



Vietnam National University of HCMC
International University
School of Computer Science and Engineering



Data Structures and Algorithms

★ Arrays ★

Dr Vi Chi Thanh - vcthanh@hcmiu.edu.vn

<https://vichithanh.github.io>



SCAN ME

Week by week topics (*)

1. OOP and Java

2. Arrays

3. Sorting

4. Queue, Stack

5. List

6. Recursion

Mid-Term

7. Advanced Sorting

8. Binary Tree

9. Hash Table

10. Graphs

11. Graphs Adv.

Final-Exam

8 LABS

Objective

- + Basics of Array in Java
- + Linear search - Binary search
- + Storing objects
- + Big O notation

Introduction

+How do we store list of integer entered by user?

```
int num1;
```

```
int num2;
```

```
int num3;
```

+If there are 100 of them !?

Introduction

- + Array is the most commonly used data structure
- + Built into most of the programming languages

123	412	19	20	
-----	-----	----	----	--

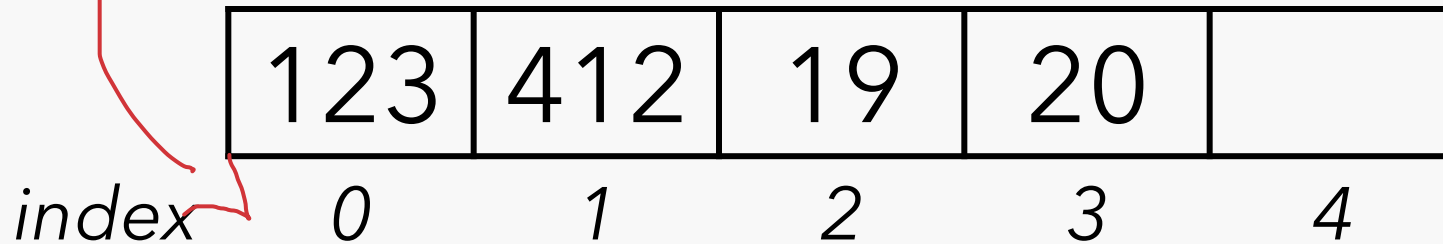
Definition

+ An ARRAY is a collection of variables all of the same TYPE

+ Element

+ Index / positions

+ In Java



Initialization

- + By default, an array of integers is automatically initialized to 0 when it's created

```
autoData[] carArray = new autoData[4000];
```

- + Initialize an array of a primitive type

```
int[] intArray = { 0, 3, 6, 9, 12, 15, 18, 21, 24, 27 };
```

Accessing array elements

+ Element is access by index number via subscript

+ In Java,

ArrayName[index]

+ **A[3] , A[0]**

+ **A[6]**

A	123	412	19	20	
index	0	1	2	3	4

Example in Java

```
int A[];  
int A1[] = new int[100];  
int A2[] = new int { 1, 7, 9, 20};
```

```
for (int i=0; i< 100; i++)  
    A1[i] = i*2;
```

```
for (int j=0; j< A2.length; j++)  
    A2[j] = j*2;
```

