

# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Warming up

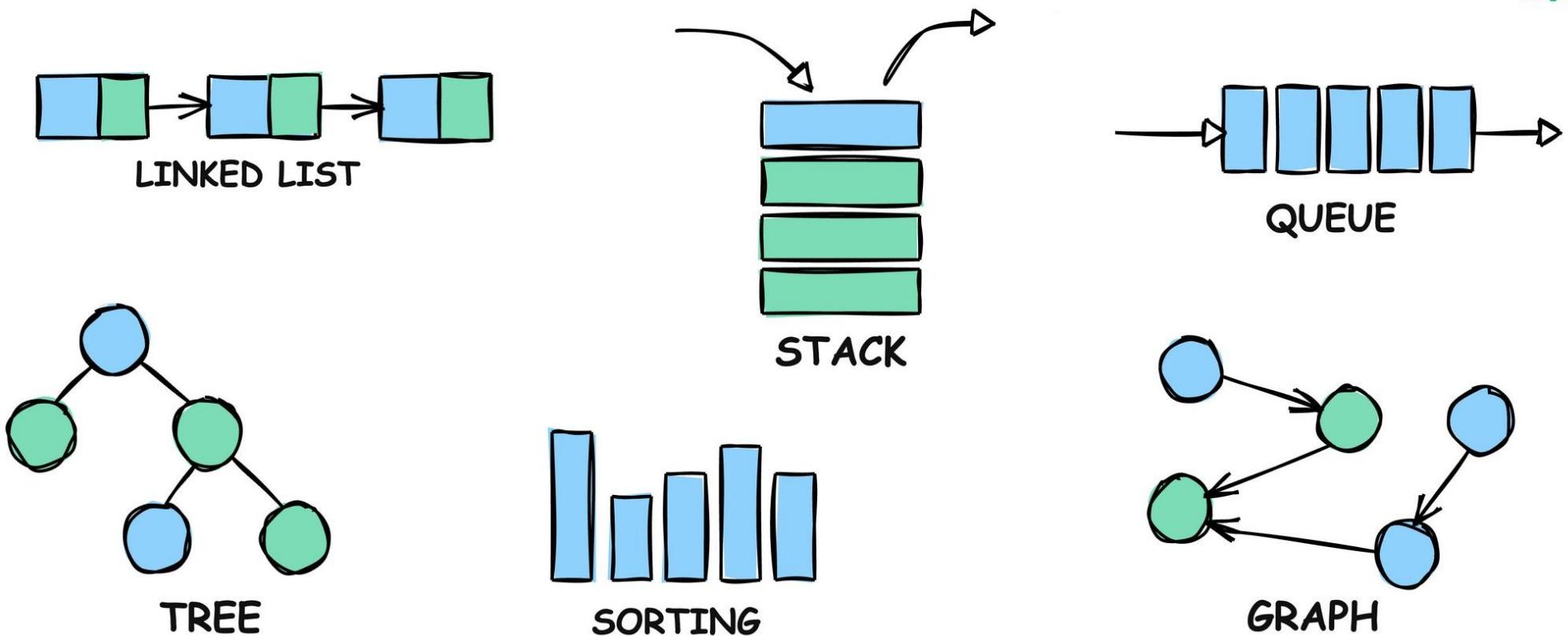
8:45-8:50

- Discuss how you are doing with the course
  - What do you expect from this course?
  - What prerequisite knowledge do you think you need in this course?
  - What do you want to have learned after this course?
- Discuss how you are doing with life?
  - How was your holiday?
  - What do you feel right now?
  - What are you looking forward to?
- Forgive yourself for coming slightly late
  - (and have the intention to come on time next lecture)

**Focus**

**30 sec**

- Breath in and out 3 times



# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# ADS is about problem solving!

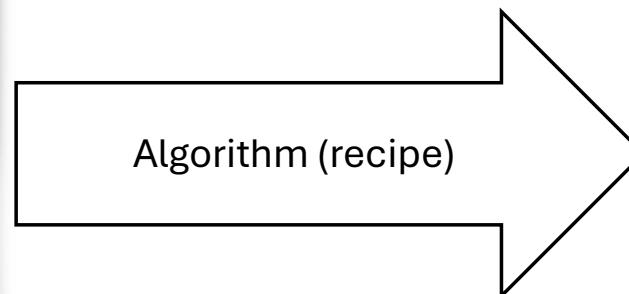


- Applied to Algorithms & Data Structures
- Extra focus on problem-solving
  - Specific problem-solving exercises
  - No solutions to implementation assignments
  - Exam exercises about ‘new’ algorithms
  - Exam exercise about the process

# Algorithm



Input data (ingredients)

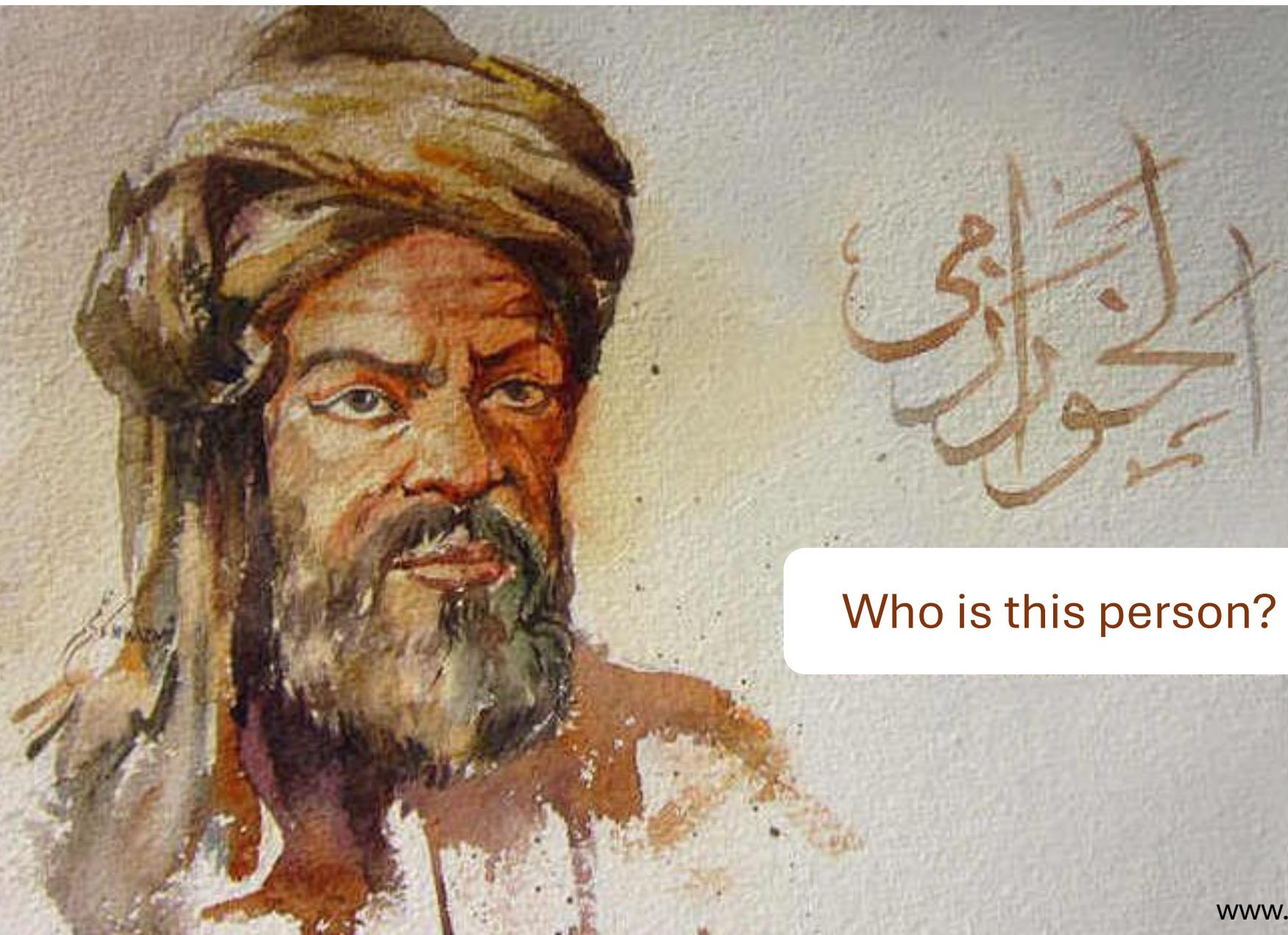


Algorithm  
Finite sequence of  
instructions to process the  
input data and produce the  
desired result.



Output (result)

Image credits: <https://www.thespruceeats.com/>, <https://www.apartmenttherapy.com/authors/joe>



Who is this person?



[www.amust.com.au/](http://www.amust.com.au/)

## Al-Khwārizmī

Persian polymath from 8th century.

Contributions to:  
mathematics, astronomy, geography.

Described arithmetic algorithms on  
decimal numbers, carried out on a dust  
board.

In the 12<sup>th</sup> century, Al-Khwārizmī's  
algorithms made it to Europe, replaced  
local abacus-based methods.

Al-Khwārizmī's latinized name,  
*Algorismus* (modern: **Algorithm**), turned  
into the name of a method used for  
computations.

# Properties of an algorithm

## Input

An algorithm has zero or more inputs, taken from a specified set.

## Output

An algorithm has one or more outputs, with a specified relation to the inputs.

## Definiteness

Each step of an algorithm must be precisely defined.

## Correctness

An algorithm must always give the right answer, and never give a wrong answer.

## Effectiveness

All steps must be sufficiently basic that they can be done exactly, in finite length.

## Finiteness

An algorithm must always terminate after a finite number of steps.

## Generality

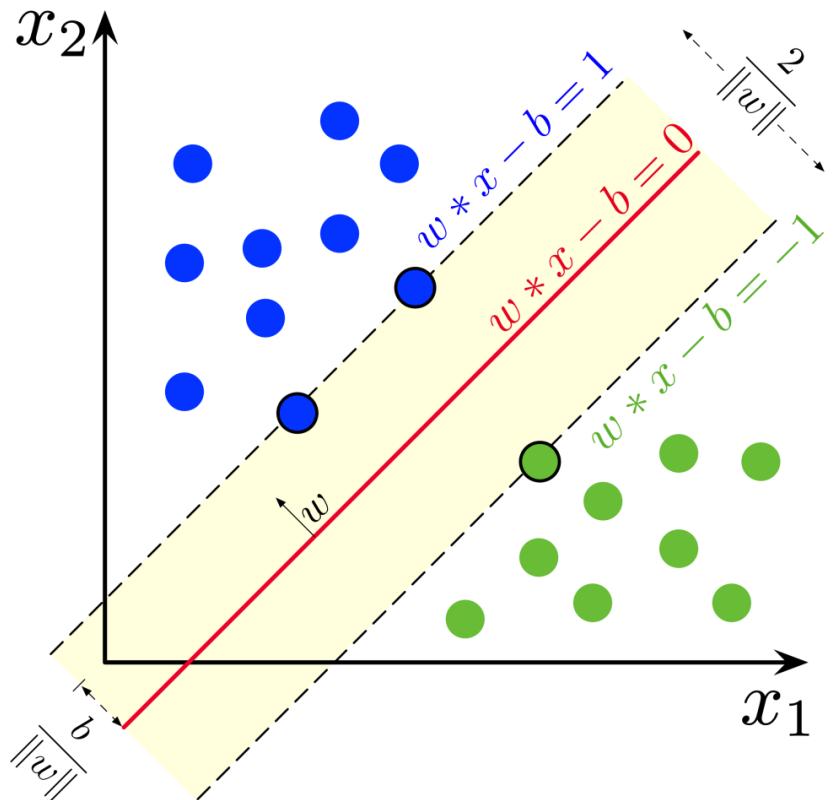
An algorithm must work for all instances of the class of problems it is designed to solve.

# Where do we find algorithms?

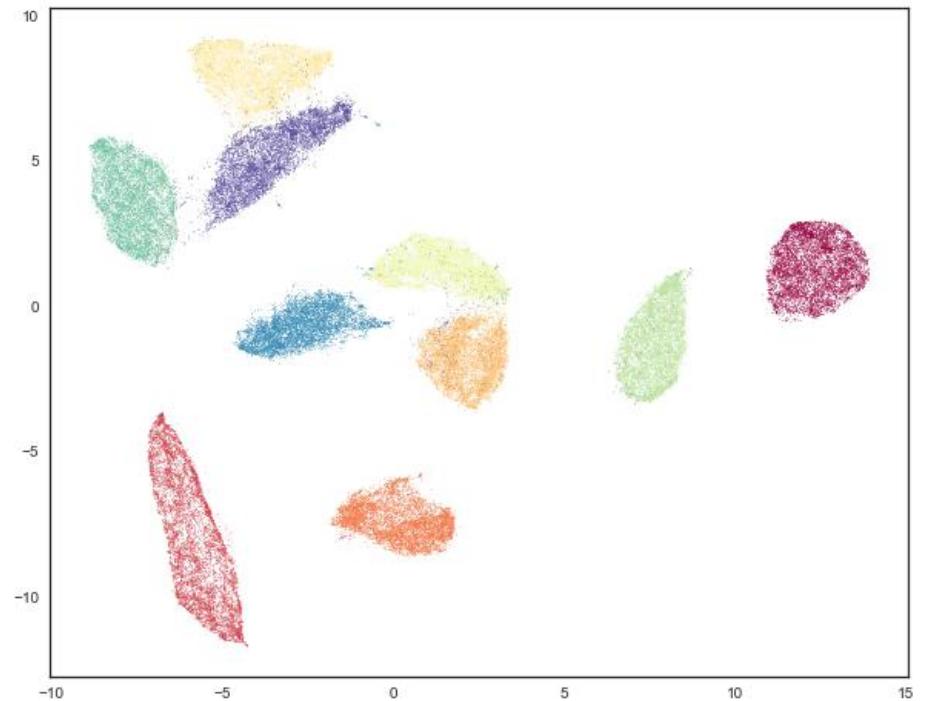
A little bit everywhere.

