COMP1819
Algorithms and Data Structures

Lecture 12: Exam revision

Dr. Tuan Vuong

06/04/2021

Content

- Exam general info.
- Sample question
- Lectures revision
- Practice questions

COMP1819 Assessment

Coursework

- Programming assignment including a report
  - Worth 50% of your COMP1819 marks

Exam

- Multiple choice, open book
- Cover both Lectures and Labs materials
- Worth 50% of your COMP1819 marks
- Date & Avenue: 6/5/2021 9.30am, Online
- Check your timetable regularly

the UNIVERSITY of GREENWICH

EXAMINATION PAPER: ACADEMIC SESSION 2019/2020

Campus	Maritime Greenwich
Faculty	Faculty of Liberal Arts and Sciences
School	School of Computing and Mathematical Sciences
TITLE OF PAPER	Algorithms and Data Structures
COURSE CODE	COMP1819
Date and Time	May 2020 - 90 minutes

Answer **ALL** questions

This is a multi-choice, open-book examination. You may access the internet but you may not communicate in any way with another person (including by electronic means).

Sample question

You can explain (choose 2)?

A.

B.

C.

D.

E.

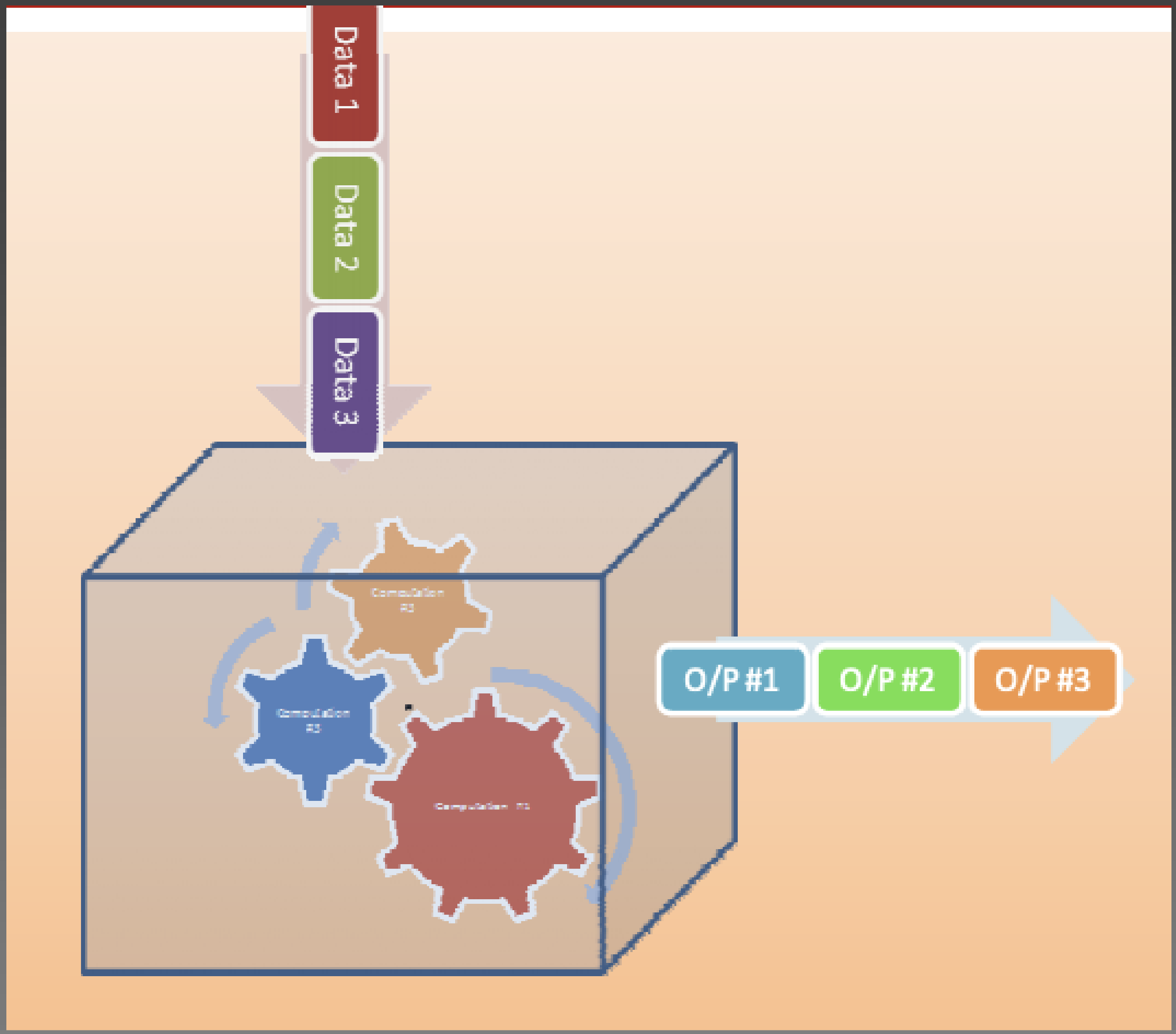
[5 marks]

So you are asked to choose 2 out of 5 choices.

Lecture 01: Introduction to Algorithms and Data Structures (ADS)

Content

- Why study ADS?
- Algorithms
- Pseudocode
- Data structures



Good programs: correct, finite (meaning?), terminate, unambiguous. We should focus on solving problems efficiently.

Practice question

6. .... is not the component of data structure.

A) Operations

B) Storage Structures

C) Algorithms

D) None of above

D is the answer here.

Lecture 02: Analysis of Algorithm

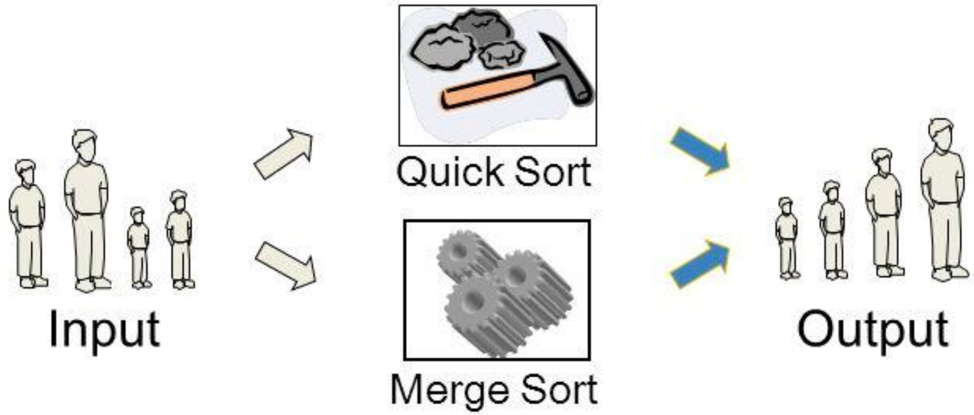
Content

- Lab 01 Discussion
- What is Algorithm Analysis?
- BigO
- Reinforcement

Analysis of Algorithms

An **algorithm** is a step-by-step procedure for solving a problem in a finite amount of time.

How to evaluate algorithms?



- Which one is better?
- What are the criteria?