Operations on array

+Insertion

+Searching

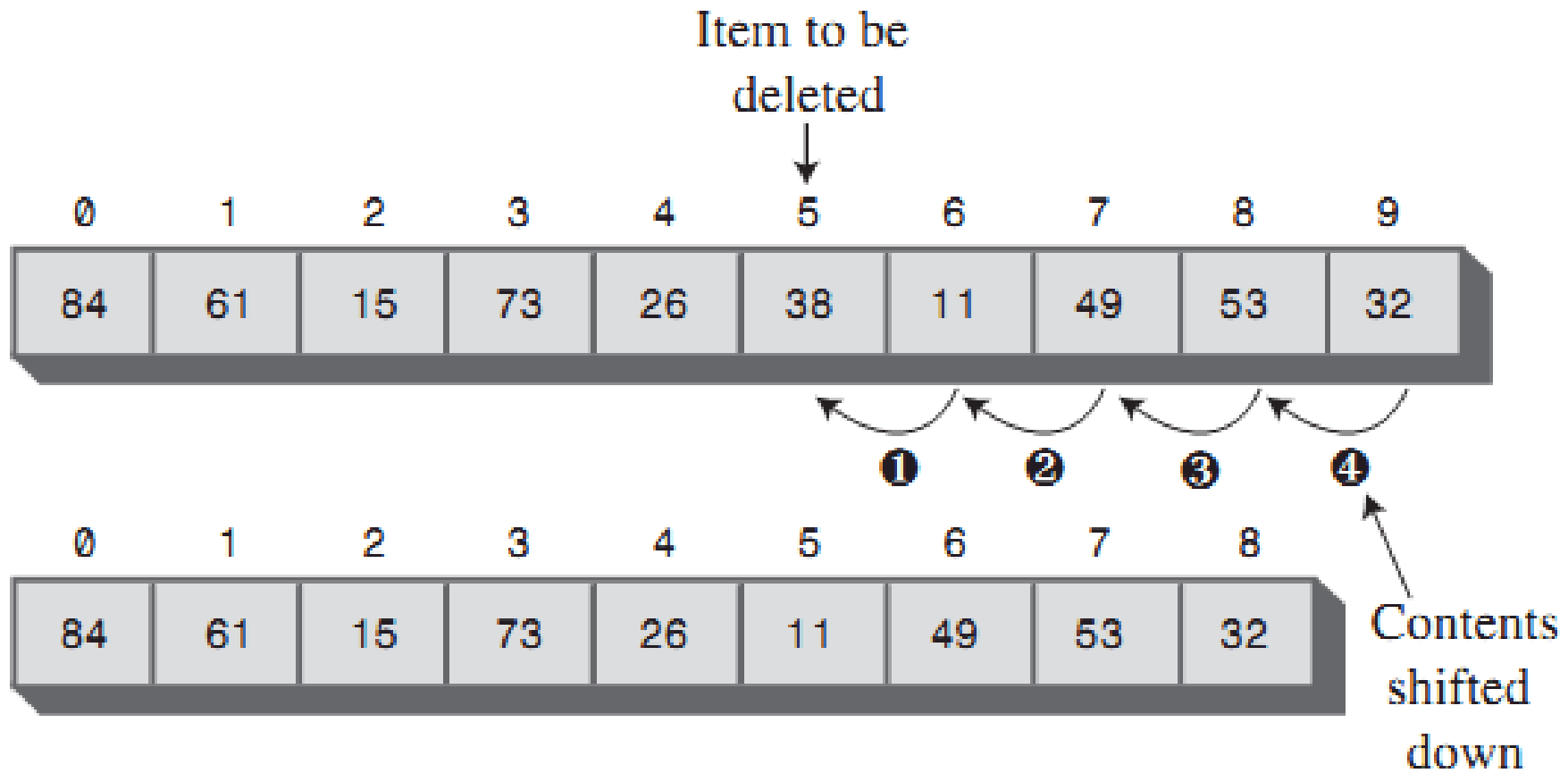
+Deletion

+Duplication issue

→How does it work ?

→Java code in p.41-42

Delete an item



Multi-dimension array

+A matrix

1	0	3	4
5	1	32	12
6	7	1	10
19	5	4	1

Two-dimension array

+Declaration

```
int [][] matrix = new int [ROWS][COLUMNS];  
int [][] matrix2 =  
{  
    {1, 2, 3},  
    {6, 1, 4},  
    {9, 5, 1}  
};
```

+Accessing

+ **matrix[0][10];**

Linear searching technique

+Look for '20' (the SearchKey)

A	123	412	19	20	25
<i>index</i>	0	1	2	3	4

+Step through the array

+Comparing the SearchKey with each element.

+Reach the end but don't find any matched element
→ Can't find

Ordered arrays

+ Data items are arranged in order of key value.