# Where do we find algorithms?

Machine Learning and AI



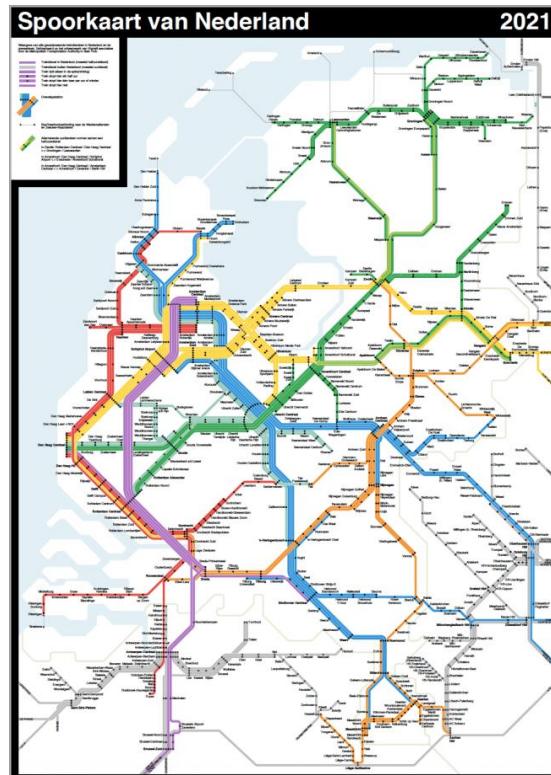
Support Vector Machine (SVM)  
<https://www.wikipedia.org>



Uniform Manifold Approximation and Projection (UMAP) 11  
<https://umap-learn.readthedocs.io/>

# Where do we find algorithms?

Scheduling, Search, Recommendation Systems

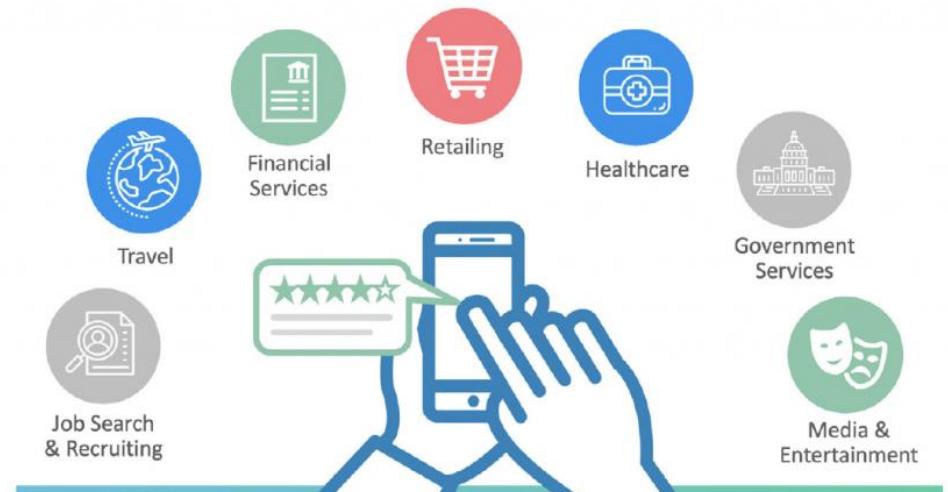


<https://www.treinreiziger.nl/spoorkaart-2019-deze-vier-versies-zijn-er/>

Train scheduling



Recommendations Are Everywhere



<https://neo4j.com/blog/enterprise-real-time-recommendation/>

Search & Recommendation

# Where do we find algorithms?

Biomedicine and Bioengineering



Molecular diagnostics  
Precision therapy



artios

NOVARTIS

Consensus

Human SARS-CoV-2

Pangolin CoV

Bat CoV (RaTG13)

T G C V I A W N S N N I D S K V G G N Y I  
T G C V I A W N S N N L D S K V G G N Y I  
T G C V I A W N S N N L D S K V G G N Y I  
T G C V I A W N S K H I D A K E G G N F I

BIONTECH

pfizer

moderna

Vaccine development

DSM



<https://www.delish.com/food/g937/alternative-uses-for-beer/>  
<https://www.schiphol.nl/en/at-schiphol/shop/products/gouda-farm-cheese/10011>

Genetwister  
keyGene



<https://www.1800flowers.com/blog/flower-facts/tulip-color-meanings/>

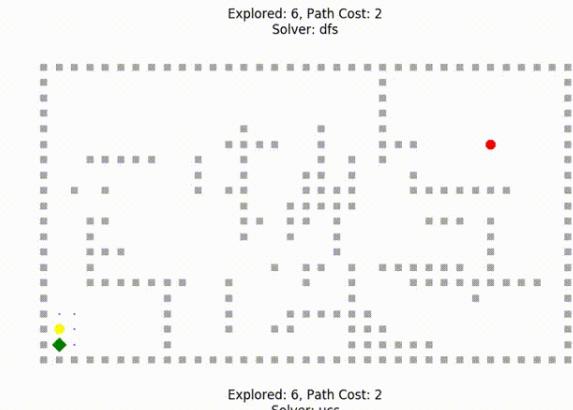
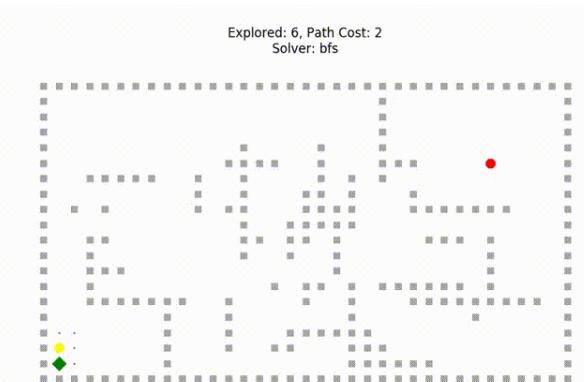
Plants and microorganisms for food industry

# Algorithms in this course

	Play All	Insertion	Selection	Bubble	Shell	Merge	Heap	Quick
Random								
Nearly sorted								
Reversed								
Few unique								

Sorting

<https://www.toptal.com/developers/sorting-algorithms>



Searching, path-finding

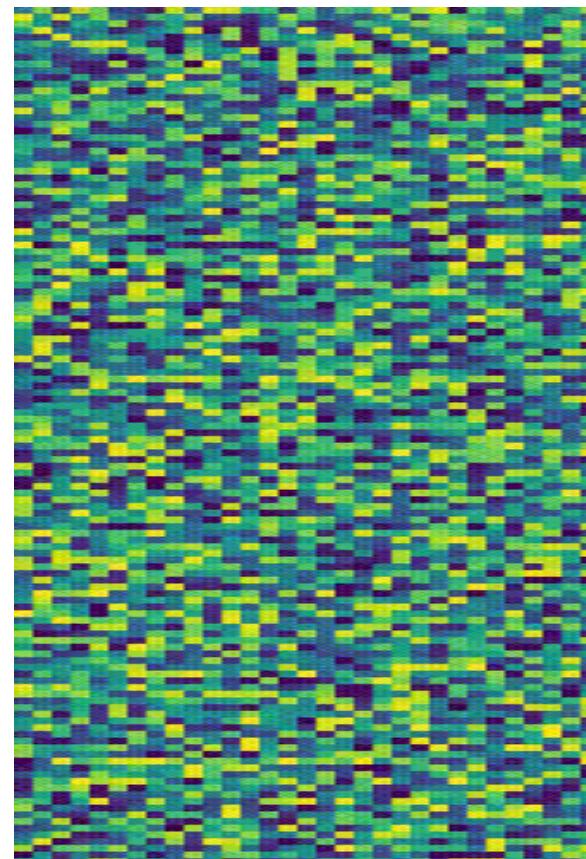
<https://www.youtube.com/channel/UCMPAzg0mKISKn8D9dVNNGcQQ/><sup>14</sup>

# Algorithm design and analysis patterns

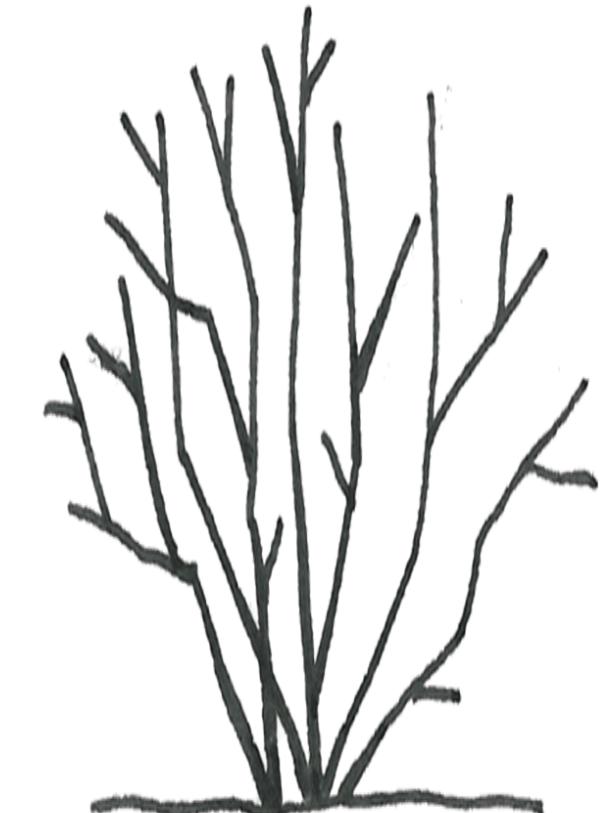


Recursion

<https://www.pinterest.com/pin/350858627217843019/>



Divide-and-Conquer



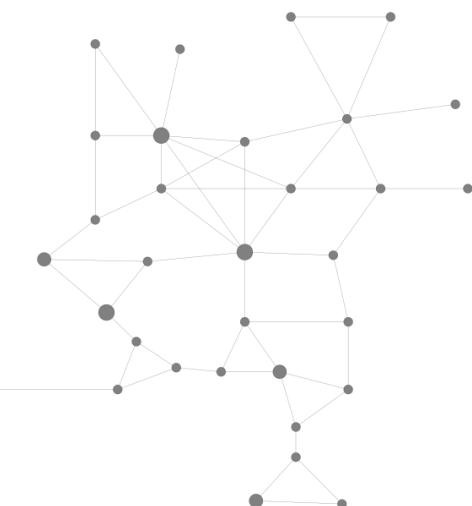
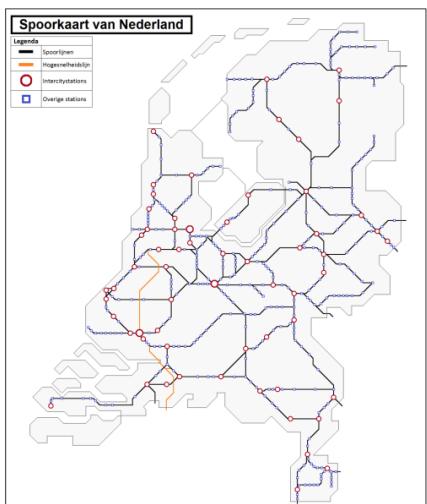
Prune-and-Search

<https://www.kb.jniplants.com/haircut-method-pruning/>

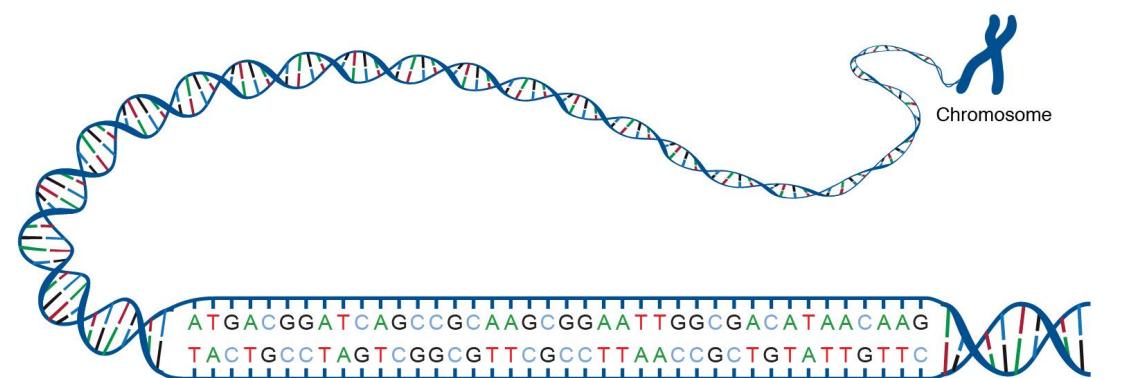
# Data structures

Algorithms rely on appropriate representations of the input data.

Data structure: representation of the data that enables efficient access and processing.



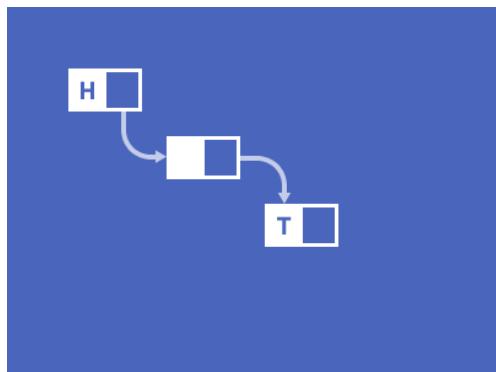
Train network: represented as a graph



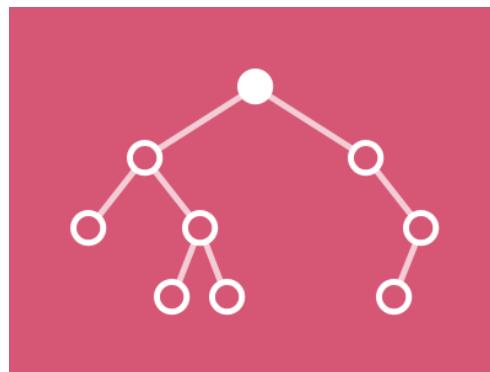
<https://www.genome.gov/genetics-glossary/acgt>

Genome: sequence of letters from  
{A,C,G,T}

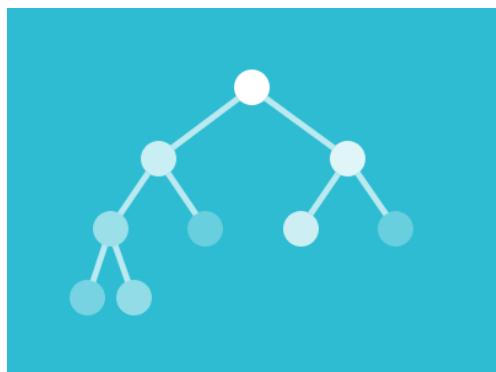
# Data structures in this course



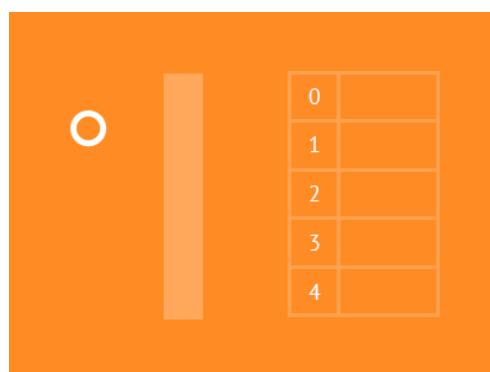
Containers (arrays,lists,trees,sets)



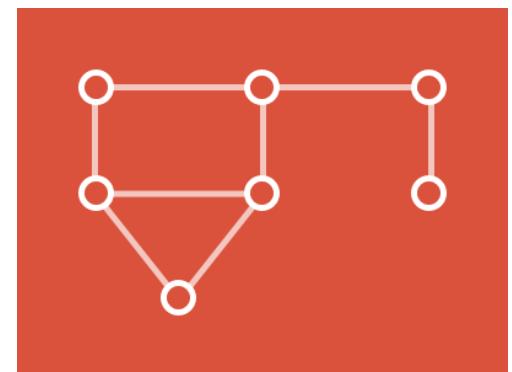
Search trees



Ordered (stacks,queues,heaps,maps)



Hash tables

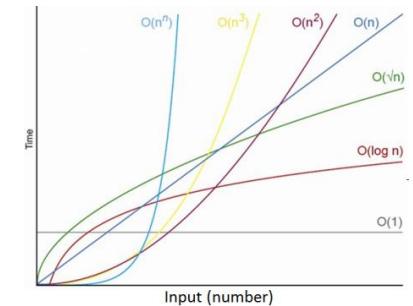


**Graph data structures**

**Data containers, Ordered data structures   Search data structures**

# Analysis of algorithms/data structures

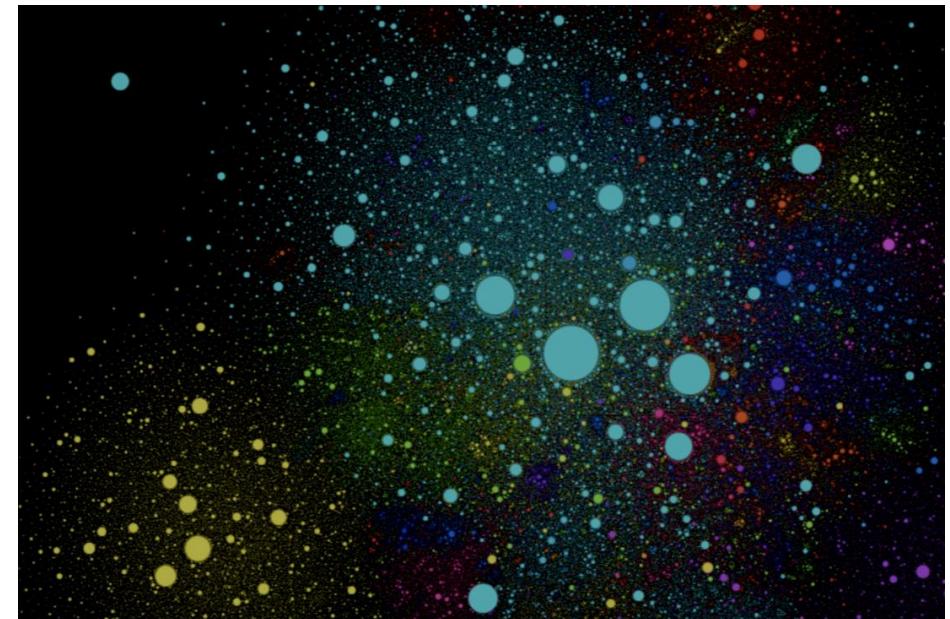
How do we choose an algorithm or data structure for a given problem?  
Based on performance: we want efficiency and scalability!