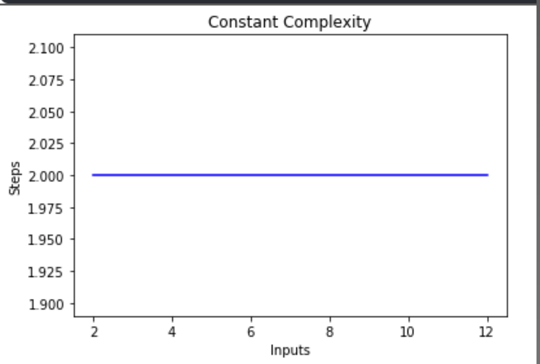
Good programs: correct, finite (meaning?), terminate, unambiguous. We should focus on solving problems efficiently.

n	Constant O(1)	Logarithmic O(log n)	Linear O(n)	Linear Logarithmic O(n log n)	Quadractic $O(n^2)$	Cubic $O(n^3)$
1	1	1	1	1	1	1
2	1	1	2	2	4	8
4	1	2	4	8	16	64
8	1	3	8	24	64	512
16	1	4	16	64	256	4,096
1,024	1	10	1,024	10,240	1,048,576	1,073,741,824

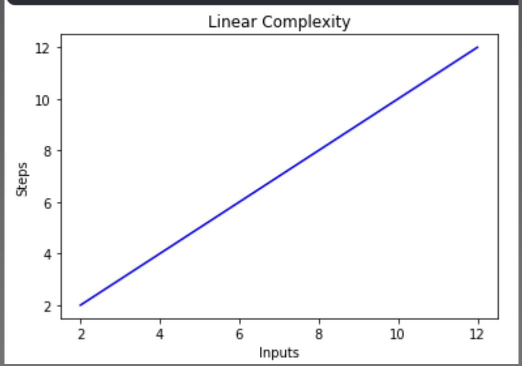
Big O Notation: the order of magnitude for a useful approximation to the actual steps in the computation.

Algorithm Analysis – Big O notation (Complexity in term of n – input size)

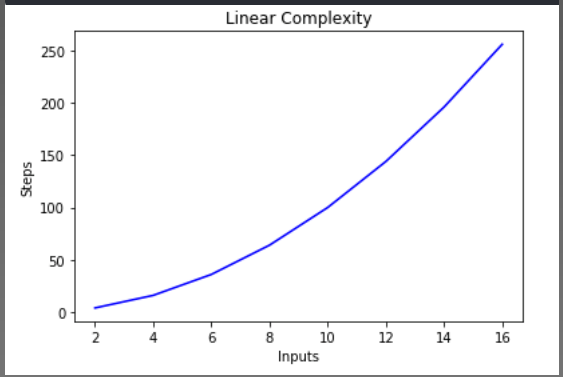
```python
def constant_algo(items):
    result = items[0] * items[0]
    print ()

constant_algo([4, 5, 6, 8])
```




```python
def linear_algo(items):
    for item in items:
        print(item)

linear_algo([4, 5, 6, 8])
```



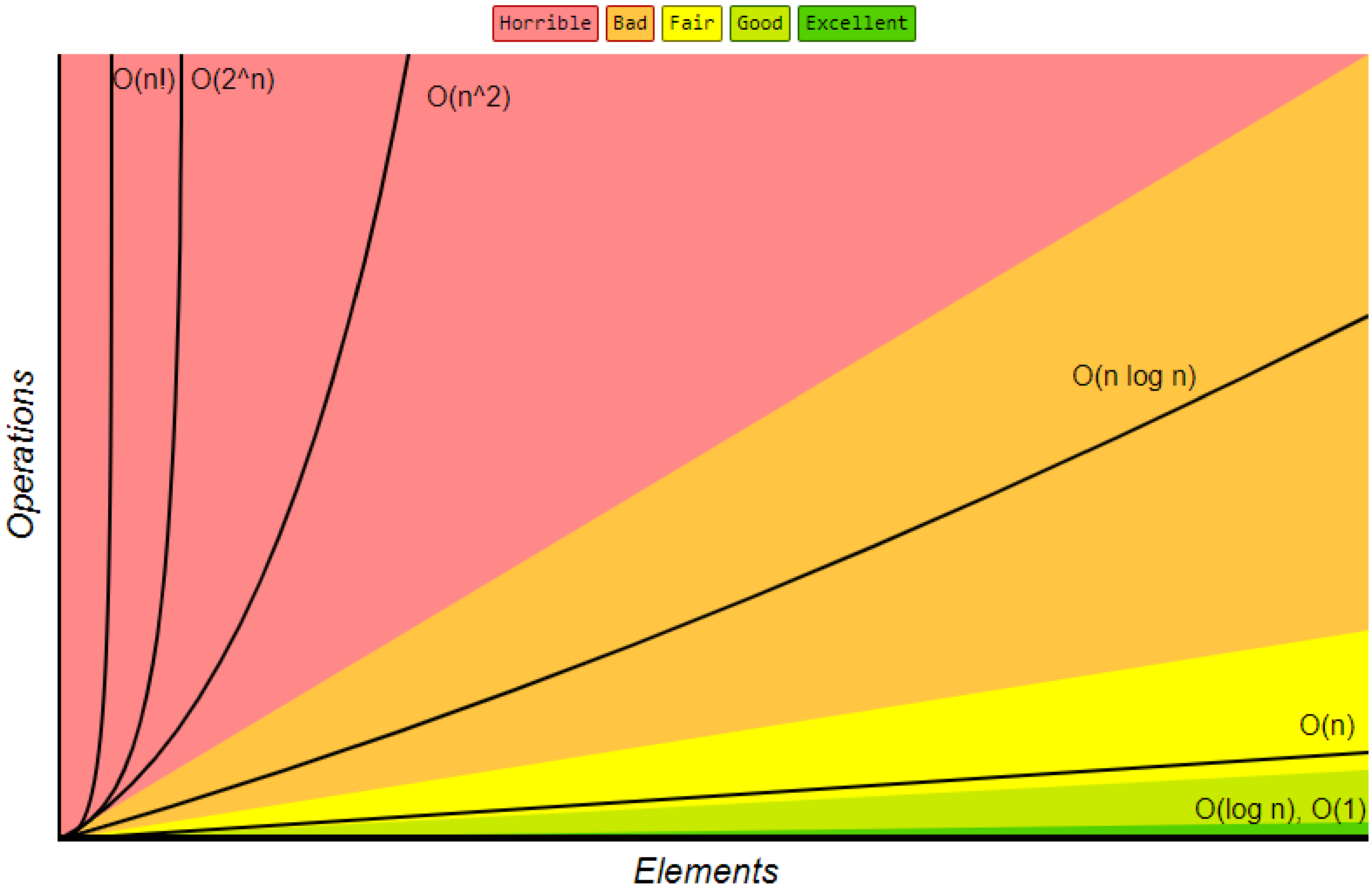

```python
def quadratic_algo(items):
    for item in items:
        for item2 in items:
            print(item, ' ' ,item)

quadratic_algo([4, 5, 6, 8])
```



Common rules:

- Constants can be omitted: $O(100) \to O(1)$, $O(3n) \to O(n)$, $O(7n^2) \to O(n^2)$
- Smaller terms can be omitted: $O(200+3n) \to O(n)$, $O(4n+7n^2) \to O(n^2)$

Big-O Complexity Chart



Practice question

What is the Big-O performance of the following?

```
for i in range(n):
    for j in range(n):
        k = 2 + 2
```

```
i = n
while i > 0:
    k = 2 + 2
    i = i // 2
```

```
for i in range(n):
    for j in range(n):
        for k in range(n):
            k = 2 + 2
```

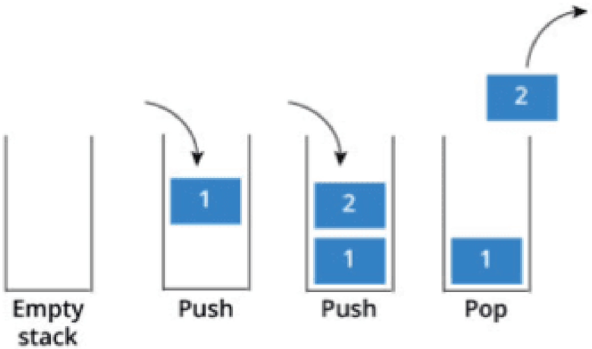
```
for i in range(n):
    k = 2 + 2
for j in range(n):
    k = 2 + 2
for k in range(n):
    k = 2 + 2
```

