+1; j<count; j++)
    {
      if (data[j].final < data[minPos].final)
        minPos = j;
    }
    swap(&data[i], &data[minPos]);
  }
}
```

```
void swap (student_info *x,
student_info *y)
{
  student_info tmp;
  tmp = *x; *x=*y; *y=tmp;
}
```

Data Structure

- A data structure - a way to organize information stored in computer memory.
- Different kinds of data structures works better for different sorts of problems.
- Appropriate structure -> Good solution

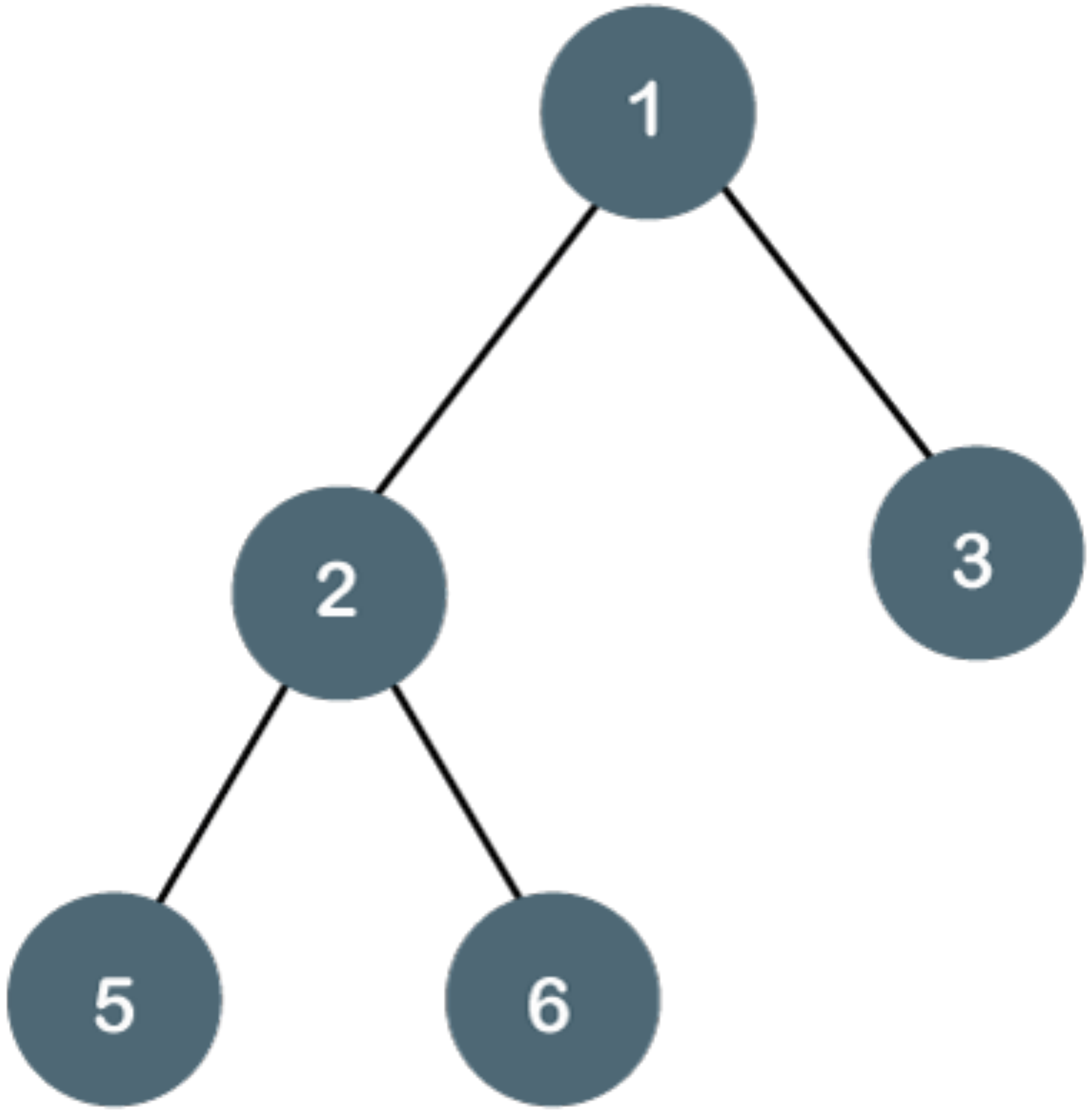
Score
67
78
87
90
82
65
68

Name	Midterm	Final
Alice	7.5	67
Bob	8.2	78
Carol	6.0	87
Dave	9.1	90
Eve	8.4	82
Felix	6.5	65
George	6.8	68

Data Structure

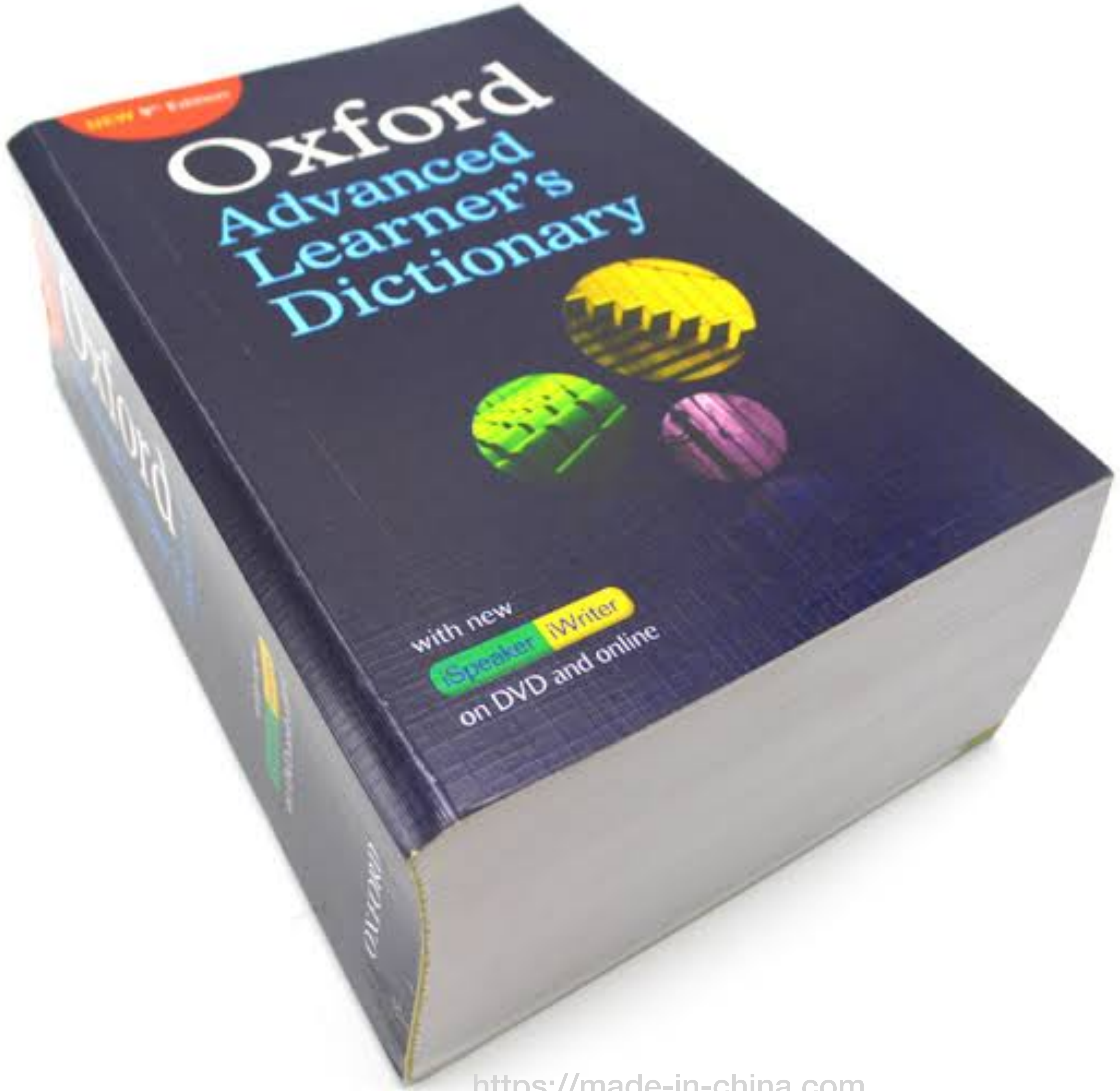