We will learn to analyse the cost of algorithms/structures as a function of input size.  
How much **time** does it take to run? How much **space** does it use?



<https://crossborder.digital/>



<https://internet-map.net/>

# Is ADS a difficult course?

It depends

- You'll need to:

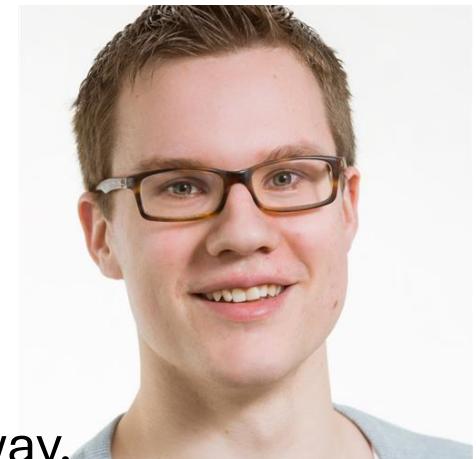
- reason your way through difficult problems,
- implement the solutions using your Java knowledge,
- and then also analyze your solutions in a mathematical way.

- On the flip side:

- you don't have to memorize a lot,
- and once you understand it, questions are usually trivial to answer.

- Advice:

- Keep up with the material, even if assignments are not mandatory.
- Attend the labs!



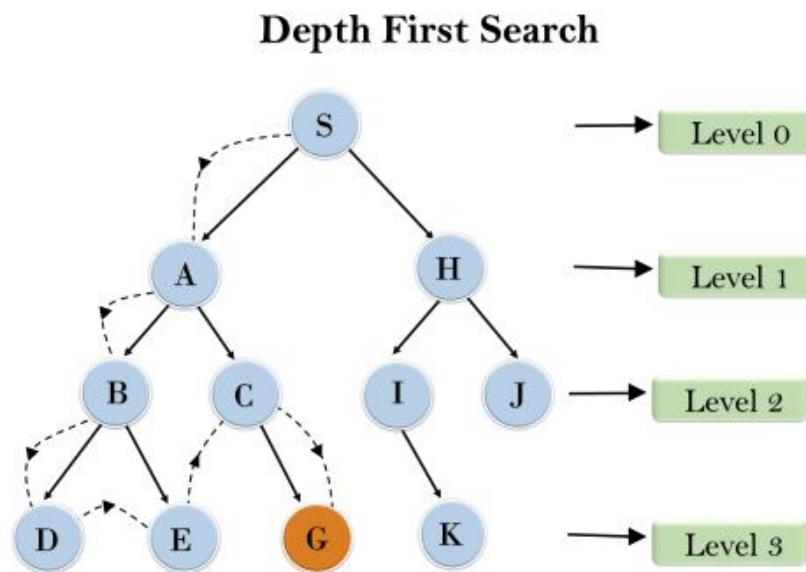
# Is ADS a difficult course?

- ADS might be different than you are used to
- In most courses, you learn the material and reproduce at the exam
  - (because of this, doing a lot of practice exams is very effective)
- In ADS, you need to learn to solve problems
- Which study strategies are effective are different here
- It is important to teach yourself a structured way to solve problems and come up with questions that help you when you are stuck
  - The Acing ADS sessions are designed with this purpose in mind

# Changes since last year

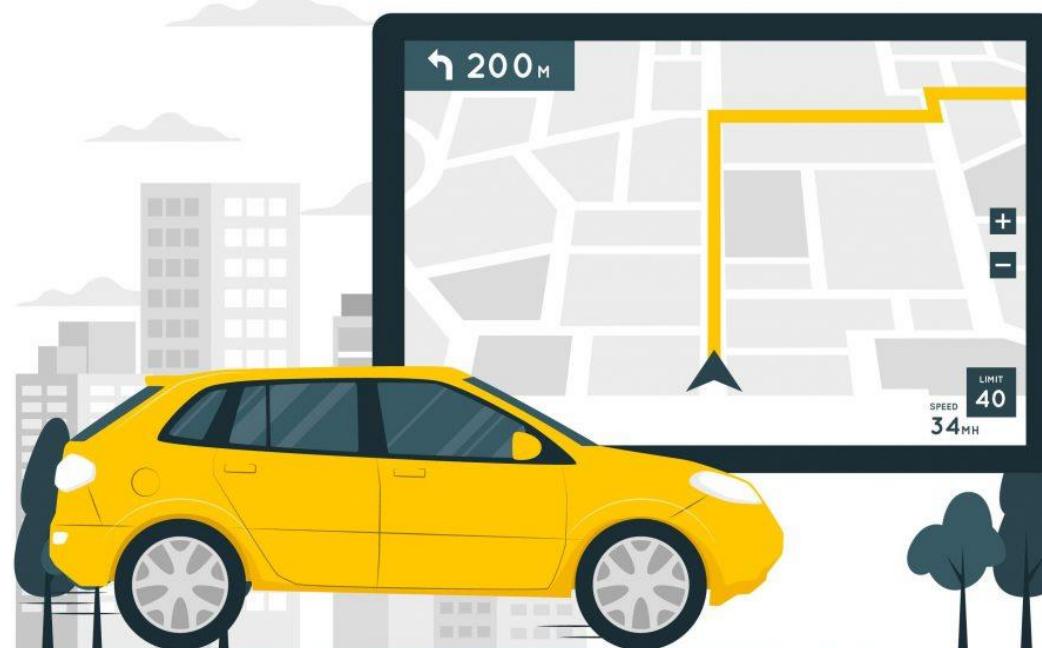
- Students were divided about flipped classroom:
  - We will mix the classical and flipped classroom style more
  - We won't do implementation assignments in lectures anymore
- Students want explicit strategies to solve implementation assignments:
  - Acing ADS is now a core part of the course, in which will teach you the general strategy of how to approach implementation problems
  - We will end Acing ADS with live-coding of the solution
- Additionally:
  - Feedback on homework is now done live by TAs

# Problem Solving



<https://columbium41.github.io/Maze-Solver/about.html>

# Problem Solving



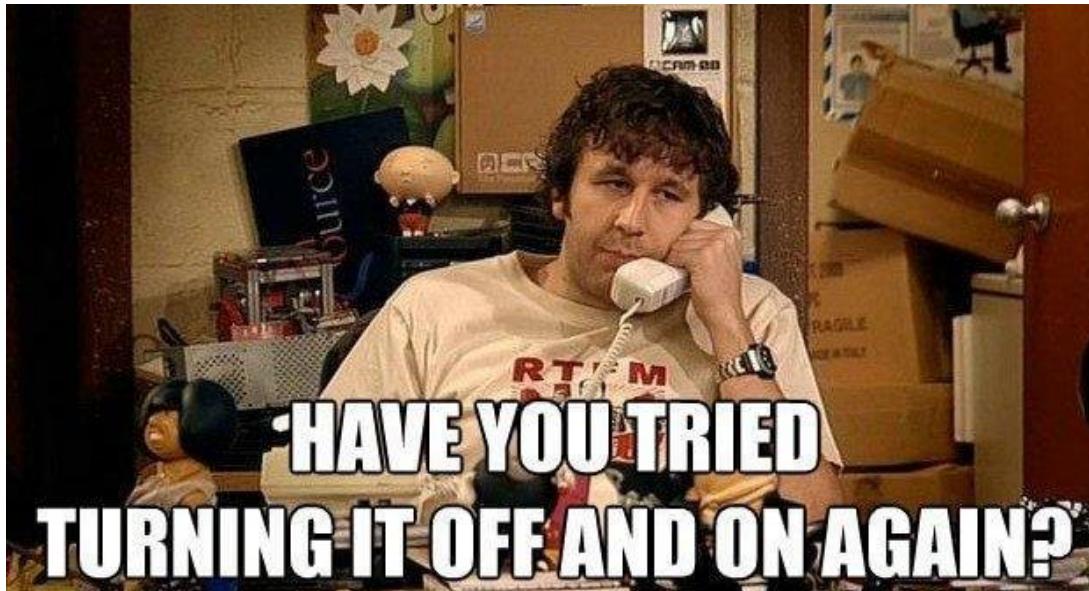
<https://routeplannernet.nl/routeplanner-auto/>

# Problem Solving



<https://www.rijkswaterstaat.nl/nieuws/archief/2023/09/26e-functioneringssluiting-maeslantkering-geslaagd>

# Problem Solving



<https://www.cipher-it.co.uk/blog/2017/11/28/have-you-tried-turning-it-off-and-on-again/>

# Our first problem

- River with towns along it over a length of 20 km
- Each town is characterized with the distance to the sea travelling along the river
- List<Integer> containing this distance (between 0 and 20) of all towns (in a random order)
- The task is to determine whether there are more towns in the first 10 km or in the last 10 km

Design an algorithm which has as input argument the list of distances as a List<Integer> and outputs true if there are more towns in the first 10 km, and false if there are more towns in the last 10 km or if it's equal.

# How to design an algorithm

- Work an example by hand
  - For the list [15, 3, 12, 7, 1, 19, 13], the answer should be false
- Write down what you did
  - Numbers lower than 10 are 1, 3 and 7, so 3 in total; numbers higher than 10 are 12, 13, 15 and 19, so 4 in total; 4 is more than 3, so I should return false
- Find patterns in what you did and generalize
  - I counted all numbers below 10, then all numbers above 10, and checked which of those counts was higher
    - More detailed: I counted all numbers below 10 by going over all numbers and checking whether they were lower than 10
    - Similar for numbers above 10
    - If there are more numbers below 10 than there are numbers above 10, I return true; otherwise, I return false

# How to design an algorithm

- Write it in pseudo-code
  - Start with a counter for the number of values below 10
  - Then, for each element in the list:
    - Check if it is smaller than 10; if so:
      - Add 1 to the counter for the values below 10
  - Make another counter for the number of values above 10
  - For each element in the list:
    - Check if it is greater than 10; if so:
      - Add 1 to the counter for values above 10
  - If the counter for values below 10 is greater than the counter for values below 10:
    - Return true
  - Else:
    - Return false

# How to design an algorithm

- Check the algorithm by hand
  - Does our algorithm work for the list [11, 3, 12, 10, 7]?
    - $11 \not< 10$ ,  $3 < 10$ ,  $12 \not< 10$ ,  $10 \not< 10$  and  $7 < 10$ , so smaller will be 2
    - $11 > 10$ ,  $3 \not> 10$ ,  $12 > 10$ ,  $10 \not> 10$  and  $7 \not> 10$ , so greater will be 2
    - $2 \not> 2$ , so return false
    - Correct?
- Translate the algorithm to code
  - While coding in this course, you think you will spend most of your time here. However, you will be doing the first five steps at the same time. This is not recommended by us but some of you will do it anyway because it seems like it's gonna save time. This is not a problem if it works in that particular exercise, but do realize you can fall back on these steps if you get stuck (don't work on an exercise for more than an hour before doing this!).
- Run test cases
- Debug failed test cases

# How to design an algorithm

- Work an example by hand
  - For the list [15, 3, 12, 7, 1, 19, 13], the answer should be false
- Write down what you did
  - Numbers lower than 10 are 1, 3 and 7, so 3 in total; numbers higher than 10 are 12, 13, 15 and 19, so 4 in total; 4 is more than 3, so I should return false
- Find patterns in what you did and generalize
  - I counted all numbers below 10, then all numbers above 10, and checked which of those counts was higher
- Write it in pseudo-code
- Check the algorithm by hand
  - Does our algorithm work for the list [11, 3, 12, 10, 7]?
- Translate the algorithm to code
- Run test cases
- Debug failed test cases

# The solution

```
public static boolean Method1(int[] distances) {  
    int smaller = 0;  
    for(int i = 0; i < distances.length; i++) {  
        if(distances[i] <= 10) {  
            smaller++;  
        }  
    }  
    int greater = 0;  
    for(int i = 0; i < distances.length; i++) {  
        if(distances[i] > 10) {  
            greater++;  
        }  
    }  
    if(smaller > greater) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

# Another solution

```
public static boolean Method2(int[] distances) {  
    int smaller = 0;  
    int greater = 0;  
    for(int i = 0; i < distances.length; i++) {  
        if(distances[i] <= 10) {  
            smaller++;  
        } else {  
            greater++;  
        }  
    }  
    if(smaller > greater) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

# Yet another solution

```
public static boolean Method3(int[] distances) {  
    int smaller = 0;  
    for(int i = 0; i < distances.length; i++) {  
        if(distances[i] <= 10) {  
            smaller++;  
        }  
    }  
    return smaller > distances.length-smaller;  
}
```

Which implementation is faster?

- Method1
- Method2
- Method3

# Empirical analysis

Input length	Method1	Method2	Method3
1	0ms	0ms	0ms
10	0ms	0ms	0ms
100	0ms	0ms	0ms
1.000	0ms	0ms	0ms
10.000	0ms	0ms	1ms
100.000	2ms	2ms	2ms
1.000.000	5ms	4ms	1ms
10.000.000	12ms	31ms	7ms
100.000.000	110ms	309ms	54ms
1.000.000.000	1177ms	3053ms	563ms

# Pitfalls of empirical analysis

What are the pitfalls of empirical analysis?

- Results may differ when different hardware/compiler/OS/. . . is used
- Experiments are restricted to a limited set of inputs
- Requires a full implementation of the algorithm
- The speed of your computer might vary over time

Why does it matter to have fully implemented the algorithm?

- During the design, it would be foolish to spend much time implementing a certain algorithm that would clearly be inferior by theoretical analysis

# Count primitive operations

- Assigning a value to a variable
- Performing an arithmetic operation
- Comparing two numbers
- Calling a method
- Returning from a method
- ...

# The solution

```
public static boolean Method1(int[] distances) { How many operations are done here?  
    int smaller = 0; 1  
    for(int i = 0; i < distances.length; i++) { 1 + (n + 1) + 2n  
        if(distances[i] <= 10) { n  
            smaller++; 2n  
        }  
    }  
    int greater = 0; 1  
    for(int i = 0; i < distances.length; i++) { 1 + (n + 1) + 2n  
        if(distances[i] > 10) { n  
            greater++; 2n  
        }  
    }  
    if(smaller > greater) { 1  
        return true; 1  
    } else { 1  
        return false;  
    }  
}
```

Total:  $10n + 8$

# Another solution

```
public static boolean Method2(int[] distances) {  
    int smaller = 0;                                1  
    int greater = 0;                                 1  
    for(int i = 0; i < distances.length; i++) {     1 + (n + 1) + 2n  
        if(distances[i] <= 10) {  
            smaller++;                            n  
        } else {  
            greater++;                           2n  
        }  
    }  
    if(smaller > greater) {                         1  
        return true;                             1  
    } else {  
        return false;                            1  
    }  
}
```

Total:  $6n + 6$

# Yet another solution

```
public static boolean Method3(int[] distances) {  
    int smaller = 0;                                1  
    for(int i = 0; i < distances.length; i++) {      1 + (n + 1) + 2n  
        if(distances[i] <= 10) {  
            smaller++;                                n  
        }                                              2n  
    }  
    return smaller > distances.length-smaller;      3  
}
```

Total:  $6n + 6$

# Pitfalls of operation counting

What are the pitfalls of operation counting?