Lecture 03: Stacks, and Queues

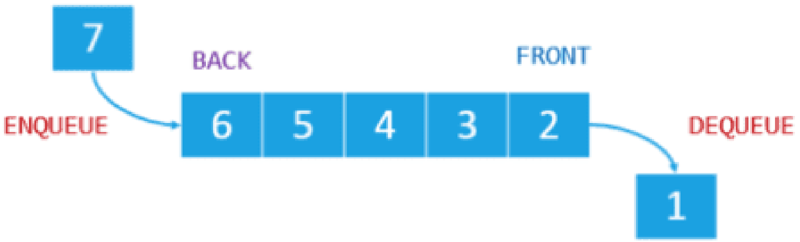
Content

- Guest speaker
- Lab 02 Discussion (Prime numbers)
- Stacks
- Queues
- Deques
- Reinforcement

Data Structure Basics



Stack



Queue

Practice question

4. Stack is also called as

A) Last in first out

B) First in last out

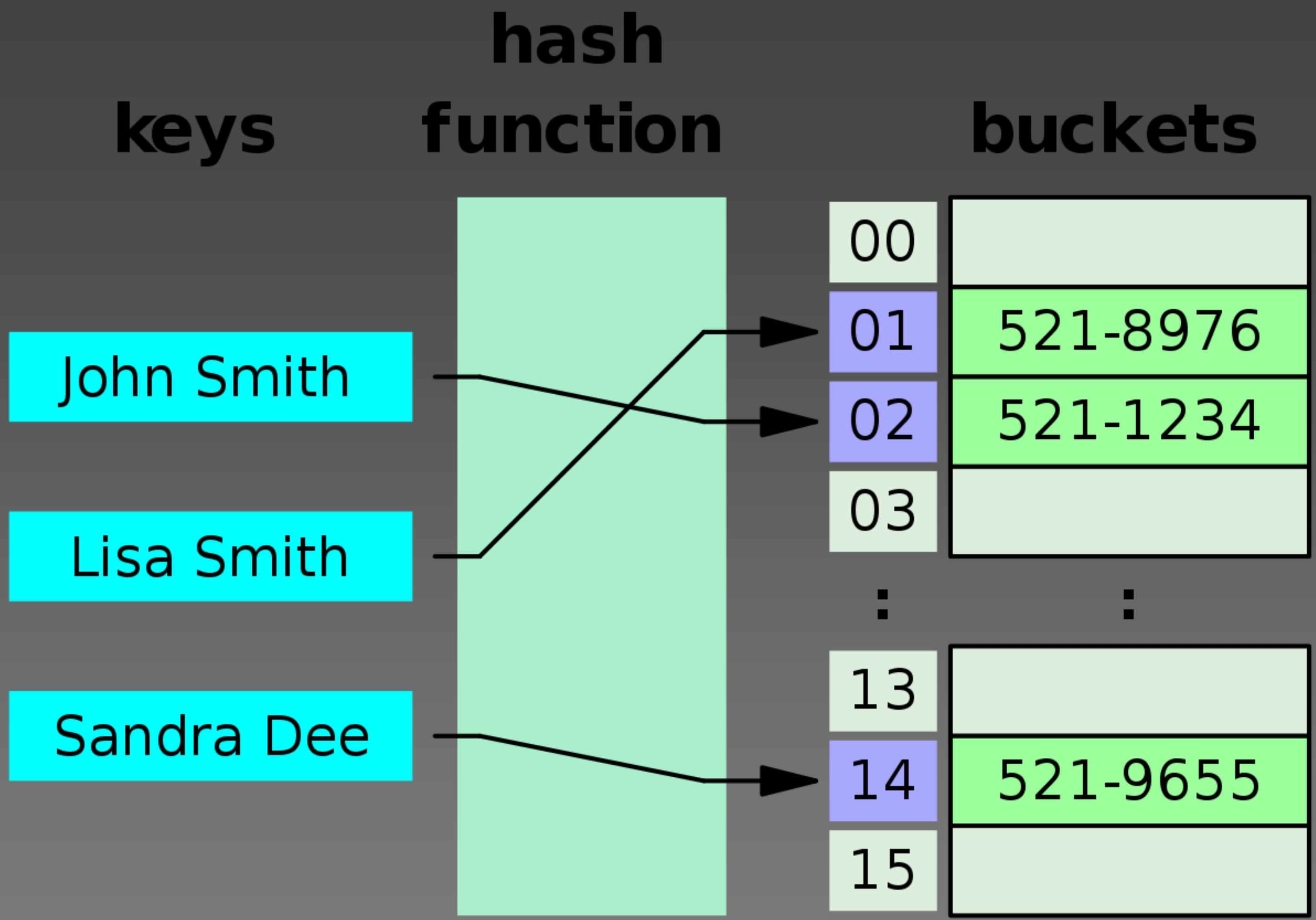
C) Last in last out

D) First in first out

Lecture 04: Searching – Linear and Binary

Content

- Lab 03
- Linear Search
- Binary Search
- Hashing
- Reinforcement



Search algorithm solves the search problem, namely, to retrieve information stored within some data structure, or calculated in the search space of a problem domain, either with discrete or continuous values

Practice question

The average number of key comparisons required for a successful search for sequential search on items is

- A n/2
- B (n-1)/2
- C (n+1)/2
- D None of these

Lecture 05: Sorting – Bubble, Selection and Insertion (for a small collection)

Content

- Review Lab 04
- Sorting: Bubble
- Selection
- Insertion
- Reinforcement

Time and Space Complexity:

SORTING ALGORITHM	TIME COMPLEXITY			SPACE COMPLEXITY
	Best Case	Average Case	Worst Case	Worst Case
Bubble Sort	**$O(N)$**	**$O(N^2)$**	**$O(N^2)$**	**$O(1)$**
Selection Sort	**$O(N^2)$**	**$O(N^2)$**	**$O(N^2)$**	**$O(1)$**
Insertion Sort	**$O(N)$**	**$O(N^2)$**	**$O(N^2)$**	**$O(1)$**

Practice question

What is the best time complexity of bubble sort?