A	412	123	25	20	19
<i>index</i>	0	1	2	3	4

A2	19	20	25	123	412
<i>Index</i>	0	1	2	3	4

Linear searching in ordered array

+Look for '20'

A	412	123	25	20	19
<i>index</i>	0	1	2	3	4

A2	19	20	25	123	412
<i>Index</i>	0	1	2	3	4

Binary searching technique

+ Guess the number between 1 and 100



46



Smaller

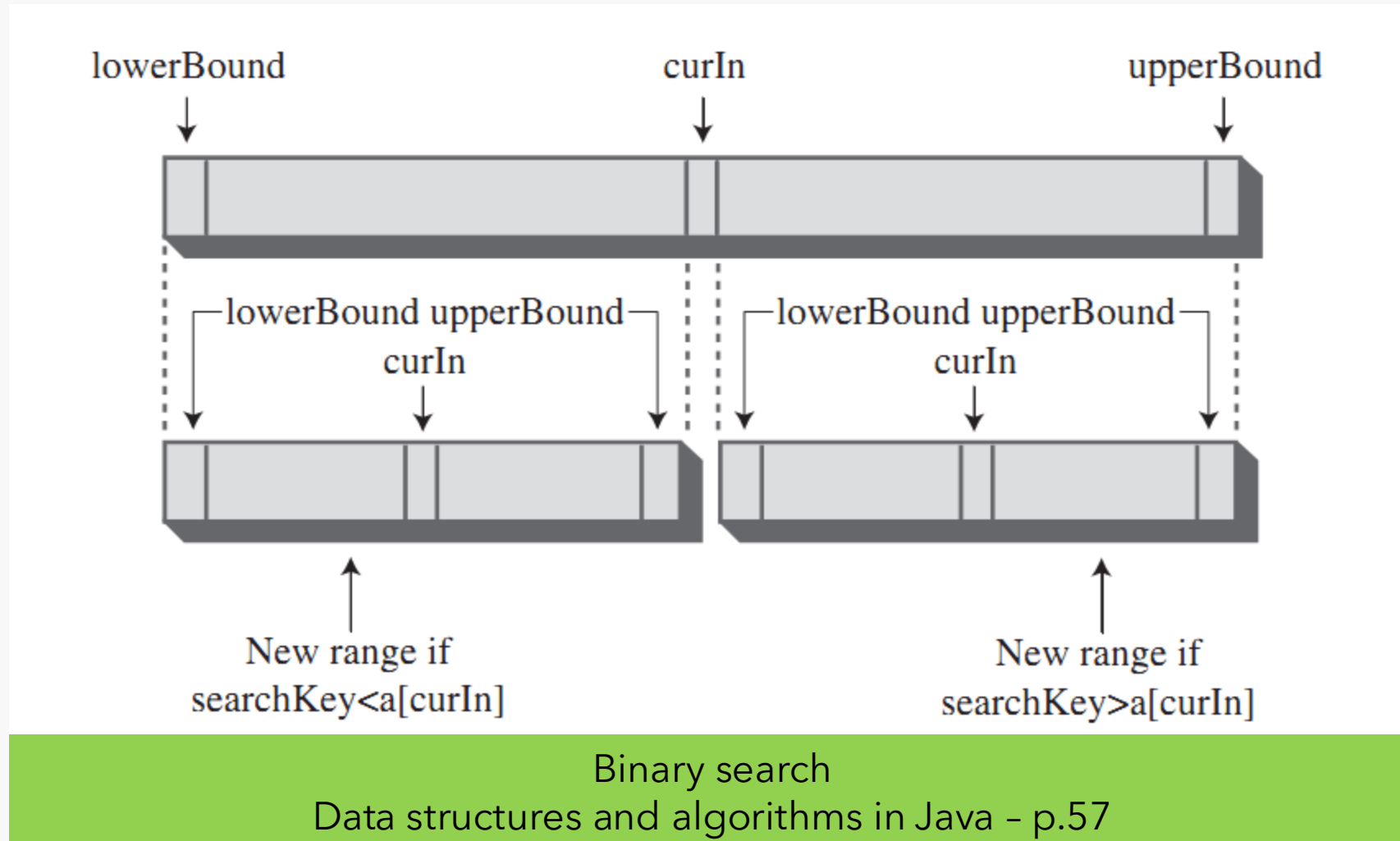


Larger

Binary searching technique

Step #	Number guessed	Result	Range of possible value
0			0-100

Algorithm

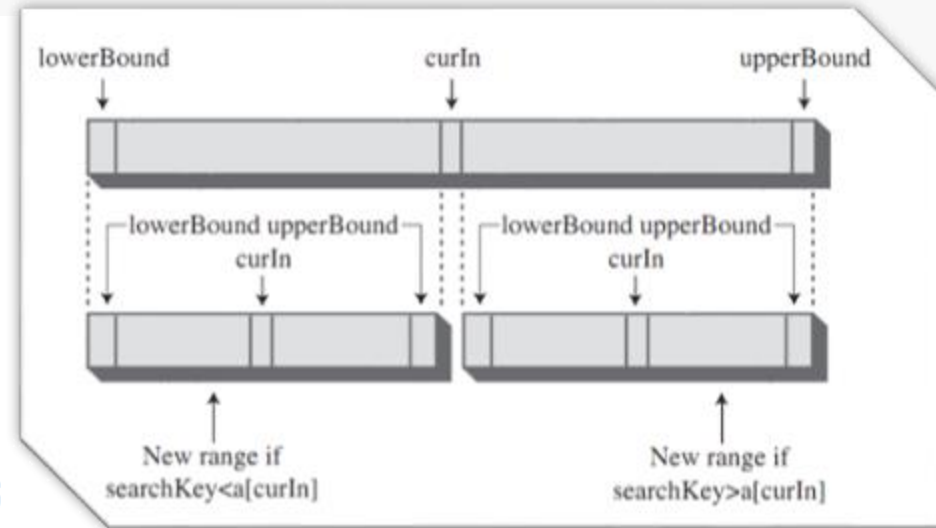


```

public int find(long searchKey)
{
    int lowerBound = 0;
    int upperBound = nElems-1;
    int curIn;

    while(true)
    {
        curIn = (lowerBound + upperBound) / 2;
        if(a[curIn]==searchKey)
            return curIn;           // found it
        else if(lowerBound > upperBound)
            return nElems;          // can't find it
        else
            // divide range
            {
                if(a[curIn] < searchKey)
                    lowerBound = curIn + 1; // it's in upper half
                else
                    upperBound = curIn - 1; // it's in lower half
            } // end else divide range
    } // end while
} // end find()

```



Question?

+Modify the Binary search algorithm for a descending array?

Advantage of ~~ordered arrays~~