- It is not clear what exactly 1 operation is
- Operation amounts might differ for various programming languages
- Not all operations take the same amount of time
- It is hard to count operations when you use methods from external libraries
- It apparently doesn't actually represent the amount of time it takes!

# Another problem

Inhabitants from the towns that lie along the river occasionally want to travel to other towns. Since they lie along a river, a boat seems to be a good way to do this. While designing such a boat, we have to know how large the fuel tank should be. The boat can fully refuel at a town, but it doesn't want to stop during a trip from one town to another. To do a trip from one town to another, the boat uses 1 unit of fuel for each km it travels. If we want our boat to be able to travel from any town to any other town, how many units of fuel should our boat be able to hold?

- Make an algorithm that produces the amount of fuel the boat should be able to hold
- Count the operations of your algorithm (just like I did in this lecture)
- If you have extra time: Make a different algorithm and compare it with your other algorithm by counting operations

# Course staff



**Megha Khosla**  
Lecturer



**Ivo van Kreveld**  
Lecturer

**Diana Banta**  
Head TA

**Julius Gvozdiová**  
Head TA

**Alexandru Postu**  
Head TA

# What?

- Learn properties of standard algorithms and data structures
- Solve algorithmic problems
- Implement algorithms and data structures using Java
- Analyse time and space complexity of algorithms and data structures

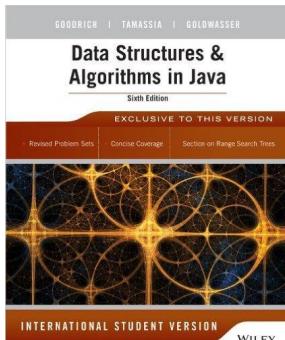
# Why?

- Problem-solving is useful anywhere in life
- While programming, it is useful to know which algorithms and data structures to use and to be able to think of your own algorithms and analyse them

# Who?

- You!
- Education is about students: you are the one going to learn something
  - You have a lot of freedom
  - You have the responsibility
  - We are here to help you

How?



# Per lecture (twice a week)

## Watch videos and scan book

# Do homework assignments

Lecture

## Continue homework assignments

Finish  
homework  
assignments

Per lab  
(once a week)

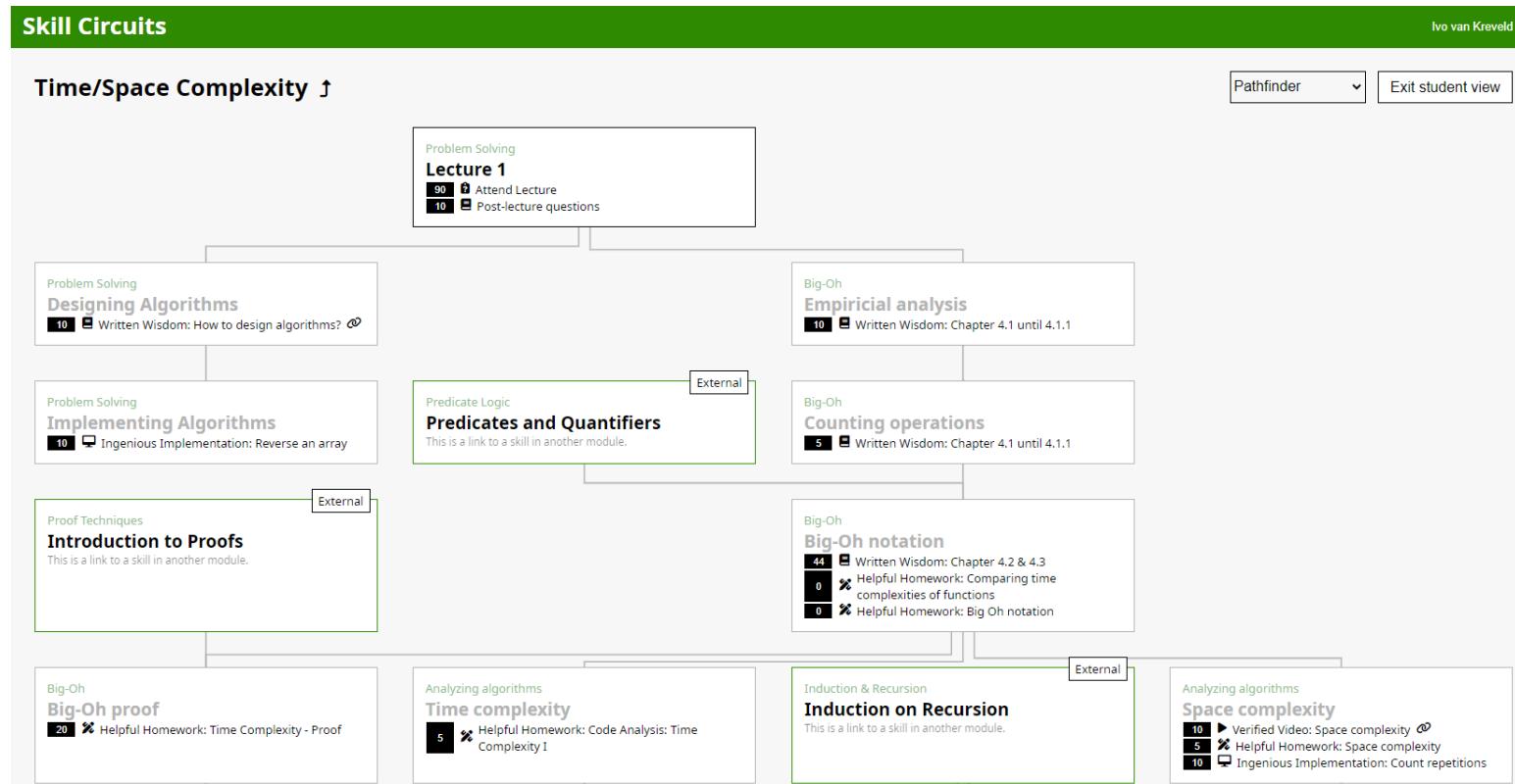
Ask questions  
about  
homework

## Get feedback on homework

# Acing ADS



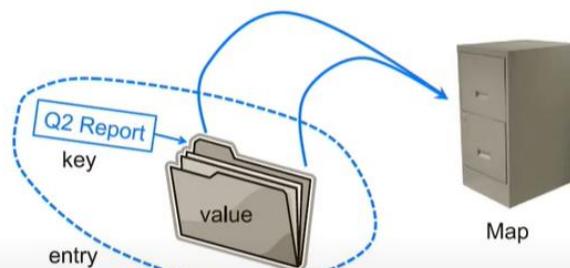
# Skill Circuits



# The videos

## The map ADT

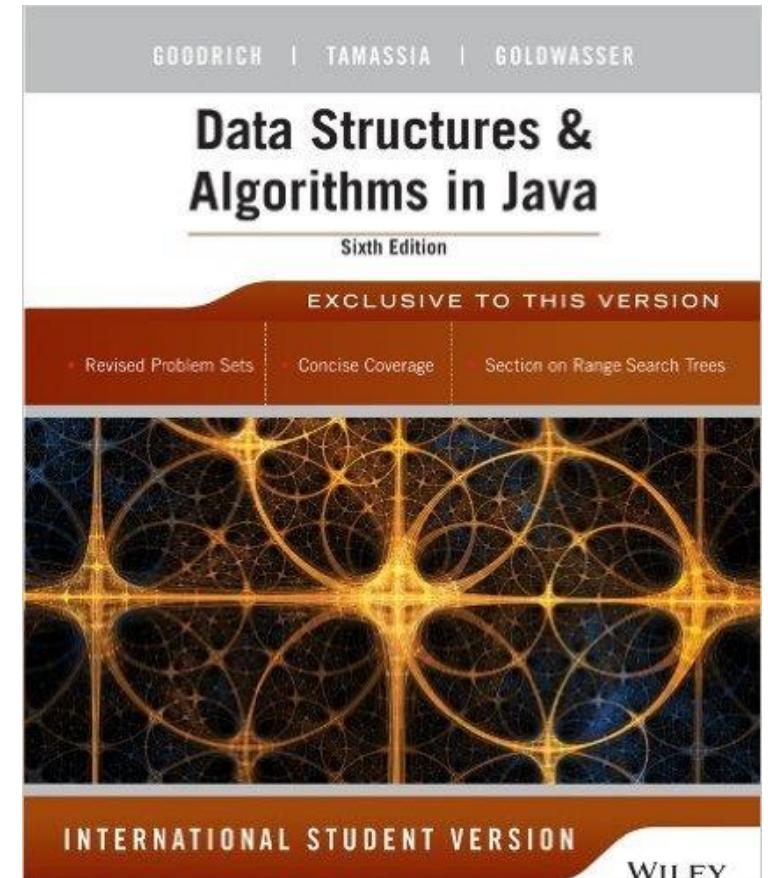
```
1 public interface Map<K,V> {  
2     int size();  
3     boolean isEmpty();  
4     V get(K key);  
5     V put(K key, V value);  
6     V remove(K key);  
7     /* more operations in the book */  
8 }
```



Week 2.6 - Maps (1/3): maps

# The book

- Data Structures And Algorithms In Java
- Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser
- 6th edition
- ISBN 978-1-118-80857-3



# How to study?

How will you study for ADS?

- a) I will only go to the lectures
- b) I will only watch the video's
- c) I will only read the book
- d) I will go to the lectures first, then watch the video's
- e) I will go to the lectures first, then read the book
- f) I will go to the lectures first, then watch the video and read the book
- g) I will watch the videos, then go to the lecture
- h) I will read the book, then go to the lecture
- i) I will watch the videos and read the book, then go to the lecture
- j) I will watch the videos and scan the book, then go to the lecture

# How to study?

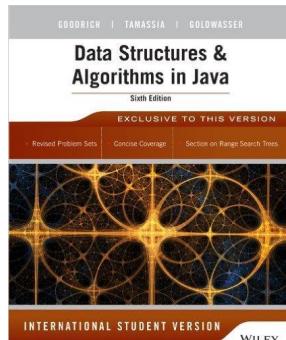
How will you study for ADS?

- a) I will only go to the lectures
- b) I will only watch the video's
- c) I will only read the book
- d) I will go to the lectures first, then watch the video's
- e) I will go to the lectures first, then read the book
- f) I will go to the lectures first, then watch the video and read the book
- g) I will watch the videos, then go to the lecture
- h) I will read the book, then go to the lecture
- i) I will watch the videos and read the book, then go to the lecture
- j) **I will watch the videos and scan the book, then go to the lecture**

# Homework

- Analysis
  - Multiple choice and open questions
  - If you submit before the deadline, you can get feedback
  - The deadline is Friday 23:59 of each week
- Implementation
  - Write Java code according to specifications
  - Public tests can help you debug your code
  - Hidden spec tests help you check whether it is correct
  - Spec tests don't check all specifications (for example, they can't check whether you used a recursive method, and don't check whether you have the correct runtime complexity)

# How?



Description

Implement the following two Java methods `isPrime` and `numPrimes`:

- `isPrime` takes an integer value `n` and returns `true` if the integer is prime, `false` otherwise.

```

9  * @return returns true if n is prime, false otherwise
10 */
11 public static boolean isPrime(int n) {
12     if (n == 0 || n == 1) {
13         return false;
14     }
15     for (int i = 2; 2 * i <= n; i++) {
16         if (n % i == 0) {
17             return false;
18         }
19     }
20     return true;
21 }
22
23 /**
24 * Counts and returns the number of prime numbers that are

```

Solution Test

Console Discussion Revision History

Layout: □

Saved Your Test Spec Test Mark Completed Reset

Status: Done

Failures (1):

JUnit Jupiter:TestSuite:checkTotalPrimes()

Method source: [classname = 'weblab.Testsuite', methodname = 'checkTotalPrimes']

→ org.opentest4j.AssertionFailedError: expected: <> but was: <>

org.junit.jupiter.api.Assertions.assertNotEqual(AssertionUtils.java:55)

org.junit.jupiter.api.Assertions.assertNotEquals(AssertionUtils.java:55)

org.junit.jupiter.api.Assertions.assertEquals(AssertionUtils.java:55)

org.junit.jupiter.api.Assertions.assertEquals(Assertions.java:52)

weblab.Testsuite.checkTotalPrimes() (TestSuite.java:19)

Test run finished after 71 ms

[ 3 containers found ]

[ 0 containers skipped ]

Attempts for "Spec Test": 1



Per lecture  
(twice a week)

Watch videos  
and scan book

Do homework  
assignments

Lecture

Continue  
homework  
assignments

Per lab  
(once a week)

Ask questions  
about  
homework

Get feedback  
on homework

Acing ADS

Finish  
homework  
assignments

Description

Implement the following two Java methods `isPrime` and `numPrimes`:

- `isPrime` takes an integer value `n` and returns `true` if the integer is prime, `false` otherwise.

```

9  * @return returns true if n is prime, false otherwise
10 */
11 public static boolean isPrime(int n) {
12     if (n == 0 || n == 1) {
13         return false;
14     }
15     for (int i = 2; 2 * i <= n; i++) {
16         if (n % i == 0) {
17             return false;
18         }
19     }
20     return true;
21 }
22
23 /**
24 * Counts and returns the number of prime numbers that are

```

Solution Test

Console Discussion

Layout: □

Saved Your Test Spec Test Mark Completed

Status: Done

Failures (1):

JUnit Jupiter:TestSuite:checkTotalPrimes()

Method source: [classname = 'weblab.Testsuite', methodname = 'checkTotalPrimes']

→ org.opentest4j.AssertionFailedError: expected: <> but was: <>

org.junit.jupiter.api.Assertions.assertNotEqual(AssertionUtils.java:55)

org.junit.jupiter.api.Assertions.assertNotEquals(AssertionUtils.java:55)

org.junit.jupiter.api.Assertions.assertEquals(AssertionUtils.java:55)

org.junit.jupiter.api.Assertions.assertEquals(Assertions.java:52)

weblab.Testsuite.checkTotalPrimes() (TestSuite.java:19)

Test run finished after 71 ms

[ 3 containers found ]

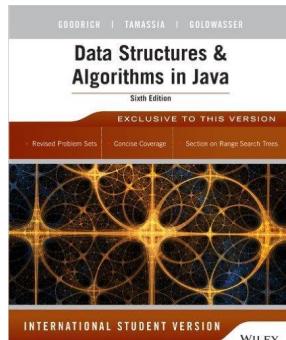
[ 0 containers skipped ]



# Acing ADS

- Similar to Acing R&L
- ... but this is a core part of this course, focusing on ‘Exam practice’
- ... maybe in a different way than you think
- You will get a really challenging problem
- We will provide help to guide you through this
- You can choose yourself how much help you use
- You will learn how to ‘help’ yourself in an exam
- You will do this in groups, which you can form there
- Maybe you can enroll with your mentor group?

# How?



Description

Implement the following two Java methods `isPrime` and `numPrimes`:

- `isPrime` takes an integer value `n` and returns `true` if the integer is prime, `false` otherwise.