A. N^2

B. NlogN

C. N

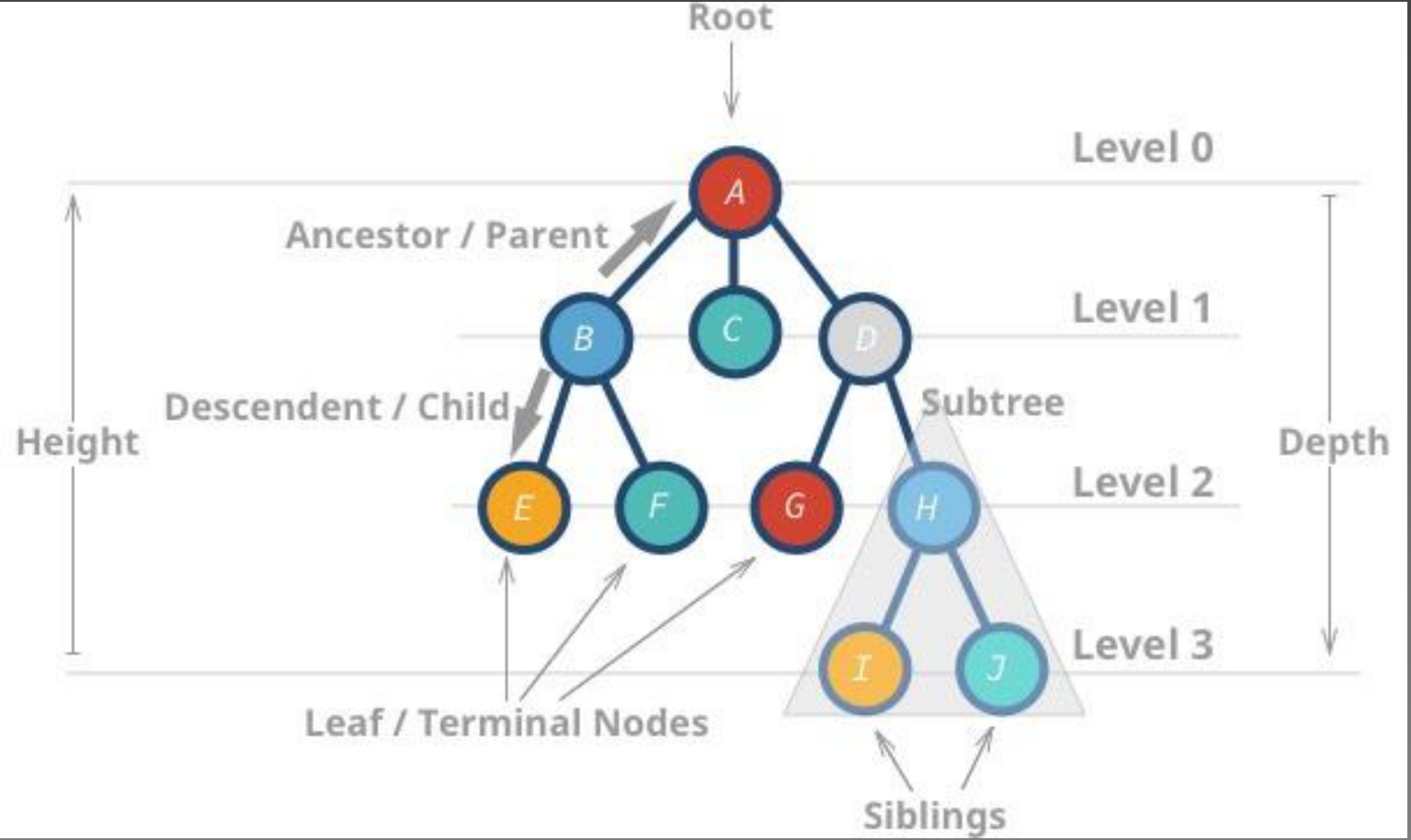
D. N(logN)^2

Lecture 06: Trees

Content

- Review Lab 05
- General Trees
- Binary Trees
- Implementing Trees
- Tree Traversal Algorithms
- Reinforcement

This one

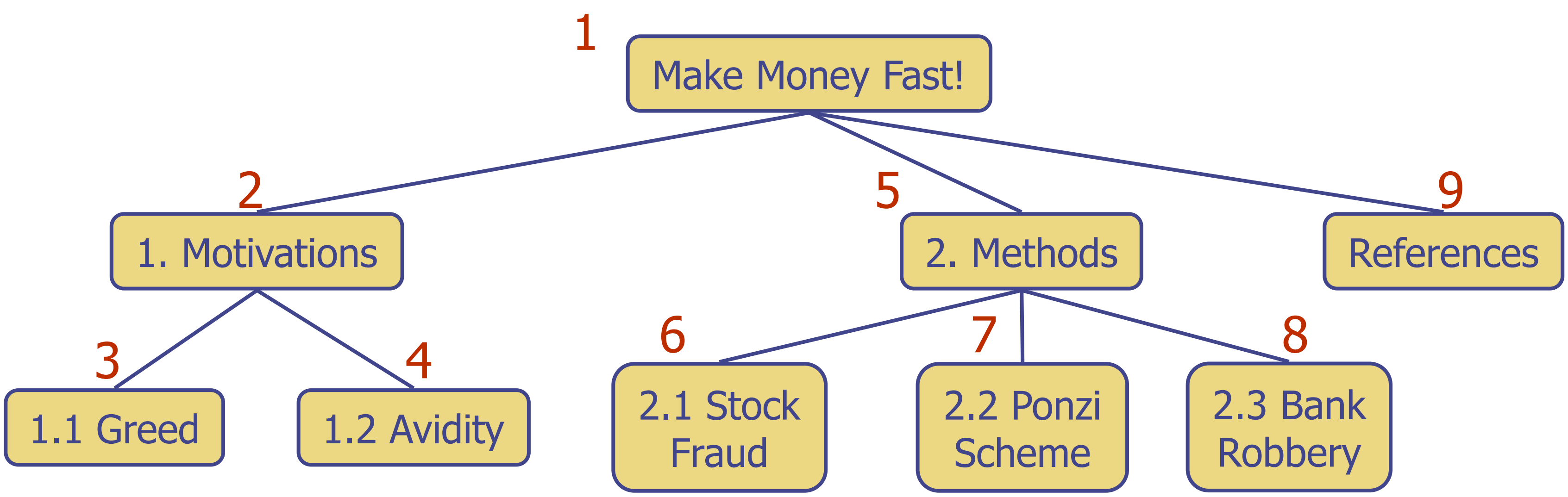


Root
Level 0
A
Ancestor / Parent
B
C
D
Level 1
Descendent / Child
Subtree
Height
Depth
E
F
G
H
Level 2
I
J
Level 3
Leaf / Terminal Nodes
Siblings

Preorder Traversal

- A traversal visits the nodes of a tree in a systematic manner
- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

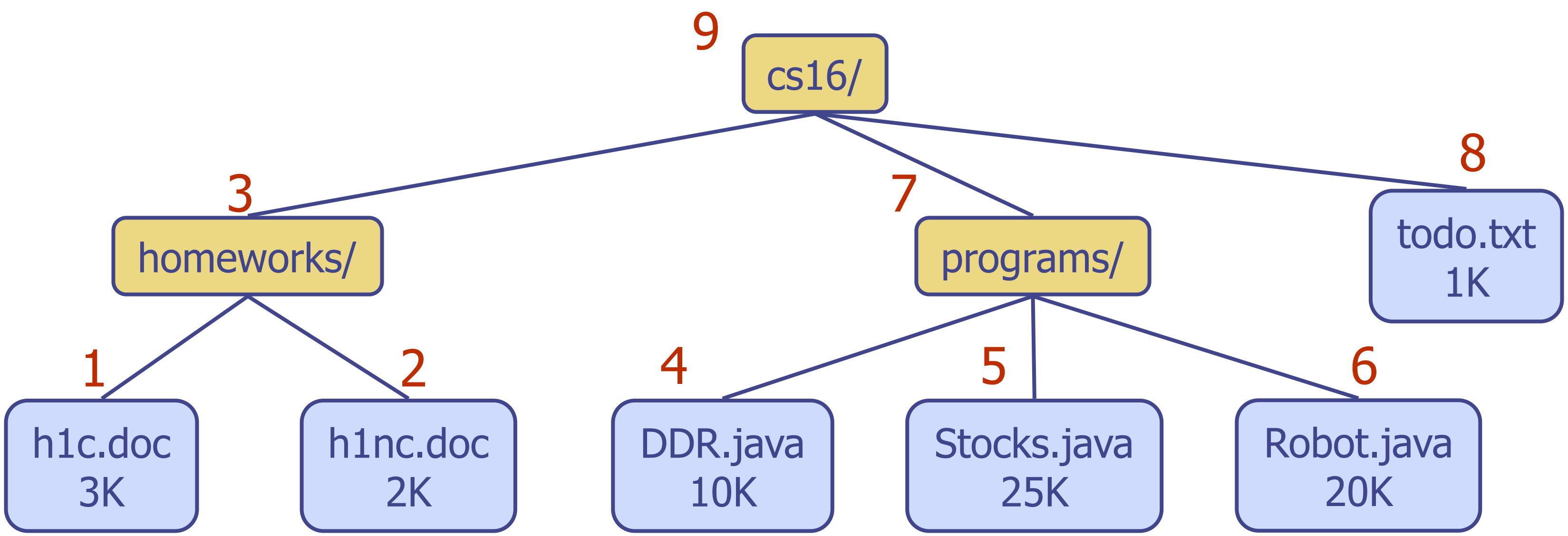
Algorithm *preOrder*(*v*)
    visit(*v*)
    for each child *w* of *v*
        preorder (*w*)