- Creating a data structure
  - Ability to refer to different memory locations
  - Stitch them together into a custom shape

1000	
1001	
1002	
1003	
1004	
1005	
1006	
1007	
1008	
1009	
1010	
…	



Data Structure & Algorithm

Example



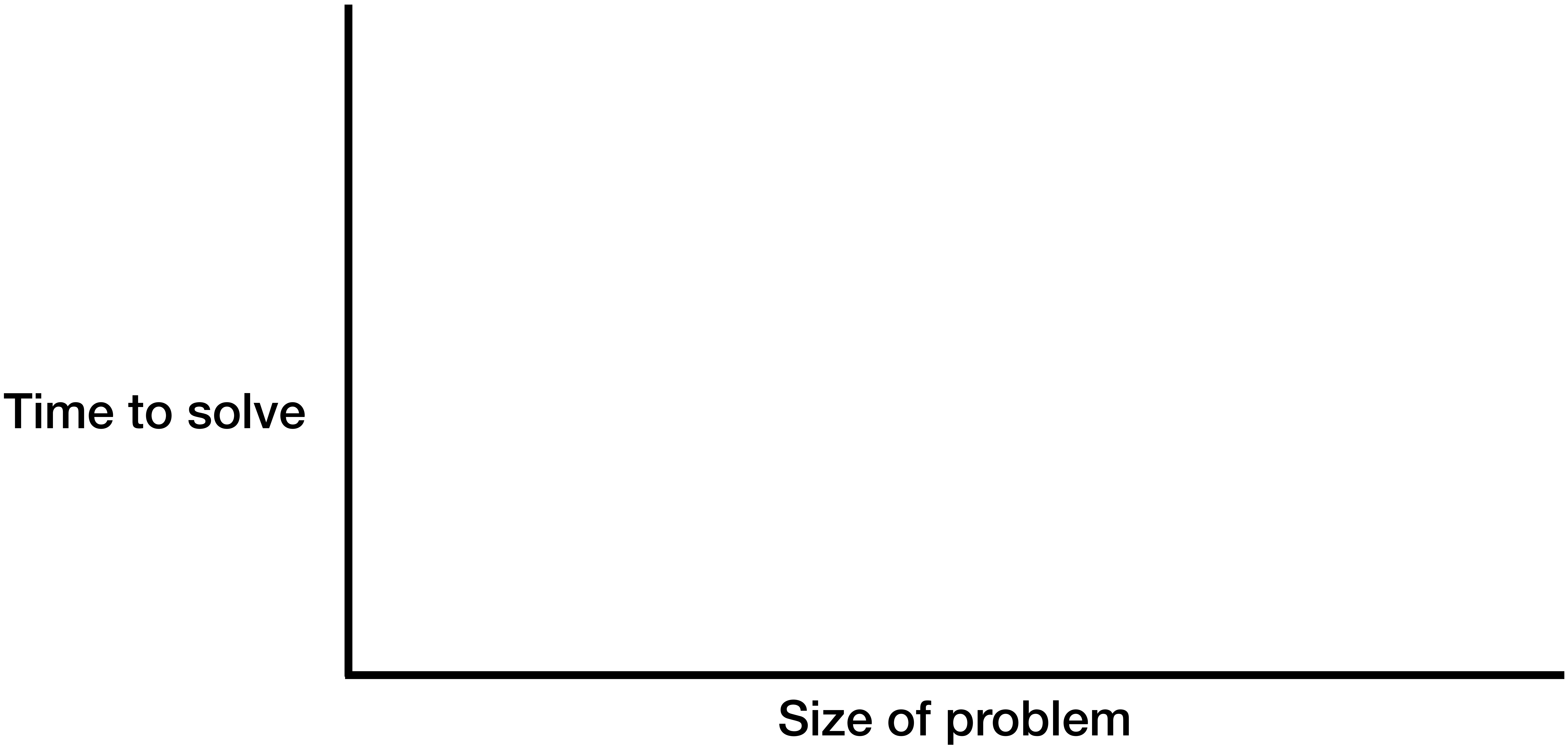
https://made-in-china.com

How to find "Structure"?