Solution	Test
<pre> 9  * @return returns true if n is prime, false otherwise 10 */ 11 public static boolean isPrime(int n) { 12     if (n == 0    n == 1) { 13         return false; 14     } 15     for (int i = 2; 2 * i &lt;= n; i++) { 16         if (n % i == 0) { 17             return false; 18         } 19     } 20     return true; 21 } 22 23 /** 24 * Counts and returns the number of prime numbers that are </pre>	<p>Console Discussion Revision History</p> <p>Layout: </p> <p>Saved Your Test Spec Test Mark Completed Reset</p> <p>Status: Done Failures (1): JUnit Jupiter:TestSuite:checkTotalPrimes()</p> <p>Method source: [class: weblab.TestSuite, methodname: 'checkTotalPrimes'] -&gt; org.junit.Assert.assertError: expected: &lt;&gt; but was: &lt;&gt; org.junit.jupiter.api.Assertions.assertNotEqual(AssertionUtils.java:55) org.junit.jupiter.api.Assertions.assertEqual(AssertionUtils.java:65) org.junit.jupiter.api.Assertions.assertEquals(AssertionUtils.java:75) org.junit.jupiter.api.Assertions.assertEquals(Assertions.java:52) weblab.TestSuite.checkTotalPrimes()</p> <p>Test run finished after 71 ms [ 3 containers found ] [ 0 containers skipped ] Attempts for "Spec Test": 1</p>



Per lecture  
(twice a week)

Watch videos  
and scan book

Do homework  
assignments

Lecture

Continue  
homework  
assignments

Per lab  
(once a week)

Ask questions  
about  
homework

Get feedback  
on homework

Acing ADS

Finish  
homework  
assignments

Description

Implement the following two Java methods `isPrime` and `numPrimes`:

- `isPrime` takes an integer value `n` and returns `true` if the integer is prime, `false` otherwise.

Solution	Test
<pre> 9  * @return returns true if n is prime, false otherwise 10 */ 11 public static boolean isPrime(int n) { 12     if (n == 0    n == 1) { 13         return false; 14     } 15     for (int i = 2; 2 * i &lt;= n; i++) { 16         if (n % i == 0) { 17             return false; 18         } 19     } 20     return true; 21 } 22 23 /** 24 * Counts and returns the number of prime numbers that are </pre>	<p>Console Discussion</p> <p>Layout: </p> <p>Saved Your Test Spec Test Mark Completed</p> <p>Status: Done Failures (1): JUnit Jupiter:TestSuite:checkTotalPrimes()</p> <p>Method source: [class: weblab.TestSuite, methodname: 'checkTotalPrimes'] -&gt; org.junit.Assert.assertError: expected: &lt;&gt; but was: &lt;&gt; org.junit.jupiter.api.Assertions.assertNotEqual(AssertionUtils.java:55) org.junit.jupiter.api.Assertions.assertEqual(AssertionUtils.java:65) org.junit.jupiter.api.Assertions.assertEquals(AssertionUtils.java:75) org.junit.jupiter.api.Assertions.assertEquals(Assertions.java:52) weblab.TestSuite.checkTotalPrimes()</p> <p>Test run finished after 71 ms [ 3 containers found ] [ 0 containers skipped ] Attempts for "Spec Test": 1</p>



# When? (a suggestion...?)

Monday	Tuesday	Wednesday	Thursday	Friday
<ul style="list-style-type: none"><li>• <b>Lecture (2h)</b></li><li>• Watch videos and study book (1h)</li></ul>	<ul style="list-style-type: none"><li>• Do homework assignments (1h)</li></ul>	<ul style="list-style-type: none"><li>• <b>Lecture (2h)</b></li><li>• <b>Get feedback and ask questions about homework (2h)</b></li><li>• <b>Acing ADS (2h)</b></li></ul>	<ul style="list-style-type: none"><li>• Finish homework assignment (2h)</li></ul>	<ul style="list-style-type: none"><li>• Watch videos and study book (1h)</li><li>• Do homework assignments (1h)</li></ul>

# Where?

- Lectures: ECHO, lecture hall A (here)
- Questions and feedback lab/Acing ADS: Flux Halls (dependent on mentor group)



# Assessment

- Analysis final exam
  - Multiple choice and open questions
  - On paper
- Implementation final exam
  - Write Java code according to specifications
  - Similar to implementation homework assignments
  - In WebLab
- Grading
  - For both exams, you need to score at least a 5.0
  - If you do, your grade will be the average (both count for 50%) of both exams, which needs to be at least 5.8 to pass the course
- Make sure to enrol timely!
- There is also a midterm, but it is not assessed

# Quadruple Quest

- Only available via the Skill Circuits
- A puzzle quest that will have you solve a mystery
- Spanning 4 courses spread over 2 years of your programme
- Based on feedback from the Quadruple Quest of R&L:
  - All the time you spend doing this quest will train your ADS skills
  - The puzzles will be some more implementation exercises...
  - ... but they will require a bit more creativity
  - ... and you will still be helping Dennis solve a case that's way larger than a stolen backpack
- We're still trying to improve it, so we welcome all feedback!
- Let us know what you think or if you need a hint at [qq-cs-ewi@tudelft.nl](mailto:qq-cs-ewi@tudelft.nl)

# How to study?

How will you study for ADS?

- a) I will do all analysis homework exercises
- b) I will do the analysis homework exercises I deem useful
- c) I will do all implementation homework exercises
- d) I will do the implementation homework exercises I deem useful
- e) I will do the weekly programming challenges
- f) I will do the Acing ADS sessions
- g) I will do the Quadruple Quest
- h) I will do the practice exams

# How to study?

How will you study for ADS?

- a) I will do all analysis homework exercises
- b) I will do the analysis homework exercises I deem useful**
- c) I will do all implementation homework exercises
- d) I will do the implementation homework exercises I deem useful**
- e) I will do the weekly programming challenges
- f) I will do the Acing ADS sessions**
- g) *I will do the Quadruple Quest*
- h) I will do the practice exams**

# Remarkable Reward Register

- Designing algorithms isn't easy, but you will get there with enough practice!
- We especially think practicing together is a very good way to study!
- We understand you have a lot of other things on your mind (family, friends, other courses)
- But if you do practice with the material, we appreciate that a lot!
- So we want to reward you for that!
- Which is why we are introducing the Remarkable Reward Register!
  - By practicing the course material, you gain points. The points of all students go into one big pot!
  - Every time the number of points in that pot achieves a certain milestone, all students get a corresponding reward.

# Remarkable Reward Register

You get points for:

- Coming to the lecture: 1 point per lecture
  - Double points for the second half of the course (week 6+)
- Attempting the assignment in an Acing ADS session: 2 points each week
  - Double points for the second half of the course (week 6+)
- Handing in a homework assignment with any sensible answer: 1 point for analysis, 1 point for implementation, and 1 point if you let a TA give you feedback, each week
  - Please don't use this as a reason to copy homework
  - You'll get the point for any sensible answer, so there's no need for that
  - Double points for the second half of the course (week 6+)
- Getting a nice grade in the midterm:

Grade	Being present	$\geq 3$	$\geq 4$	$\geq 5$	$\geq 6$	$\geq 7$	$\geq 8$	$\geq 9$	10
Points	1	2	3	5	8	11	14	17	20

# Rewards

## QUESTIONS

- 1- A B C D
- 2- A B C D
- 3- A B C D
- 4- A B C D
- 5- A B C D
- 6- A B C D

Answers analysis  
homework week



1000 points: The answers to the analysis homework of week 1!

# Rewards

## QUESTIONS

- 1- A B C D
- 2- A B C D
- 3- A B C D
- 4- A B C D
- 5- A B C D
- 6- A B C D

Answers analysis  
homework week

1 2 3 4 5 6 7 8

--	--	--	--	--	--	--	--	--

Each 1000 points: The answers to an extra week of the analysis homework!

# Rewards

QUESTIONS
1- A B C D
2- A B C D
3- A B C D
4- A B C D
5- A B C D
6- A B C D

Answers analysis  
homework week

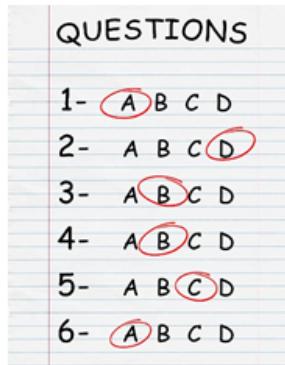
1 2 3 4 5 6 7 8




Template for  
cheat sheet

11000 points: A template for the cheat sheet you can bring at the exam!

# Rewards



Answers analysis  
homework week

1 2 3 4 5 6 7 8

--	--	--	--	--	--	--	--	--



Guest lecture  
from Stefan



Template for  
cheat sheet

14000 points: A guest lecture about approximation algorithms by Stefan Hugtenburg!

# Rewards



Guest lecture  
from Stefan



Some  
Implementation  
solutions

## QUESTIONS

- 1- A B C D
- 2- A B C D
- 3- A B C D
- 4- A B C D
- 5- A B C D
- 6- A B C D

Answers analysis  
homework week

1 2 3 4 5 6 7 8

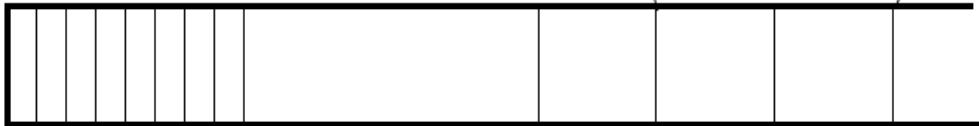

17000 points: The solution to some of the implementation assignments!

# Rewards

QUESTIONS
1- A B C D
2- A B C D
3- A B C D
4- A B C D
5- A B C D
6- A B C D

Answers analysis  
homework week

1 2 3 4 5 6 7 8



Guest lecture  
from Stefan



Exam topic in  
recap session



Template for  
cheat sheet



Some  
Implementation  
solutions

20000 points: One topic of the exam corresponds to one topic of the recap session!

# Rewards

QUESTIONS
1- A B C D
2- A B C D
3- A B C D
4- A B C D
5- A B C D
6- A B C D

Answers analysis  
homework week

1 2 3 4 5 6 7 8



Guest lecture  
from Stefan



Exam topic in  
recap session



Template for  
cheat sheet



Some  
Implementation  
solutions



Figure from  
the exam

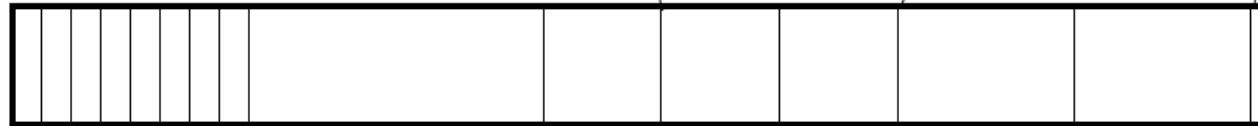
23000 points: A figure that will appear in the exam!

# Rewards

QUESTIONS
1- A B C D
2- A B C D
3- A B C D
4- A B C D
5- A B C D
6- A B C D

Answers analysis  
homework week

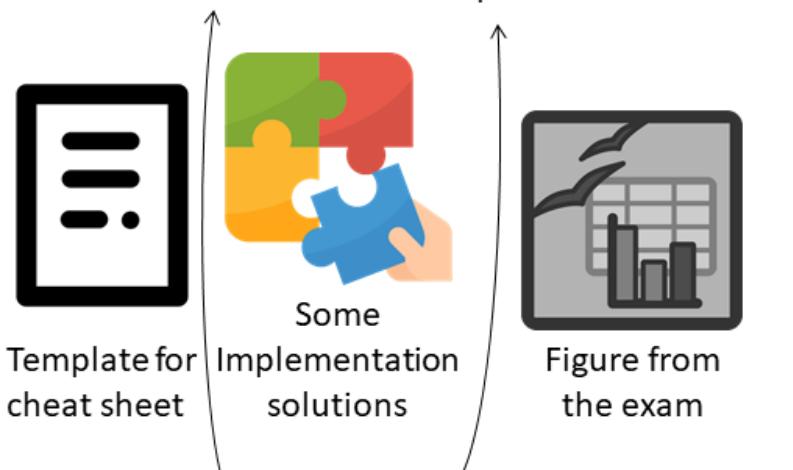
1 2 3 4 5 6 7 8



Guest lecture  
from Stefan

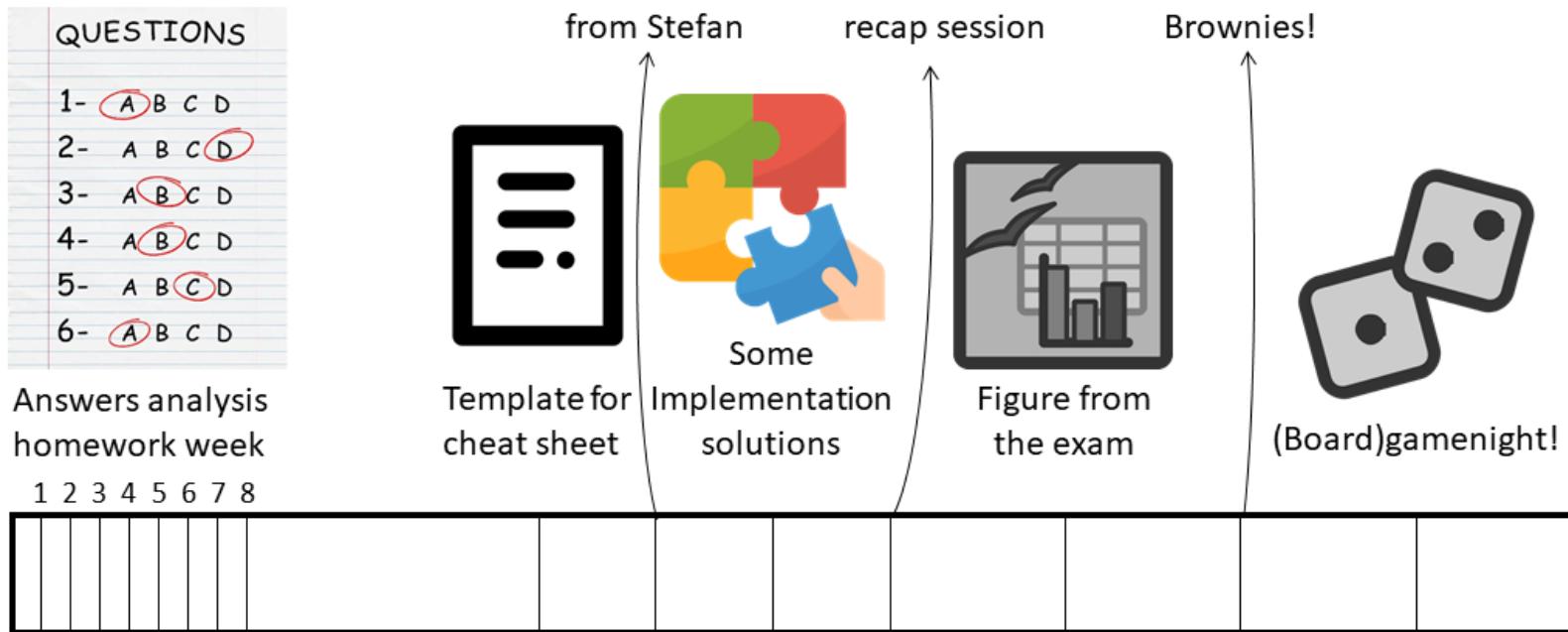
Exam topic in  
recap session

Brownies!



26000 points: The ADS-team will bake brownies for all of you!

# Rewards



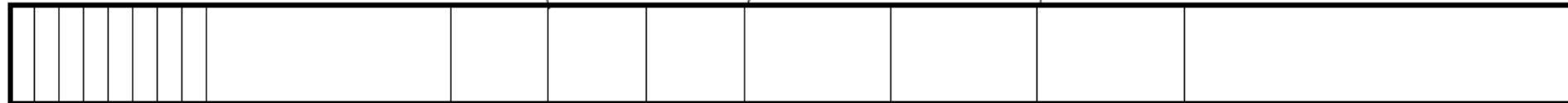
29000 points: The ADS-team will organize a (board)gamenight for everyone!