Postorder Traversal

- In a postorder traversal, a node is visited after its descendants
- Application: compute space used by files in a directory and its subdirectories

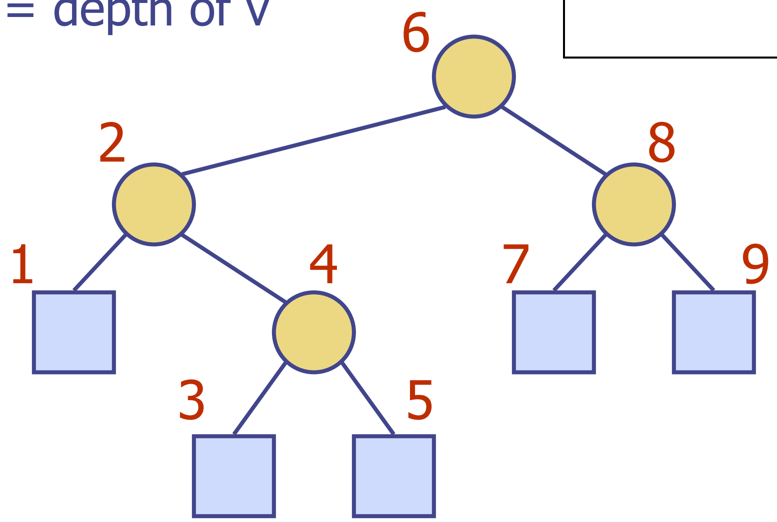
Algorithm *postOrder*(v)
    for each child w of v
        postOrder (w)
    visit(v)



Inorder Traversal

- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Application: draw a binary tree
  - x(v) = inorder rank of v
  - y(v) = depth of v

Algorithm *inOrder*(*v*)
    if *v* **has a left child**
        inOrder (*left* (*v*))
    visit(*v*)
    if *v* **has a right child**
        inOrder (*right* (*v*))



Practice question

Postorder traversal of a given binary search tree, T produces the following sequence of keys 10, 9, 23, 22, 27, 25, 15, 50, 95, 60, 40, 29 Which one of the following sequences of keys can be the result of an in-order traversal of the tree T? (GATE CS 2005)

A 9, 10, 15, 22, 23, 25, 27, 29, 40, 50, 60, 95

B 9, 10, 15, 22, 40, 50, 60, 95, 23, 25, 27, 29

C 29, 15, 9, 10, 25, 22, 23, 27, 40, 60, 50, 95

D 95, 50, 60, 40, 27, 23, 22, 25, 10, 9, 15, 29

Lecture 07: Maps

Content

- Lab 06 Walk-through
- Python dictionary
- Maps
- Map ADT, Implementation
- Application
- Reinforcement
- CW Q&A

Maps

- A **map** is a searchable collection of items that are key-value pairs
- The main operations of a map are for searching, inserting, and deleting items
- Multiple items with the same key are not allowed
- Applications:
  - address book
  - student-record database

Word count example

```
freq = {}
for piece in open(filename).read().lower().split():
  # only consider alphabetic characters within this piece
  word = ''.join(c for c in piece if c.isalpha())
  if word:                                    # require at least one alphabetic character
    freq[word] = 1 + freq.get(word, 0)

max_word = ''
max_count = 0
for (w,c) in freq.items():    # (key, value) tuples represent (word, count)
  if c > max_count:
    max_word = w
    max_count = c
print('The most frequent word is', max_word)
print('Its number of occurrences is', max_count)
```

How to order the frequency in order?

Practice question

2. What will be the output of the following Python code s

```python
d = {"john":40, "peter":45}
```

a) “john”, 40, 45, and “peter”

b) “john” and “peter”

c) 40 and 45

d) d = (40:”john”, 45:”peter”)

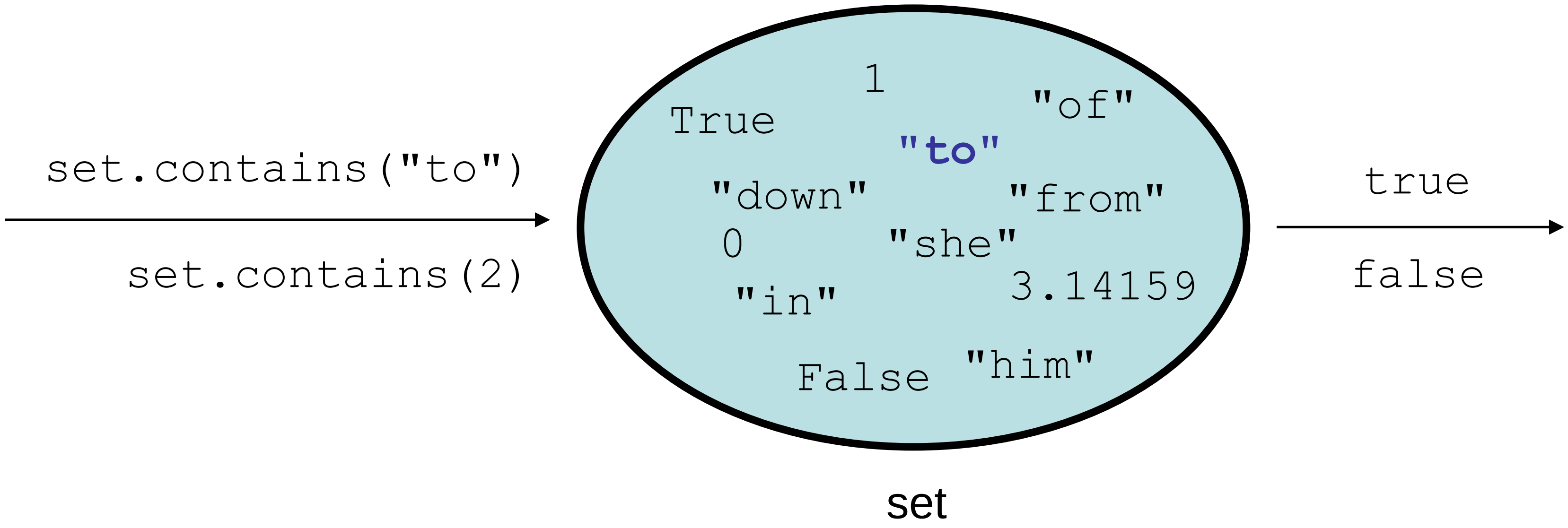
Lecture 08: Sets

Content

- Lab 07 Walk-through
- Python Sets
- Sets ADT
- Revisions Lecture 01-07
- CW Q&A

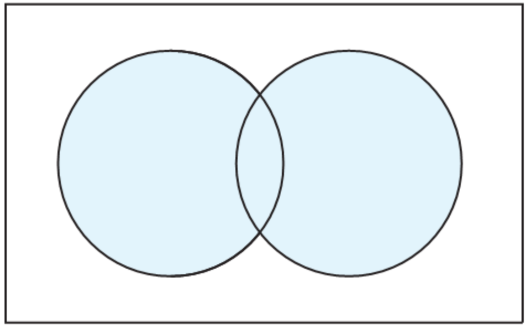
Definitions

- A **set** is an unordered collection of elements, without duplicates that typically supports efficient membership tests.