Algorithm Efficiency

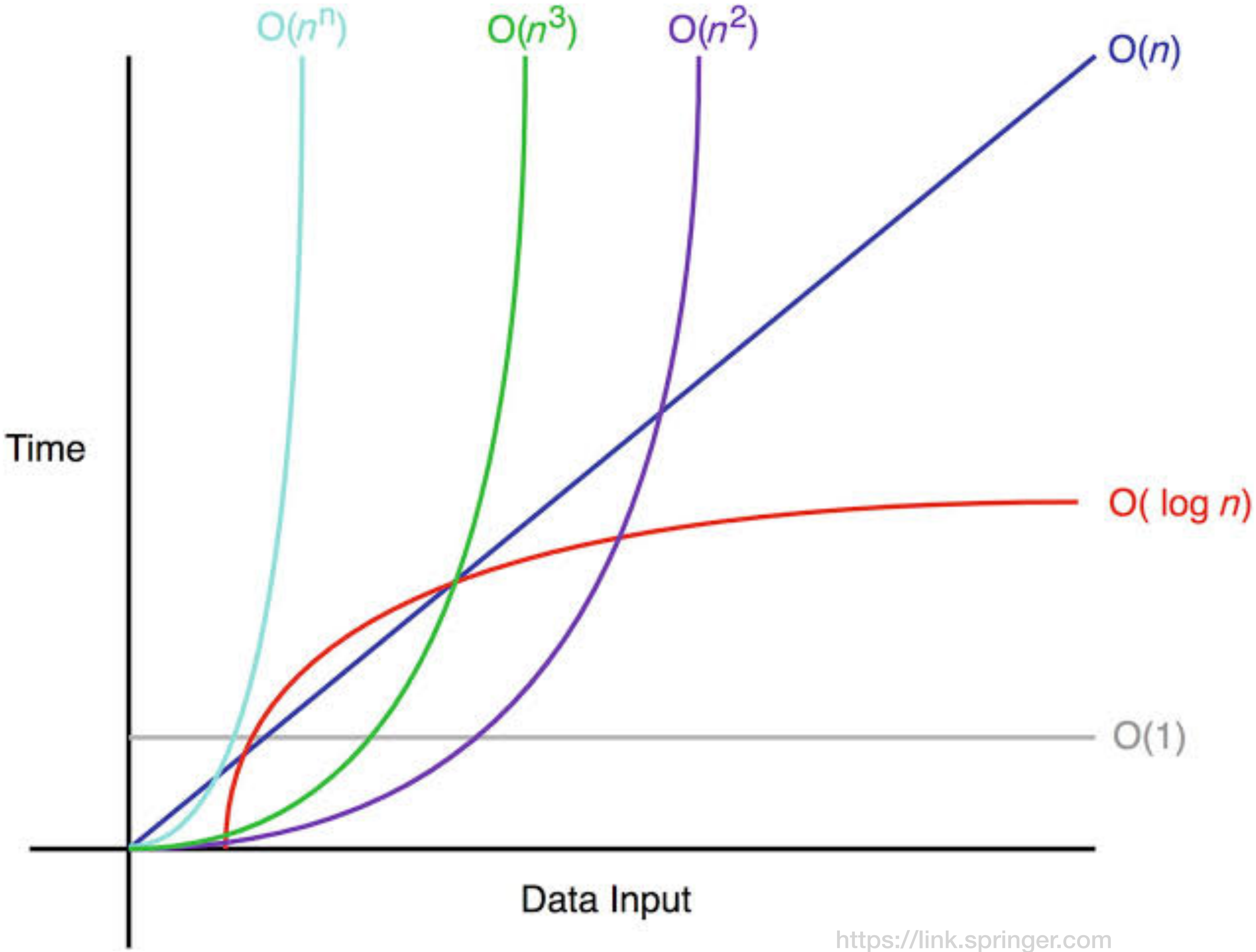
	#operations for 1 page	#operations for 100 page	#operations for 1000 page	…
Method#1				
Method#2				
…				

Algorithm Efficiency

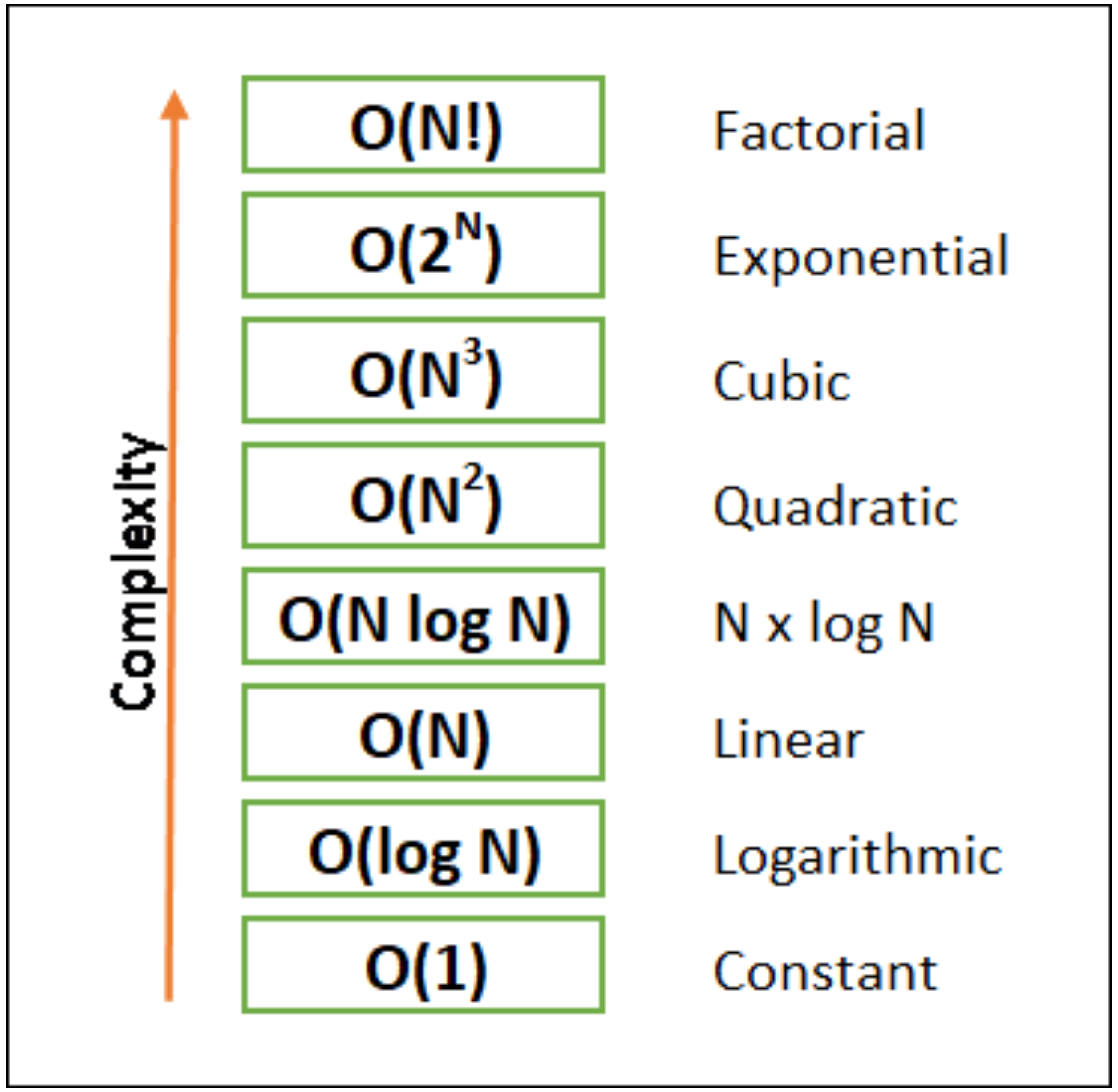


Algorithm Efficiency

Big O Notation



https://link.springer.com



O(N!)	Factorial
$O(2^N)$	Exponential
$O(N^3)$	Cubic
$O(N^2)$	Quadratic
O(N log N)	N x log N
O(N)	Linear
O(log N)	Logarithmic
O(1)	Constant

https://towardsdatascience.com

Algorithm Efficiency

Big O Notation

1. Understand how the algorithm works
2. Identify a basic unit of the algorithm to count
3. Map growth of count from step 2 to appropriate Big O class

Algorithm Efficiency

Big O - Examples

1	x = 5 + (5 * 5);	O(…)

2	for (i = 0; i < N; i++)
	   printf(“%d”, i);
		O(…)

3	for (i = 0; i < N; i++)
	   for (j = 0; i < N; i++)
	      printf(“%ld”, i*j);
		O(…)

4	x = 5 + (5 * 5);
	for (i = 0; i < N; i++)
	   printf(“%d”, i);
	for (i = 0; i < N; i++)
	   for (j = 0; i < N; i++)
	      printf(“%ld”, i*j);
		?