# Rewards

QUESTIONS	
1-	A B C D
2-	A B C D
3-	A B C D
4-	A B C D
5-	A B C D
6-	A B C D

Answers analysis  
homework week

1 2 3 4 5 6 7 8



100% of the points you can get (which is around 67000): A free bike for everyone!

# Questions?

- Content
  - Ask your fellow students
  - Ask them on Answers ([answers.tudelft.nl](http://answers.tudelft.nl))
  - Ask them to TAs during the shared labs
  - Ask Megha/me during/after the lecture
- Administrative issues
  - [ads-cs-ewi@tudelft.nl](mailto:ads-cs-ewi@tudelft.nl)

Good luck!

And who knows...



# The solution to the boat problem

Operations	Code
1	public static int MaxFuel1(int[] distances) {
$1 + (n + 1) + 2n$	int currentMax = 0;
$(1 + (n + 1) + 2n)*n$	for(int i = 0; i < distances.length; i++) {
$3n^2$	for (int j = 0; j < distances.length; j++) {
$n^2$	int fuel = Math.abs(distances[i] - distances[j]);
$n^2$	if (fuel > currentMax) {
1	currentMax = fuel;
1	}
1	}
1	return currentMax;
Total: $8n^2 + 5n + 4$	}

# Another solution to the boat problem

Operations	Code
1	int currentMax = 0;
$1 + (n + 1) + 2n$	for(int i = 0; i < distances.length; i++) {
$2n + (n^2/2 + n/2) + n^2 - n$	for (int j = i + 1; j < distances.length; j++) {
$3n^2/2 - 3n/2$	int fuel = Math.abs(distances[i] - distances[j]);
$n^2/2 - n/2$	if (fuel > currentMax) {
$n^2/2 - n/2$	currentMax = fuel;
	}
	}
1	return currentMax;
Total: $4n^2 + 2n + 4$	}

# Yet another solution to the boat problem

public static int MaxFuel3(int[] distances) {	Operations
int currentMax = 0;	1
for (int i = 0; i < distances.length; i++) {	$1 + (n + 1) + 2n$
int fuel = distances[i];	n
if (fuel > currentMax) {	n
currentMax = fuel;	n
}	
}	
int currentMin = Integer.MAX_VALUE;	1
for (int i = 0; i < distances.length; i++) {	$1 + (n + 1) + 2n$
int fuel = distances[i];	n
if (fuel < currentMin) {	n
currentMin = fuel;	n
}	
}	
return currentMax - currentMin;	2
}	

Total:  $12n + 8$

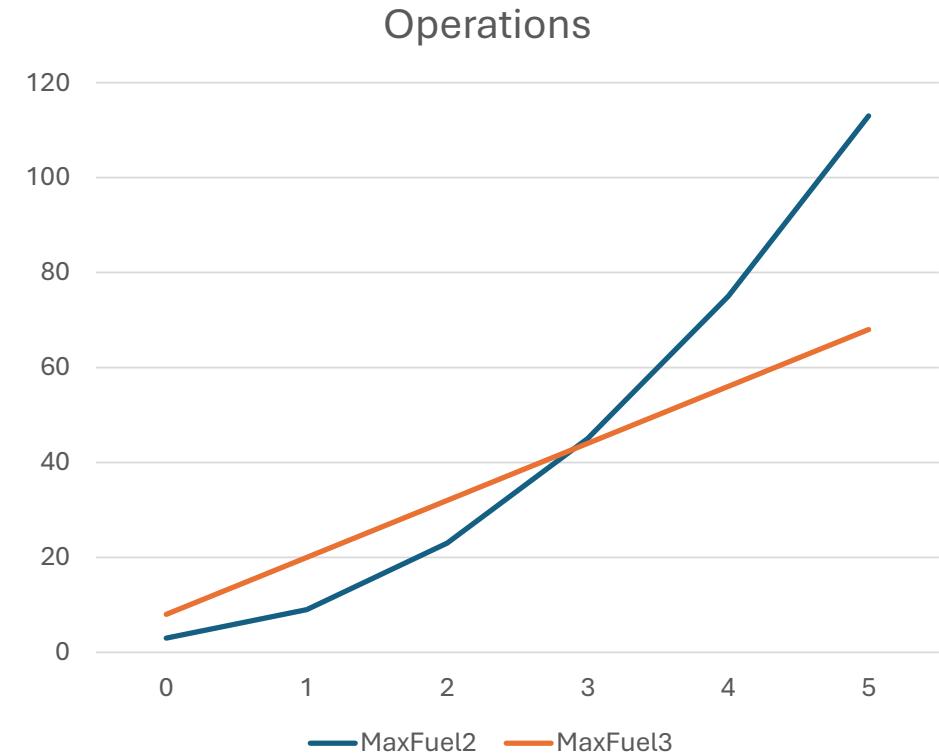
# Which solution is faster?

Which function is smaller?