Practice question

A U B Union



main.py saved

```python
# Python3 program for union() function

set1 = {2, 4, 5, 6}
set2 = {4, 6, 7, 8}
set3 = {7, 8, 9, 10}

# union of two sets
print("set1 U set2 : ", set1.union(set2))

# union of three sets
print("set1 U set2 U set3 :", set1.union(set2, set3))

```

What is the output?

```
set1 U set2 :  {2, 4, 5, 6, 7, 8}
set1 U set2 U set3 : {2, 4, 5, 6, 7, 8, 9, 10}
```

Lecture 09: Recursive algorithms and analysis

Content

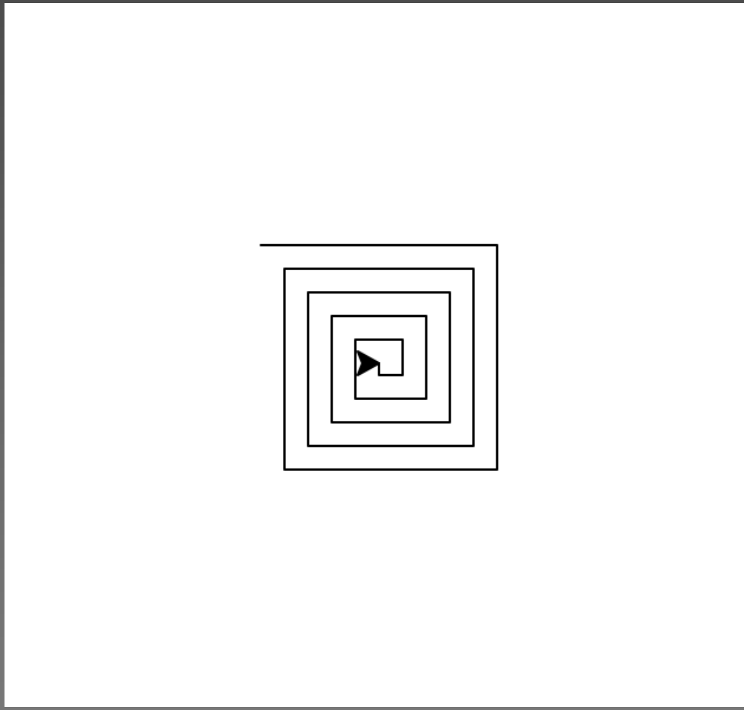
- Lab 08 Walk-through/CW Q&A
- Recursive algorithms & Analysis
- Linear and Binary recursion
- Visualising recursion
- Reinforcement

Three laws of recursion

- A recursive algorithm must
  - Have a base case.
  - Change its state and move forward the base case.
  - Call itself, recursively.

Drawing Spiral

```
import turtle

myTurtle = turtle.Turtle()
myWin = turtle.Screen()

def drawSpiral(myTurtle, lineLen):
    if lineLen > 0:
        myTurtle.forward(lineLen)
        myTurtle.right(90)
        drawSpiral(myTurtle,lineLen-5)

drawSpiral(myTurtle,100)
myWin.exitonclick()
```