Algorithm Efficiency

Big O Definition

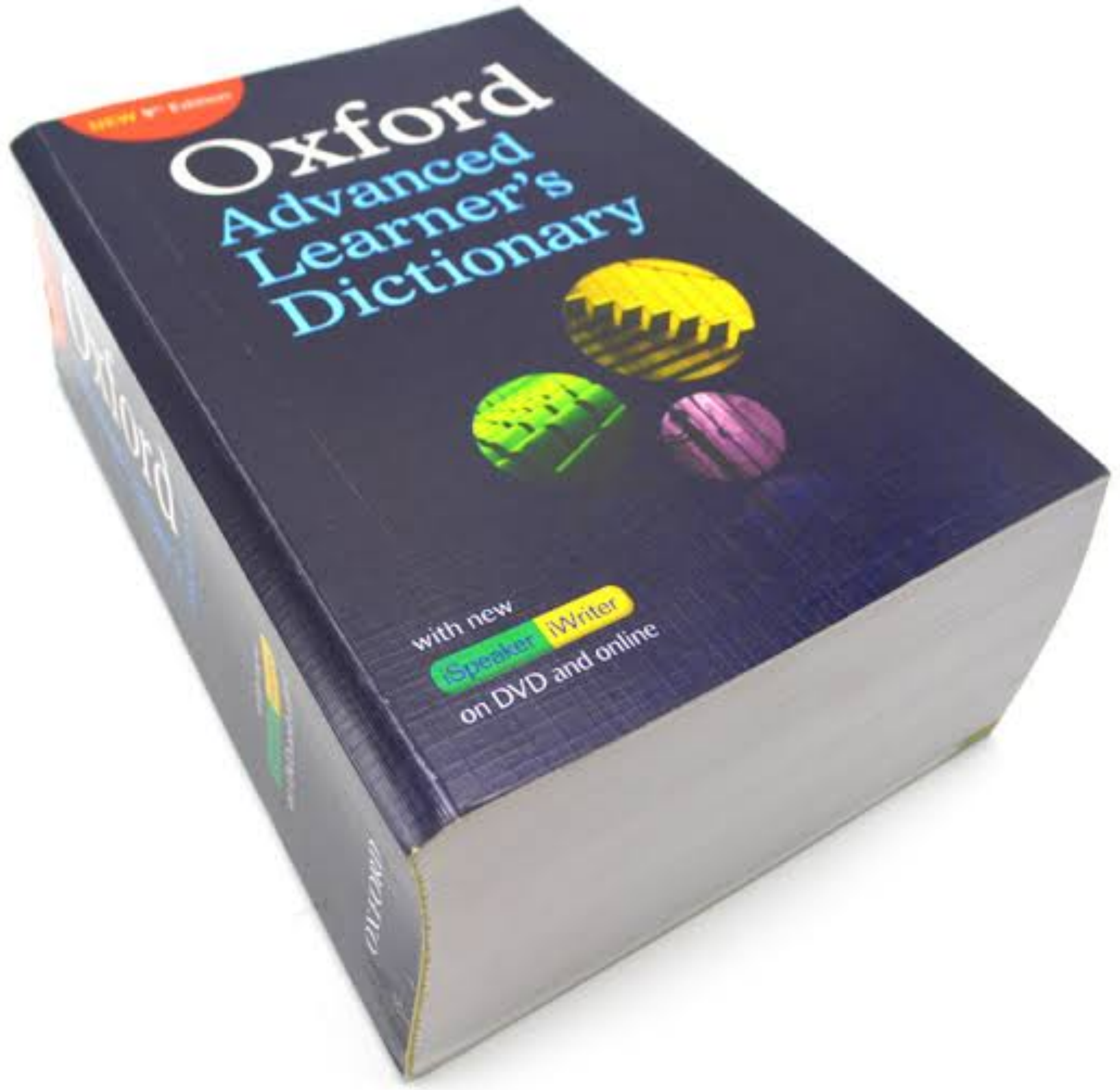
Let $f(n)$ and $g(n)$ be functions from positive integers to positive reals.