- + Searching time : Good
- + Inserting time : Not good
- + Deleting time : Not good

Search OK

+ → Useful when

- + Searches are frequent
- + Insertions and deletions are not

Logarithm

+ Binary search

→ $\text{Log}_2(N)$

Range	Comparisons needed
10	4
100	7
1,000	10
10,000	14
100,000	17
1,000,000	20
10,000,000	24
100,000,000	27
1,000,000,000	30

Must known

2^i	n	$\log_2 n$		2^i	n	$\log_2 n$
2^0	1	0		2^6	64	6
2^1	2	1		2^7	128	7
2^2	4	2		2^8	256	8
2^3	8	3		2^9	512	9
2^4	16	4		2^{10}	1024	10
2^5	32	5		2^{11}	2048	11

Storing objects

We need to

- + Store a collections of Students
- + Search student by Student name
- + Insert a new student, delete a student

In class work:

- + Read the sample code in p.65-69
 - + The **Person** Class
 - + The **classDataArray.java** Program

Big O notation

- + To measure the EFFICIENCY of algorithms
- + Some notions
 - + Constant
 - + Proportional to N
 - + Proportional to log(N)
- + Big O relationships between time and number of items

Algorithms

- + Are sequences of instructions
- + To ~~solve~~ a problem
- + In a finite ~~amount~~ of time

Analysis of Algorithms

+ What are requirements for a good algorithm?

+ Precision:

+ Proved by mathematics

+ Implementation and test

+ Simple

+ Effectiveness:

+ Run time duration (time complexity)