- a.  $f(n) = 4n^2 + 2n + 4$
- b.  $g(n) = 12n + 8$

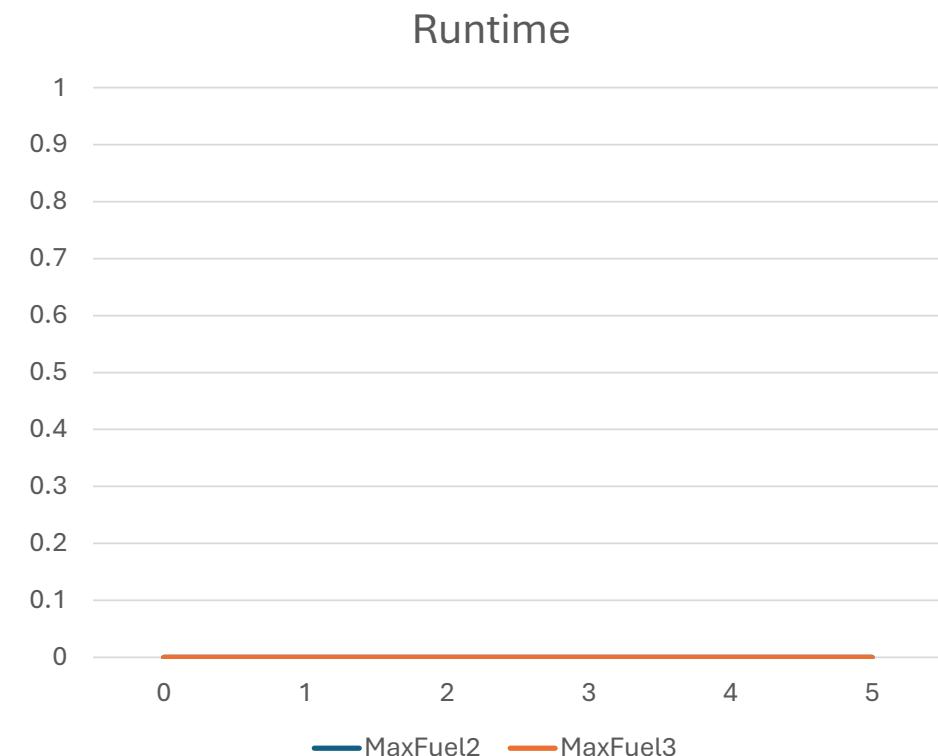
# Operation comparison

n	MaxFuel2	MaxFuel3
0	3	8
1	9	20
2	23	32
3	45	44
4	75	56
5	113	68



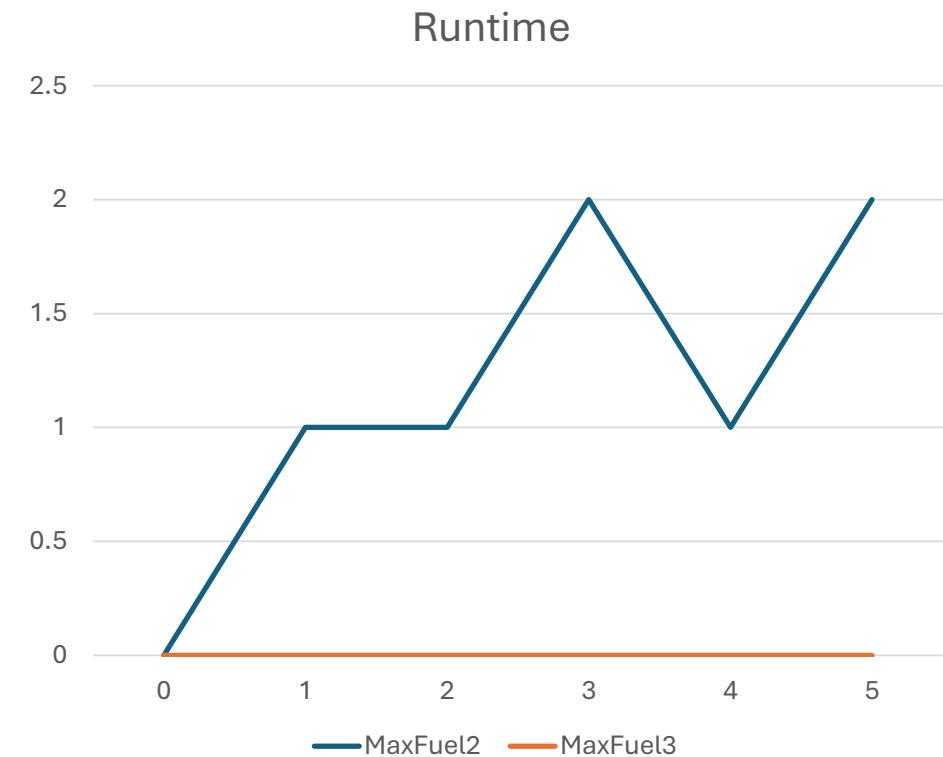
# Runtime comparison

n	MaxFuel2	MaxFuel3
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0



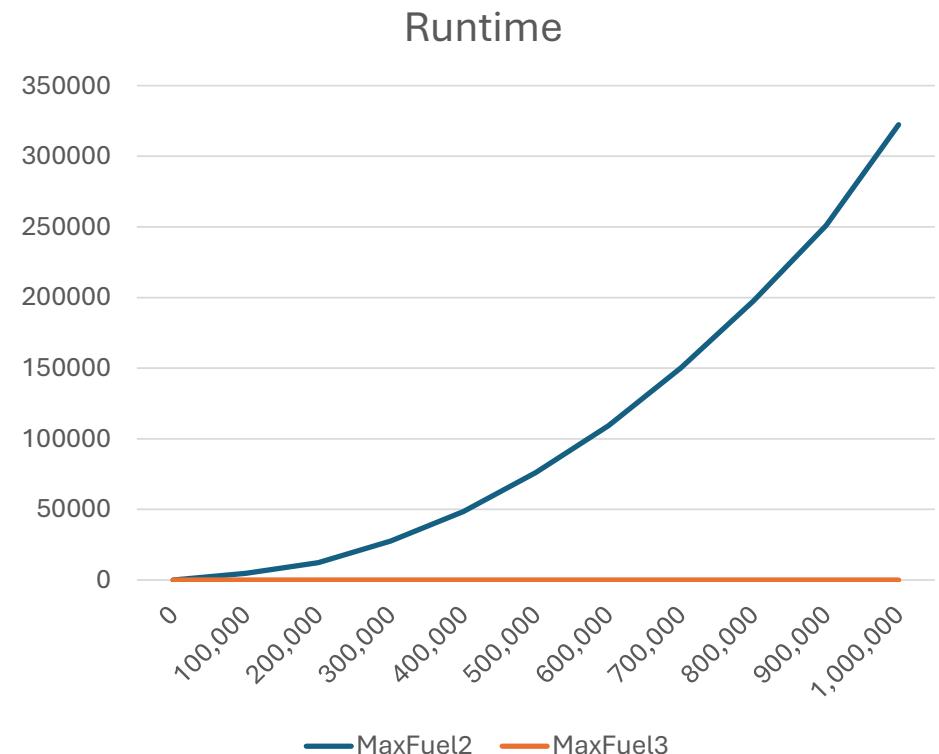
# Runtime comparison

n	MaxFuel2	MaxFuel3
0	0	0
100	1	0
200	1	0
300	2	0
400	1	0
500	2	0

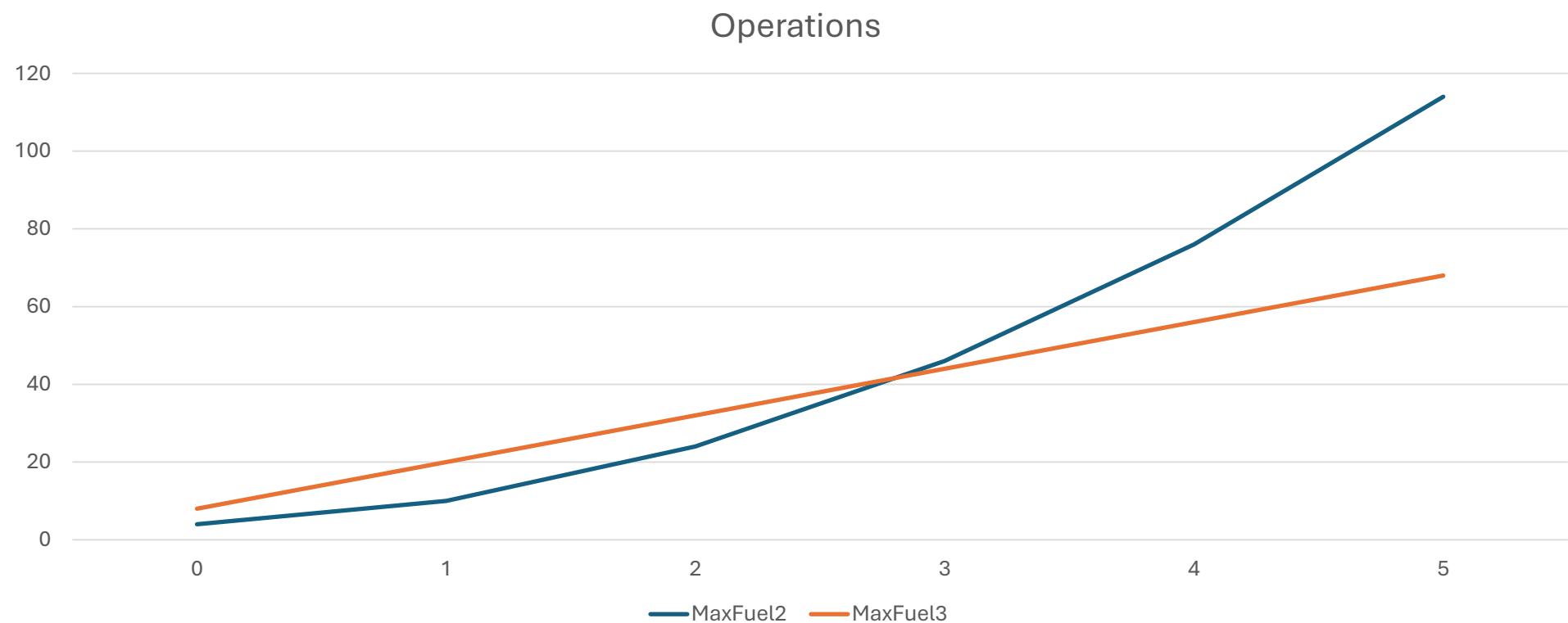


# Runtime comparison

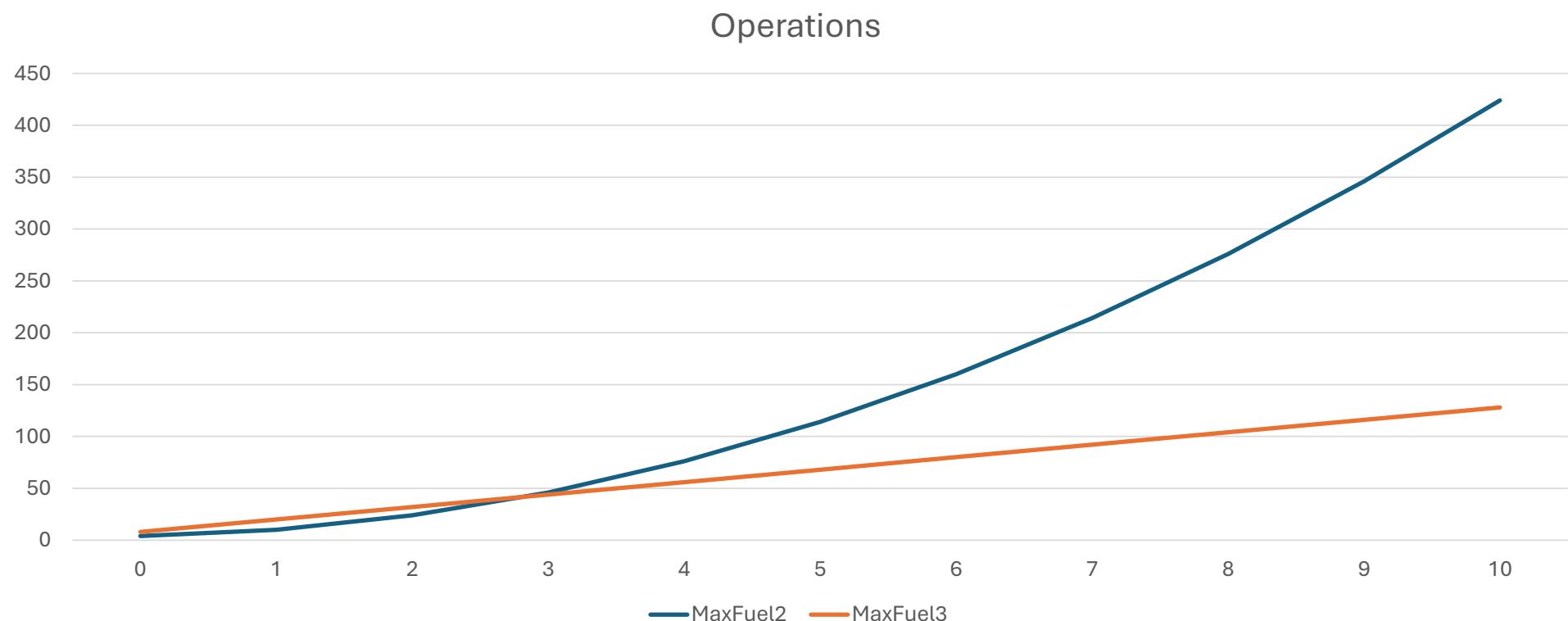
n	MaxFuel2	MaxFuel3
0	0	0
100,000	4760	1
200,000	12,205	0
300,000	27,492	0
400,000	48,506	0
500,000	76,119	0
600,000	109,228	0
700,000	150,162	1
800,000	197,453	0
900,000	251,002	1
1,000,000	322,362	1



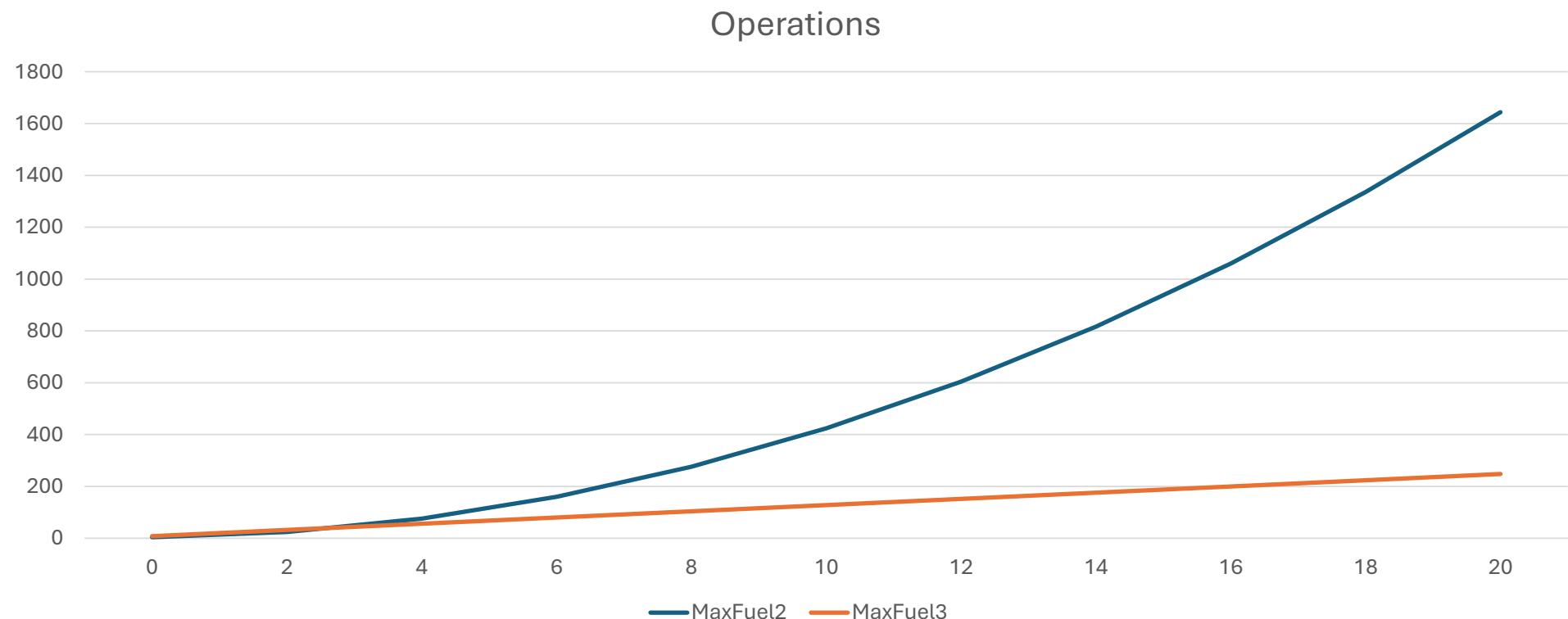
# Operation comparison



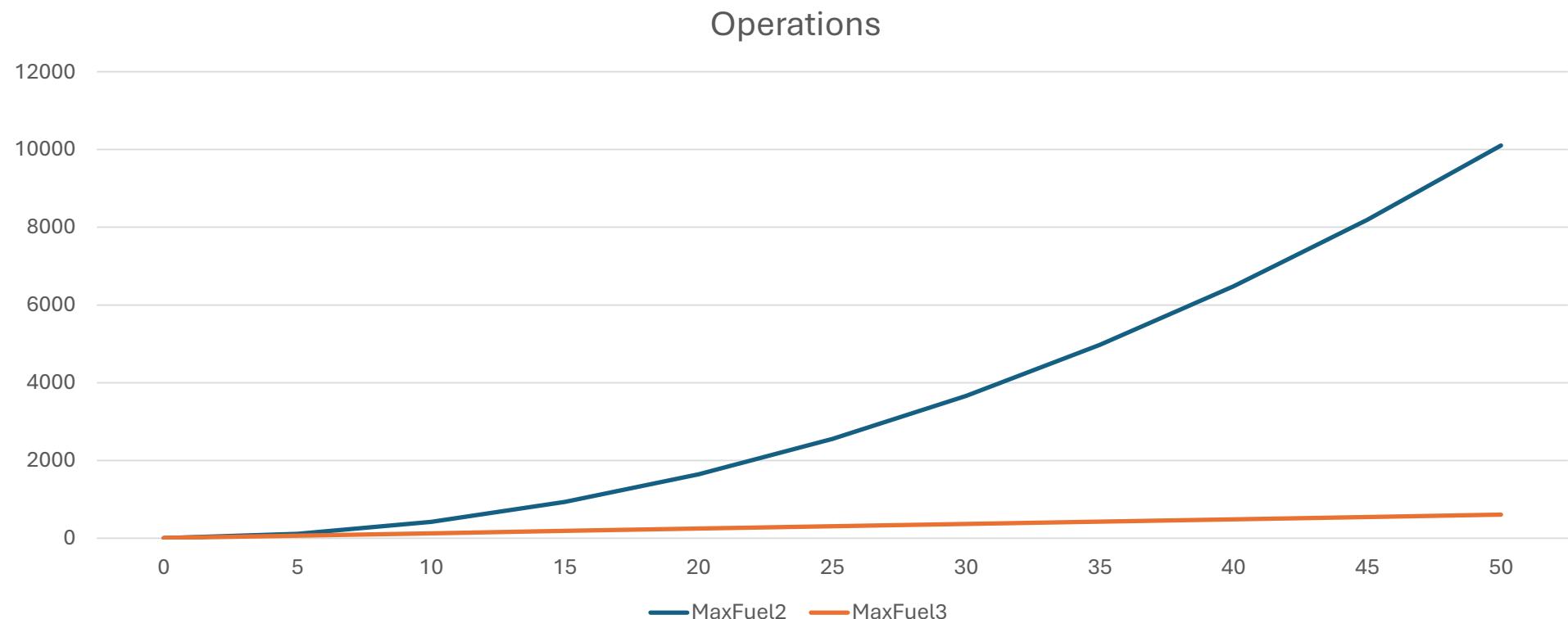
# Operation comparison



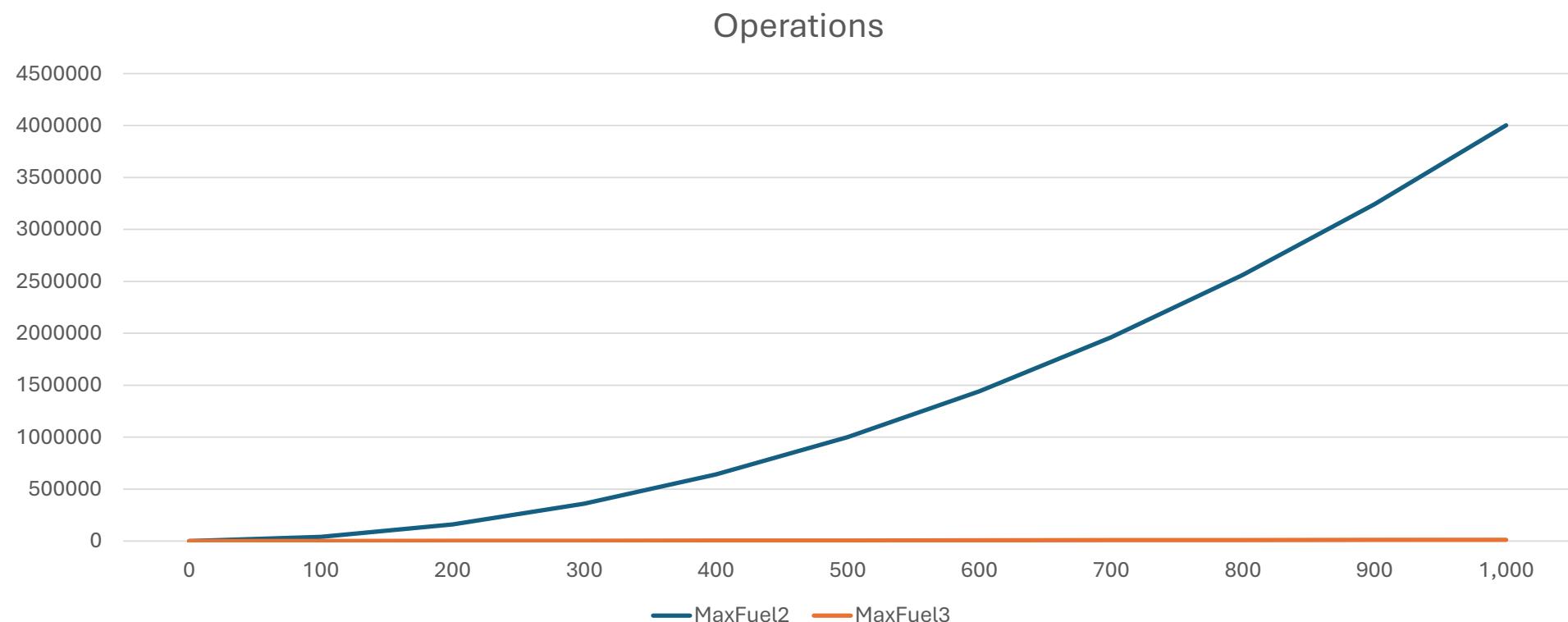
# Operation comparison



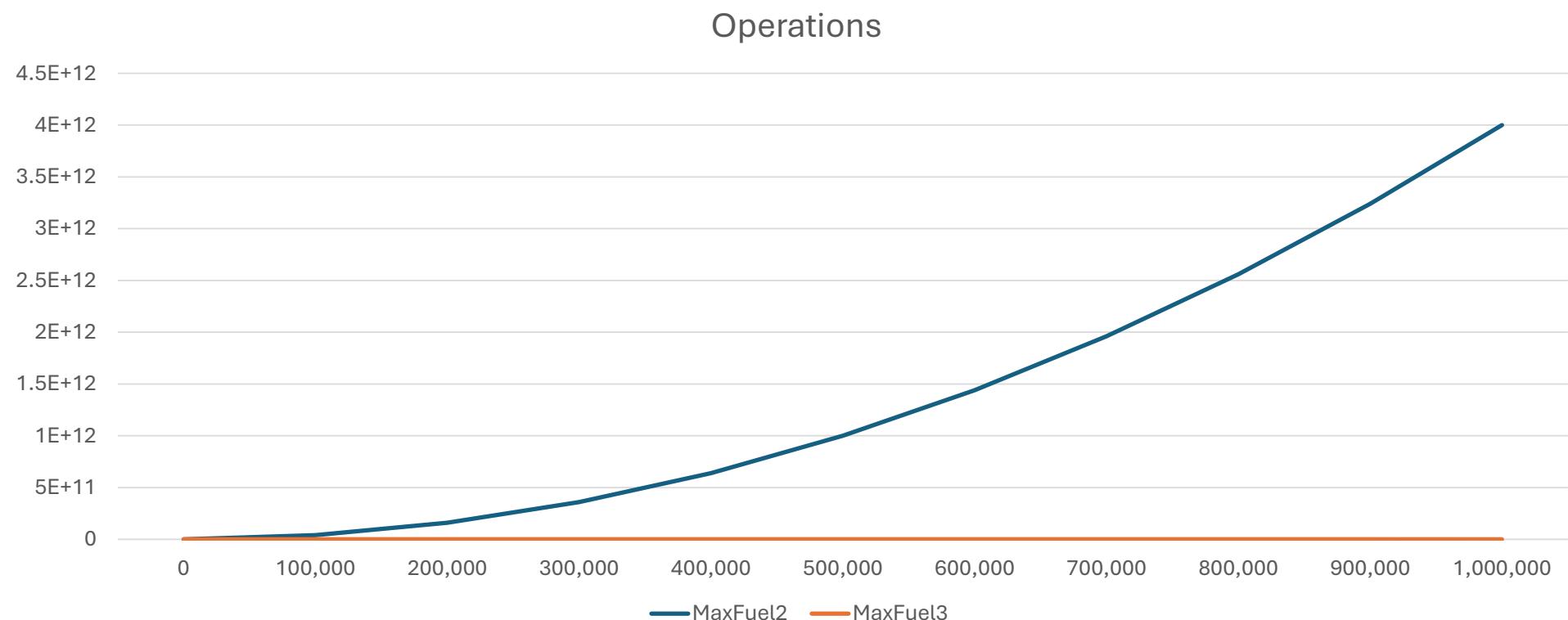
# Operation comparison



# Operation comparison



# Operation comparison



# Which solution is faster?

Which function is smaller?