$f(n) = O(g(n))$ if there is a constant $c > 0$ such that $f(n) \leq c*g(n)$ for large n

นิยาม 1: $f(n)$ และ $g(n)$ เป็นฟังก์ชันซึ่งมีโดเมนเป็นเลขจำนวนเต็มบวก และมี ranges เป็นจำนวนจริงบวก
สามารถเขียนได้เป็น
$f(n) = O(g(n))$ ถ้ามีค่าคงที่บวก c ที่ทำให้ $f(n) \leq c*g(n)$ เป็นจริงทุกค่าสำหรับ x ที่เป็นจำนวนเต็มบวก

Algorithm Efficiency

Big O - Examples



Find “Ant”, “List”, “Zombie”, …

Algorithm Efficiency

Cases

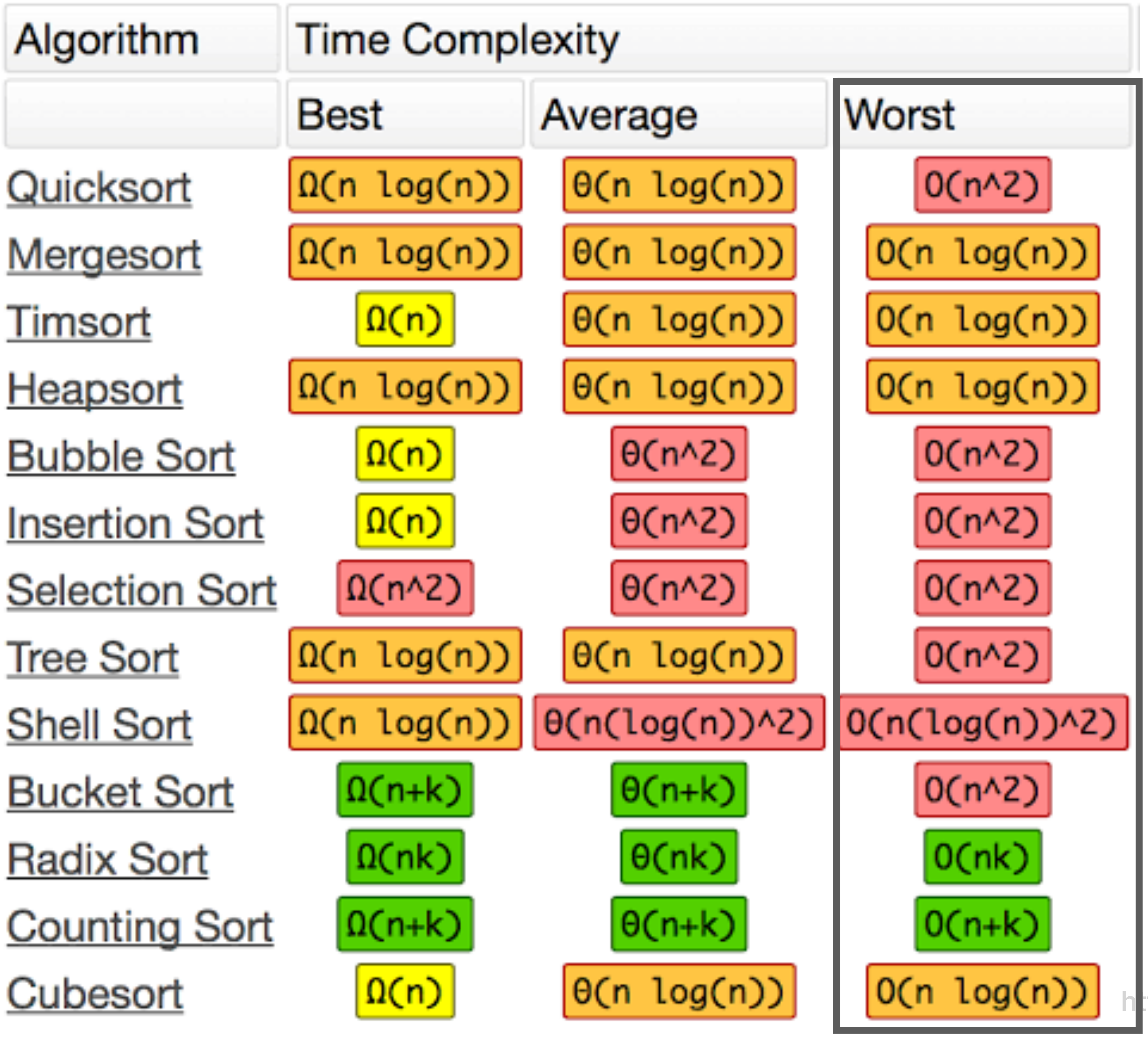
Find character ‘A’ in a given string length n

Best case	$C(n) = 1$
Worst case	$C(n) = n$
Average case	$C(n) = (1+2+3+\ldots+n)1/n$ $= [n(n+1)/2] * [1/n]$ $= (n+1)/2$

Algorithm Efficiency

Big O

Complexity Summary of Array Sorting Algorithms



Algorithm	Time Complexity		