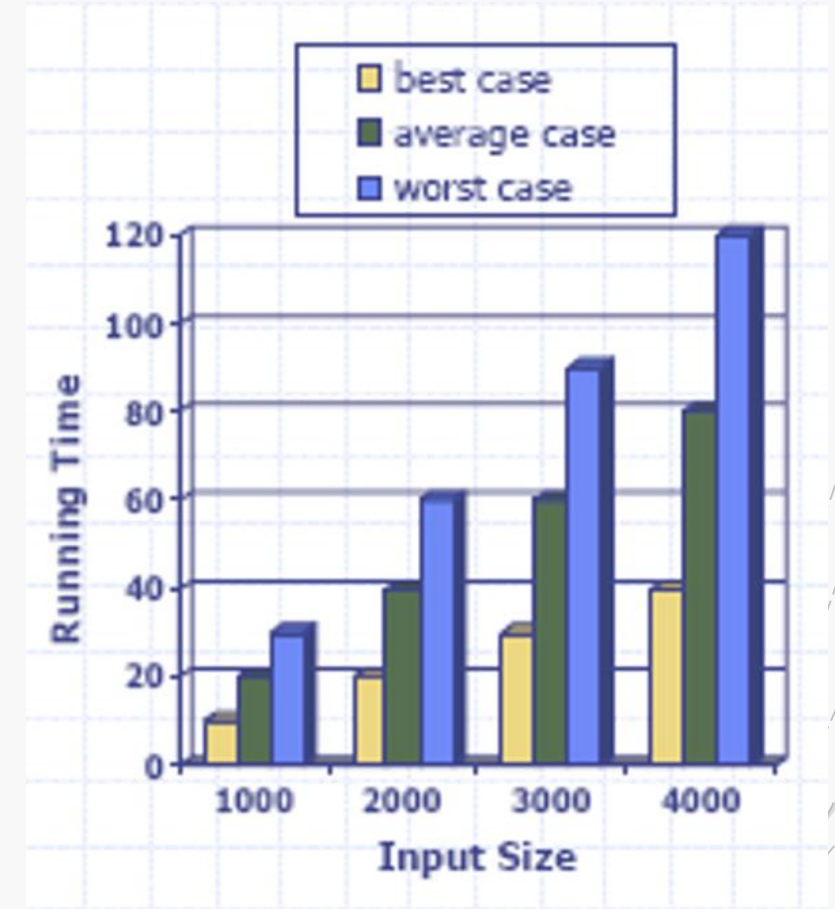
+ Memory space (space complexity)

What is a computational complexity?

- + The same problem can be solved with various algorithms that differ in efficiencies.
- + The computational complexity (or simply speaking, complexity) of an algorithm is a measure of how “complex” the algorithm is.
 - + How difficult is to compute something that we know to be computable?
 - + What resources (time, space, machines, ...) will it take to get a result?
- + We also often talk instead about how “efficient” the algorithm is
- + Measuring efficiency (or complexity) allows us to compare one algorithm to another
- + Here we'll focus on one complexity measure: **the computation time** of an algorithm.

Running time

- + Most algorithms transform input objects into output objects
- + The running time of an algorithm typically grows with the input size
- + Average case time is often difficult to determine
- + We focus on the **worst-case** running time



Time complexity of an algorithm

+ Run time duration of a program depend on

+ Size of data input

+ Computing system (platform: operation system, speed of CPU, type of statement...)

+ Programming languages

+ State of data

➔ It is necessary to evaluate the run time of a program such that it does not depend on computing system and programming languages.

Time complexity of an algorithm

- + Time complexity = the number of operations given an input size.
 - + What is meant by "number of operations"?
 - + What is meant by "size"?
- + The number of operations performed = **function of the input size n .**
- + What if there are many different inputs of size n ?
 - + Worst case
 - + Best case
 - + Average case
- + "*number of operations*" = "*running time*"?