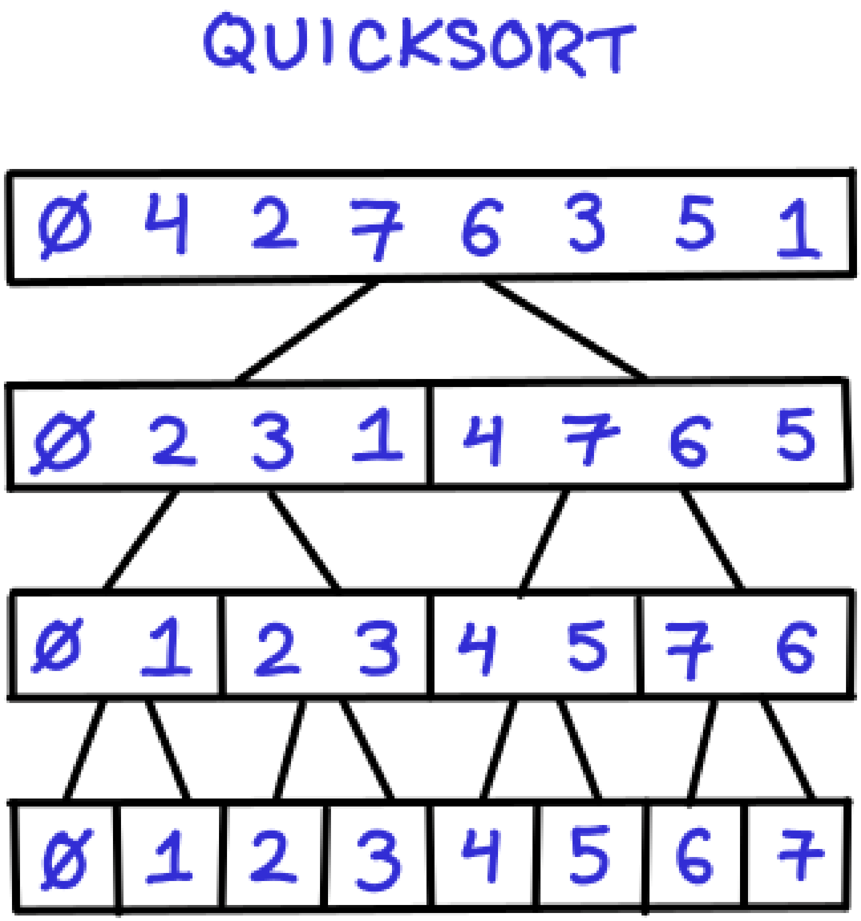


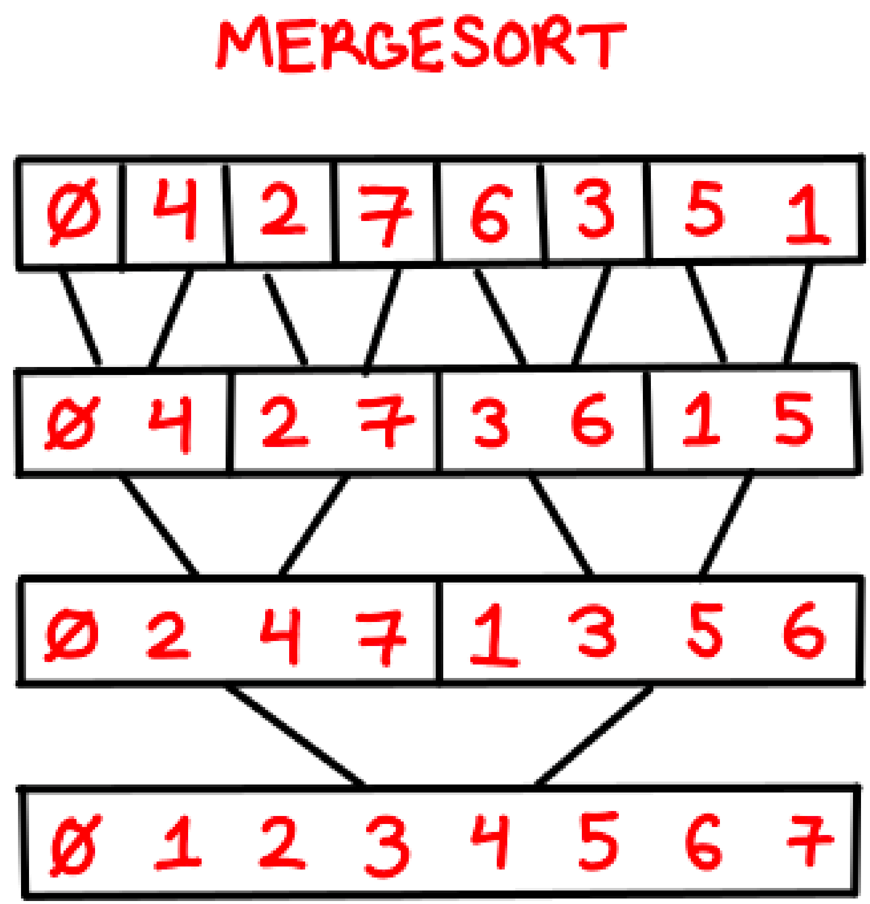
Lecture 10: MergeSort & QuickSort

Content

- Lab 09 Walk-through/CW Q&A
- Merge Sort
- Quick Sort
- Reinforcement

Today





https://hackernoon.com/

Divide-and-Conquer

- Divide-and conquer is a general algorithm design paradigm:
  - Divide: divide the input data S in two disjoint subsets S_1 and S_2
  - Recur: solve the subproblems associated with S_1 and S_2
  - Conquer: combine the solutions for S_1 and S_2 into a solution for S
- The base case for the recursion are subproblems of size 0 or 1
- Merge-sort is a sorting algorithm based on the divide-and-conquer paradigm

Merge-Sort

- Merge-sort on an input sequence S with n elements consists of three steps:
  - Divide: partition S into two sequences S_1 and S_2 of about $n/2$ elements each
  - Recur: recursively sort S_1 and S_2
  - Conquer: merge S_1 and S_2 into a unique sorted sequence

Algorithm ***mergeSort***(S)

Input sequence S with n elements

Output sequence S sorted according to C

if $S.size() > 1$

$(S_1, S_2) \leftarrow partition(S, n/2)$

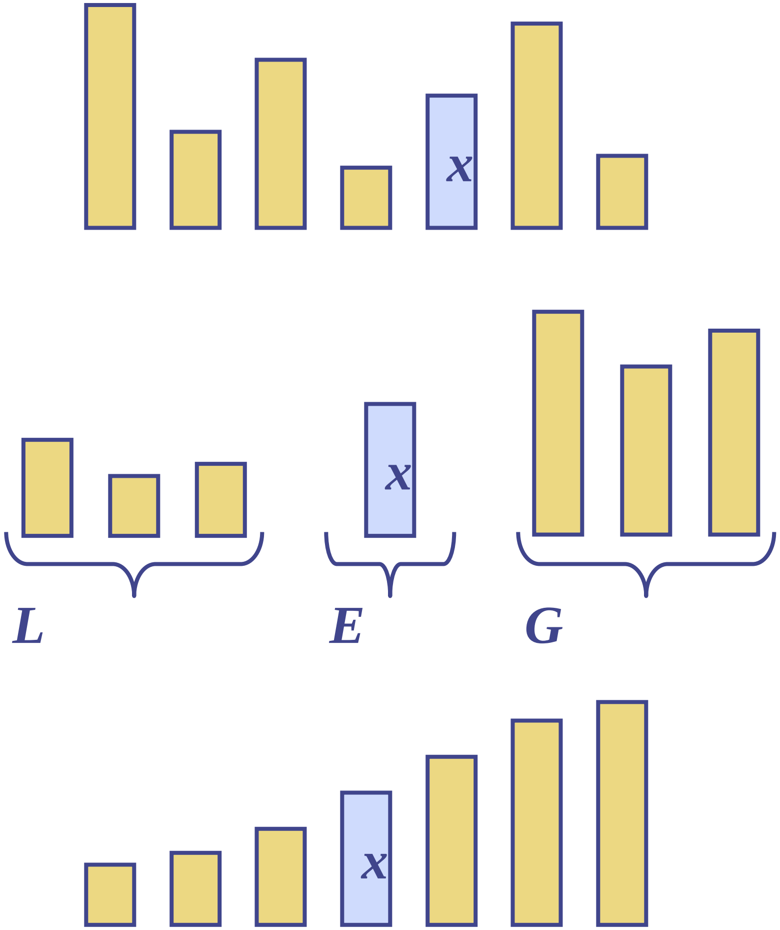
$mergeSort(S_1)$ -> S1 already sorted

$mergeSort(S_2)$ -> S2 already sorted

$S \leftarrow merge(S_1, S_2)$

Quick-Sort

- Quick-sort is a randomized sorting algorithm based on the divide-and-conquer paradigm:
  - Divide: pick a random element x (called pivot) and partition S into
    - L elements less than x
    - E elements equal x
    - G elements greater than x
  - Recur: sort L and G
  - Conquer: join L, E and G



In-Place Quick-Sort

- Quick-sort can be implemented to run in-place
- In the partition step, we use replace operations to rearrange the elements of the input sequence such that
  - the elements less than the pivot have rank less than h
  - the elements equal to the pivot have rank between h and k
  - the elements greater than the pivot have rank greater than k
- The recursive calls consider
  - elements with rank less than h
  - elements with rank greater than k

Algorithm *inPlaceQuickSort*(S, l, r)
    Input sequence S, ranks l and r
    Output sequence S with the elements of rank between l and r rearranged in increasing order
    if $l \geq r$
    return
$i \leftarrow$ a random integer between l and r
$x \leftarrow S.elemAtRank(i)$
$(h, k) \leftarrow inPlacePartition(x)$
$inPlaceQuickSort(S, l, h - 1)$
$inPlaceQuickSort(S, k + 1, r)$

Summary of Sorting Algorithms

Algorithm	Time	Notes
selection-sort	$O(n^2)$	▪ in-place ▪ slow (good for small inputs)
insertion-sort	$O(n^2)$	▪ in-place ▪ slow (good for small inputs)
quick-sort	$O(n \log n)$ expected	▪ in-place, randomized ▪ fastest (good for large inputs)
merge-sort	$O(n \log n)$	▪ sequential data access ▪ fast (good for huge inputs)

Practice question

Consider the list of characters: ['P','Y','T','H','O','N']. Show how this list is sorted using the following algorithms:

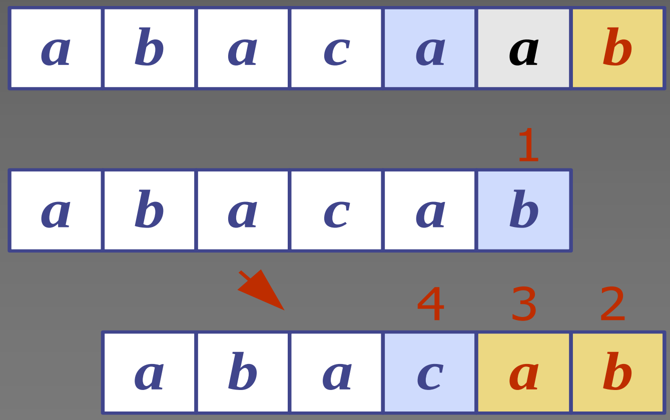
- bubble sort
- selection sort
- insertion sort
- merge sort
- quick sort

Lecture 11: Text Processing

Content

- Lab 10 Walk-through
- Text Processing/ Pattern-matching
- Exam General info.
- Reinforcement