	Best	Average	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))

https://tonysispblog.wordpress.com

Designing an Algorithm

- Perform operations on the stored data contained in data structures.
- **Modularization process** - a complex algorithm is often divided into smaller units called modules.
  - Top-down approach - dividing the complex algorithm into one or more modules
  - Bottom-up approach - designing the most basic or concrete modules and then proceed towards designing higher level modules

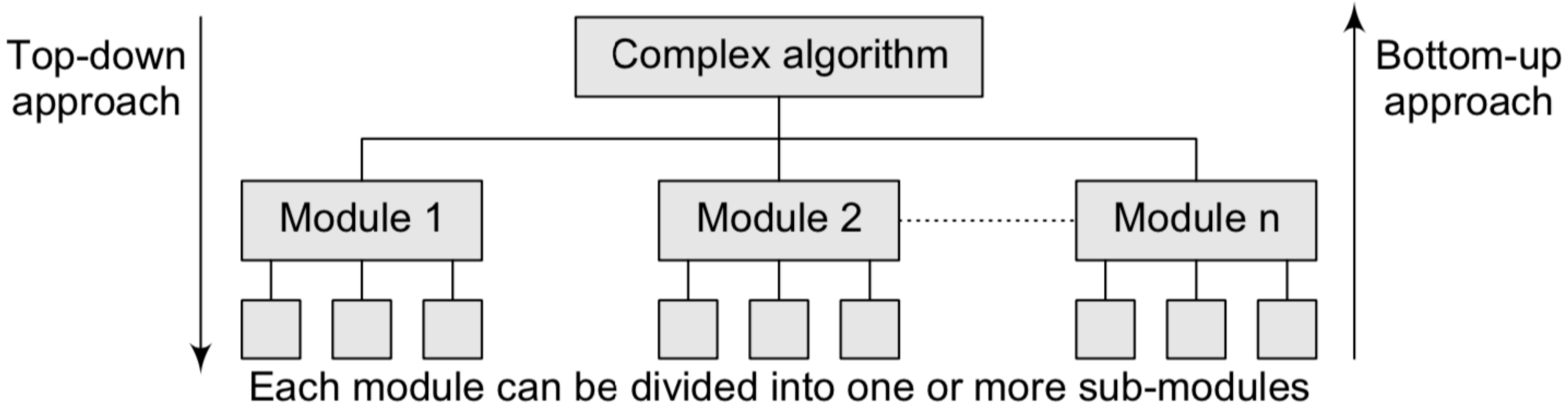


Figure 2.9 Different approaches of designing an algorithm

Wrap up

- Course introduction
- What to learn?
- Data structures & algorithms
- Review CPE100
- Big O