Big-Oh Notation

+ Given function $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that

$$f(n) \leq c * g(n) \text{ for all } n \geq n_0$$

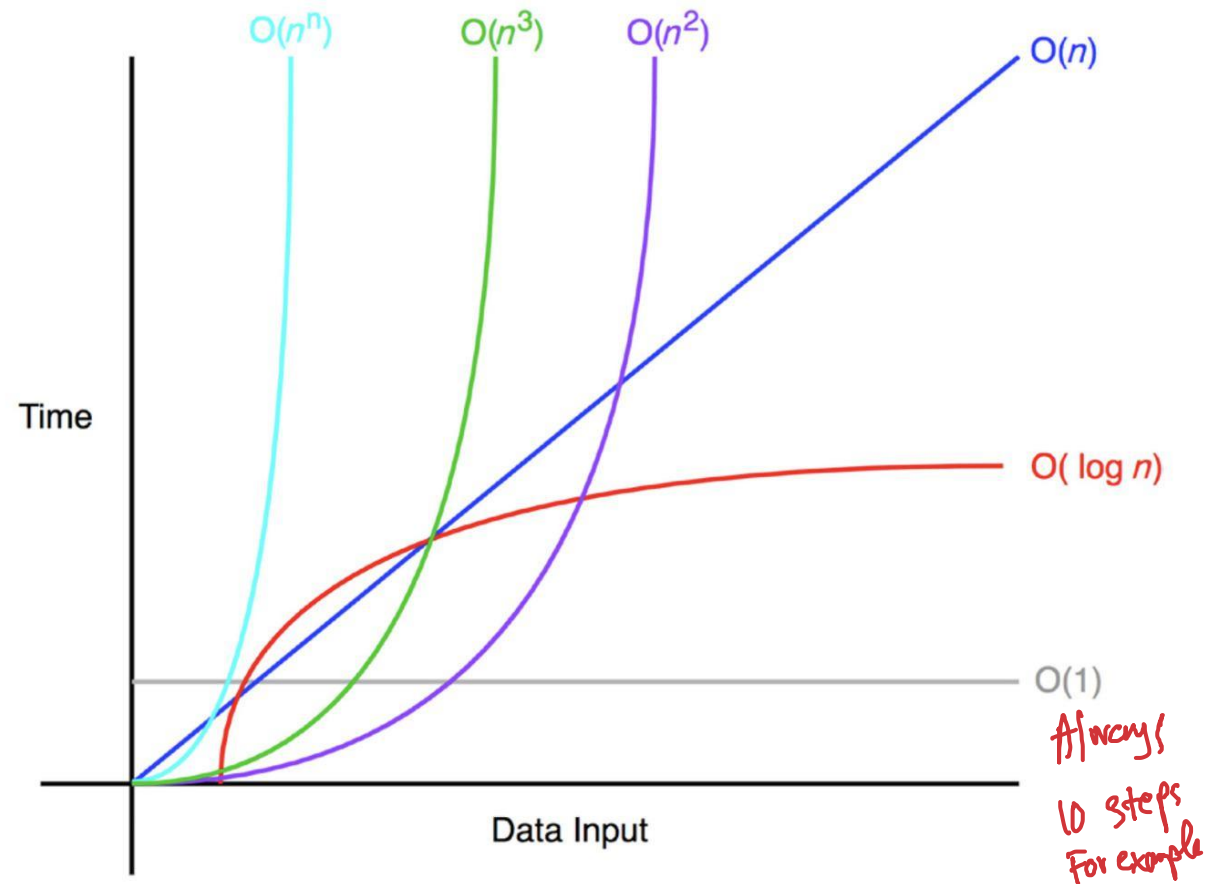
+ **Example:** $2n + 10$ is $O(n)$

$$2n + 10 \leq cn$$

$$(c - 2)n \geq 10$$

$$n \geq 10(c - 2)$$

→ Pick $c = 3$ and $n_0 = 10$



Big-Oh Notation

Midtgran

+ **Another example:**

+ The function n^2 is not $O(n)$

$$n^2 \leq cn$$

$$n \leq c$$

→ Cannot find a constant c and n_0 to satisfy this equation

$O(1)$ – constant

+ The time needed by the algorithm does not depend on the number of items

+ Example

+ Insertion in an unordered array

+ Any others ?

↳ Delete items at the end of Array

$O(N)$ – Proportional to N

- + Linear search of K items in an array of N items

On average $T = K * N / 2$

- + Average linear search times are proportional to size of array.