Pattern Matching



Watch: https://www.youtube.com/watch?v=nK7SLhXcqRo

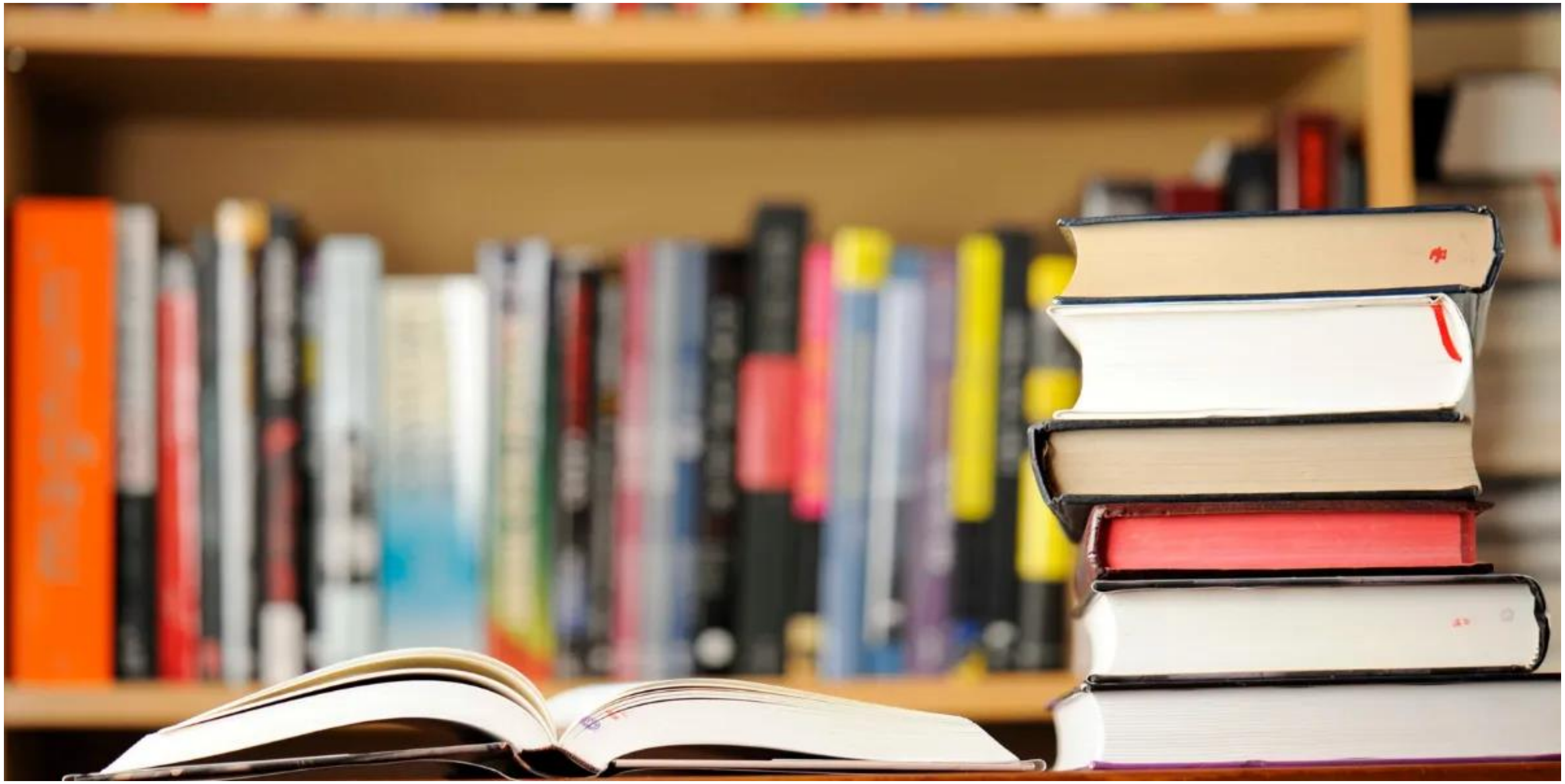
Practice question
(won’t be in the exam)

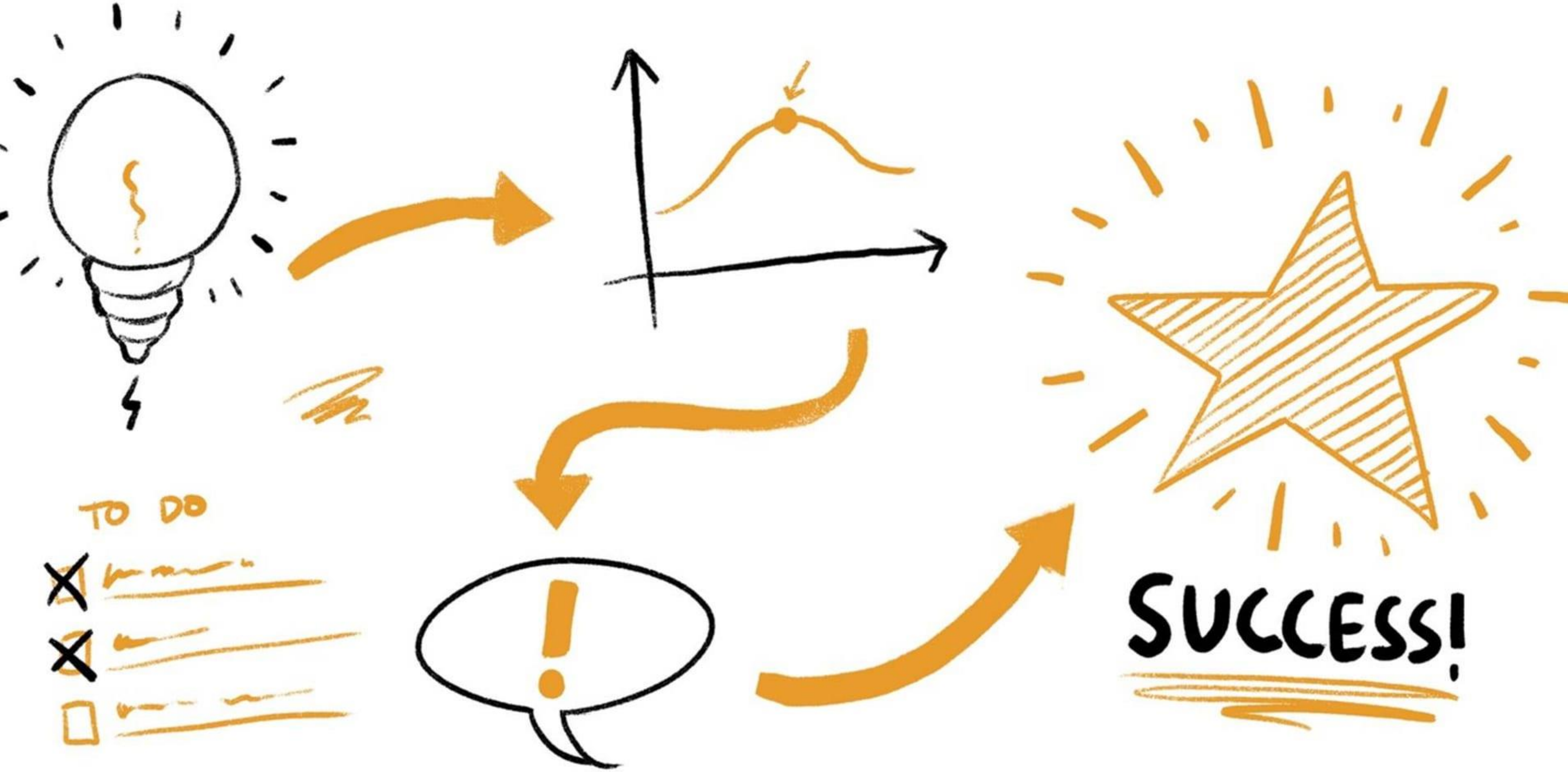
Quick overview

- Exam general information
- Lecture reviews
- Sample question and practice questions

Extra reading

- Labs materials





Success is not a matter of luck—it's an algorithm
(cnbc.com)