- + For an array of N' items

If $N' = 2 * N$

Then $T' = 2 * T$

$O(\log N)$ - Proportional to $\log(N)$

+For binary search:

$$T = K * \log_2(N)$$

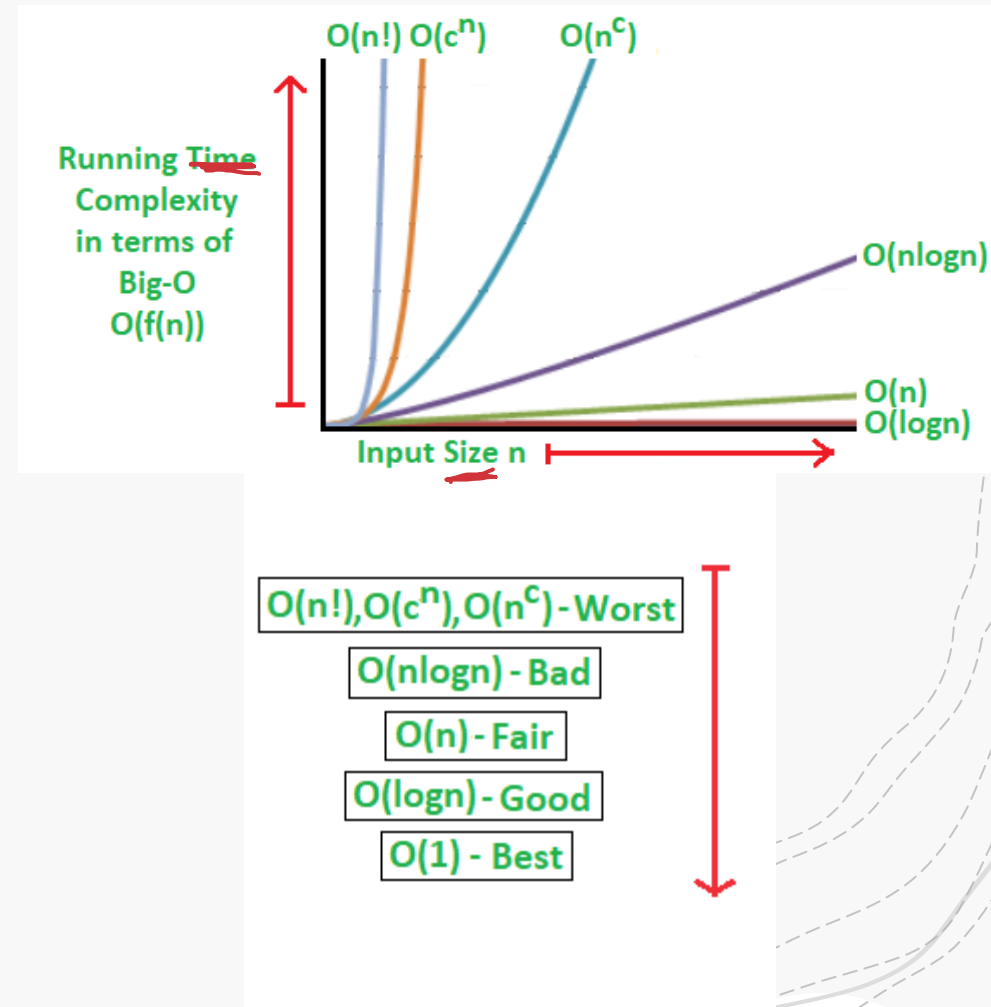
+We can say : $T = K * \log(N)$

+*any logarithm is related to any other logarithm by a constant (3.322 to go from base 2 to base 10*

+*we can add the difference between \log_2 and \log into K*

Some common growth orders of functions

constant	$O(1)$
logarithmic	$O(\log n)$
linear	$O(n)$
$n \log n$	$O(n \log n)$
quadratic	$O(n^2)$
polynomial	$O(n^b)$
exponential	$O(b^n)$
factorial	$O(n!)$




Summary

- + Arrays in Java are objects created with new operator
- + Unordered arrays offer *insert*
 - + fast insertion
 - + slow searching and deletion
- + Binary search can be applied to an ordered array
- + Big O notation provides a convenient way to compare the speed of algorithms
- + An algorithm that runs in $O(1)$ is the best, $O(\log N)$ is good, $O(N)$ is fair and $O(N^2)$ is bad

Further reading

- + Lafore, R. (2017). Data Structures and Algorithms in Java. United Kingdom: Pearson Education.
- + Chapter 2: Arrays (p.33)



Vietnam National University of HCMC
International University
School of Computer Science and Engineering



THANK YOU

Dr Vi Chi Thanh - vcthanh@hcmiu.edu.vn

<https://vichithanh.github.io>



SCAN ME