- a.  $f(n) = 4n^2 + 2n + 4$
- b.  $g(n) = 12n + 8$

# Which solution is faster?

Which function is smaller?

- a.  $f(n) = 4n^2 + 2n + 4$
- b.  $g(n) = 1000000n + 8$

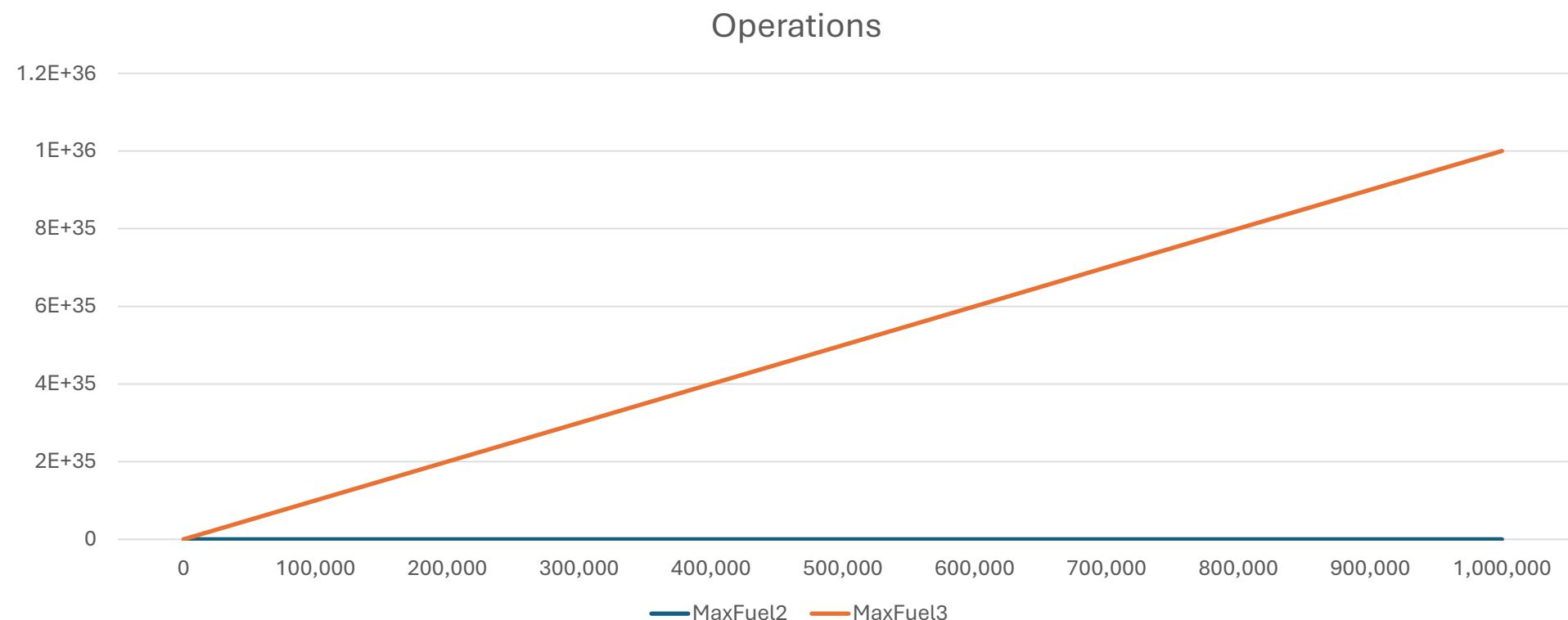
# Which solution is faster?

Which function is smaller?

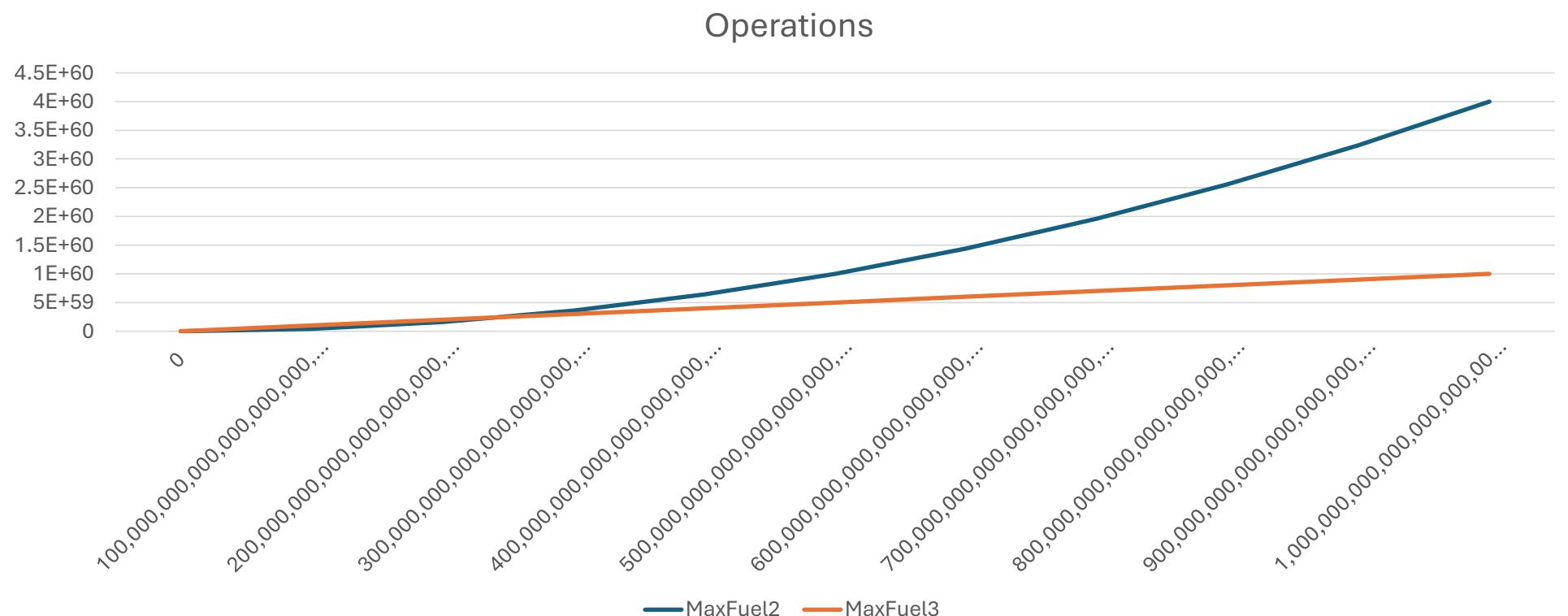
a.  $f(n) = 4n^2 + 2n + 4$

b.  $g(n) = 10000000000000000000000000000000n + 8$

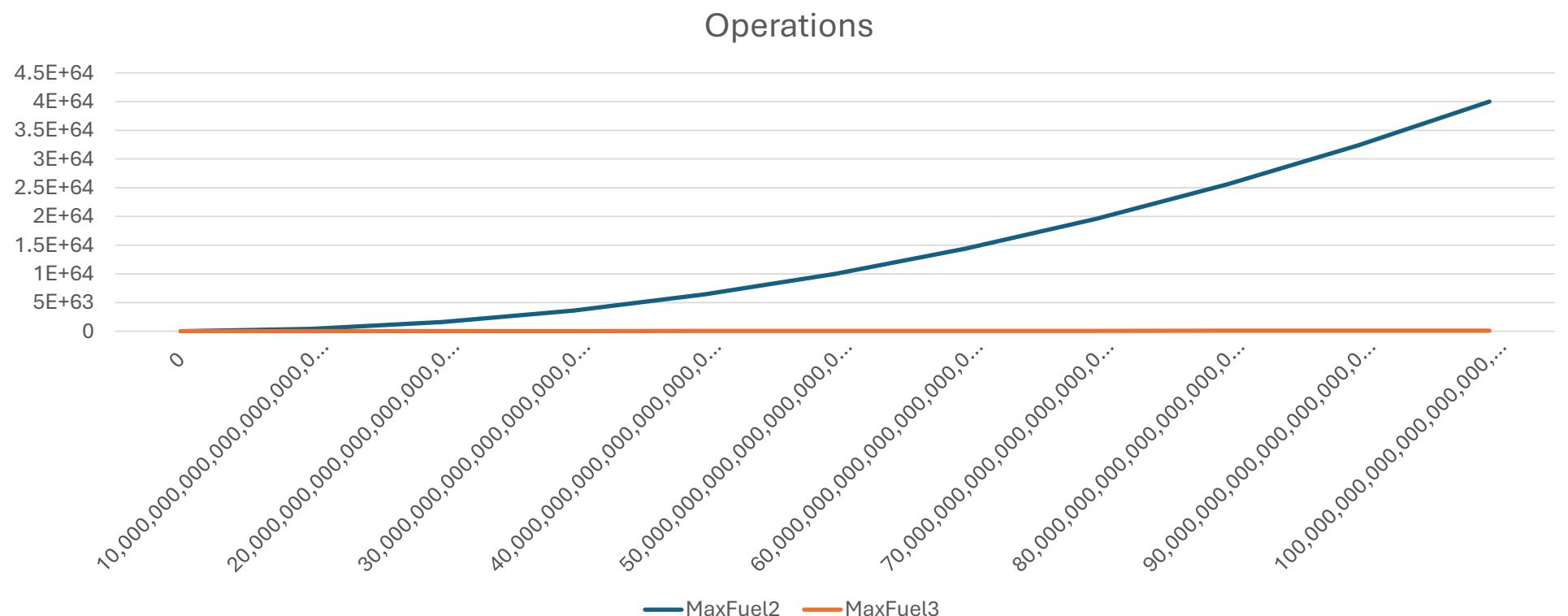
# Operation comparison



# Operation comparison



# Operation comparison



# Drop constant factor

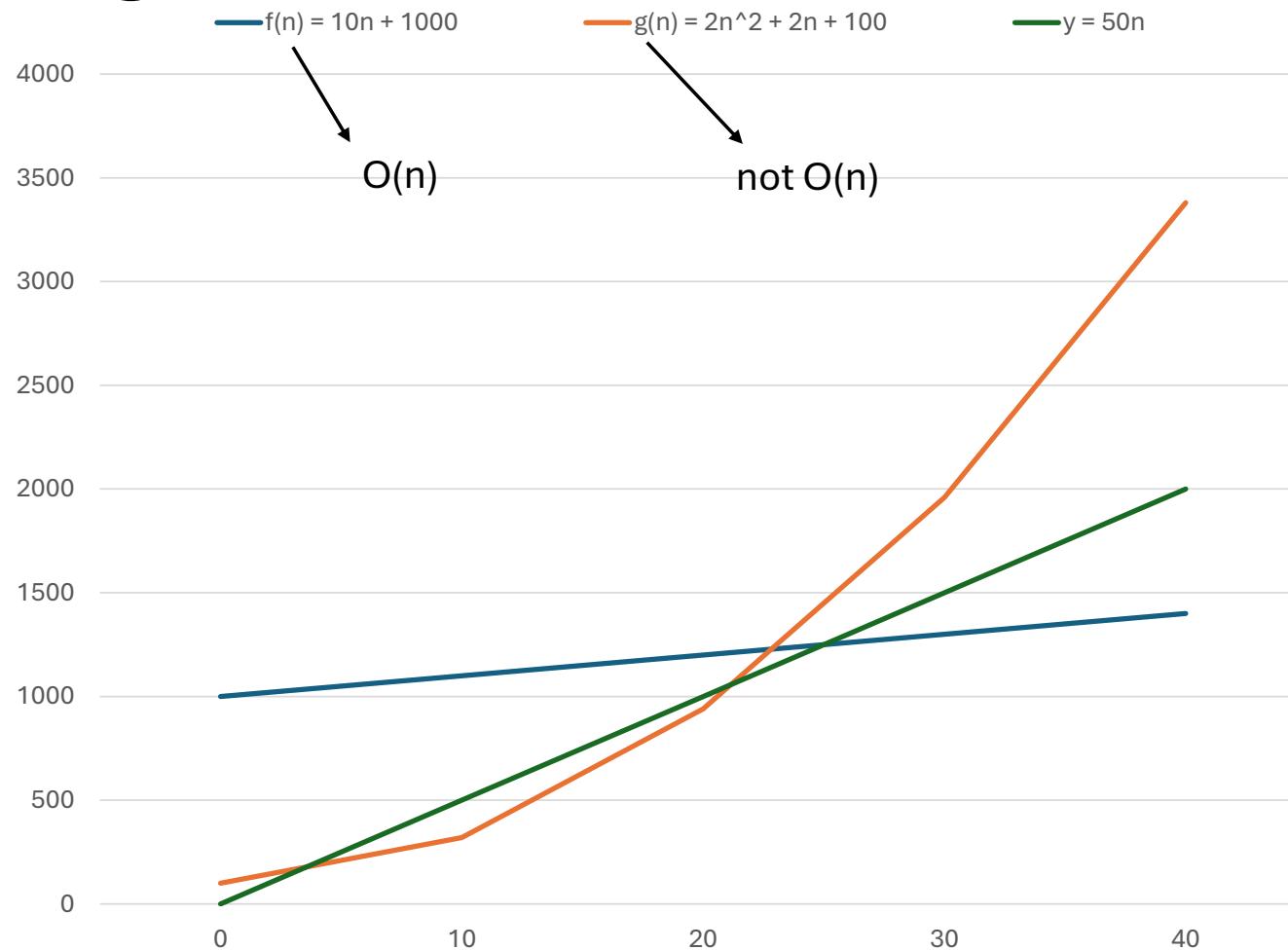
$$\begin{aligned}f(n) &= 4n^2 + 2n + 4 \\&= 4n^2 + 2n + 5 \\&= 5n^2 + 2n + 4 \\&= 1,000n^2 + 1,000,000n + 1,000,000,000\end{aligned}$$

(in this context at least)

# Operation counting revisited

Operations	Code
$c_a$	<pre>int currentMax = 0;</pre>
$c_b n + c_c$	<pre>for(int i = 0; i &lt; distances.length; i++) {</pre>
$c_d n^2 + c_e n + c_f$	<pre>    for (int j = i + 1; j &lt; distances.length; j++) {</pre>
$c_g n^2 + c_h n + c_i$	<pre>        int fuel = Math.abs(distances[i] - distances[j]);</pre>
$c_j n^2 + c_k n + c_l$	<pre>        if (fuel &gt; currentMax) {</pre>
$c_m n^2 + c_n n + c_o$	<pre>            currentMax = fuel;</pre>
$c_p$	<pre>        }</pre>
$c_p$	<pre>}</pre>
Total: $c_0 n^2 + c_1 n + c_2$	<pre>return currentMax;</pre>

# Big-Oh notation



A function  $f(n)$  is  $O(n)$  if and only if we can put a factor before  $n$  such that the result will always be greater than  $f(n)$  for values of  $n$  above some value.

Exercise: write this in predicate logic!  
Start with " $f(n)$  is  $O(n)$   $\Leftrightarrow \dots$ "

Extra exercise: write in predicate logic when a function  $f(n)$  is  $O(n^2)$ ! Write in predicate logic when a function  $f(n)$  is  $O(h(n))$ !

**Focus**

**30 sec**

- Breath in and out 3 times

# Cooling down

10:25-10:30

- Discuss how you are doing with the course
  - What did you learn in this lecture?
  - How are you going to prepare for the next lecture?
  - How are you doing in the course in general?
- Discuss how you are doing with life?
  - How was your weekend?
  - What do you feel right now?
  - What are you looking forward to?
- Forgive me for letting the lecture run late

## ALGORITHMS BY COMPLEXITY

MORE COMPLEX →

LEFTPAD

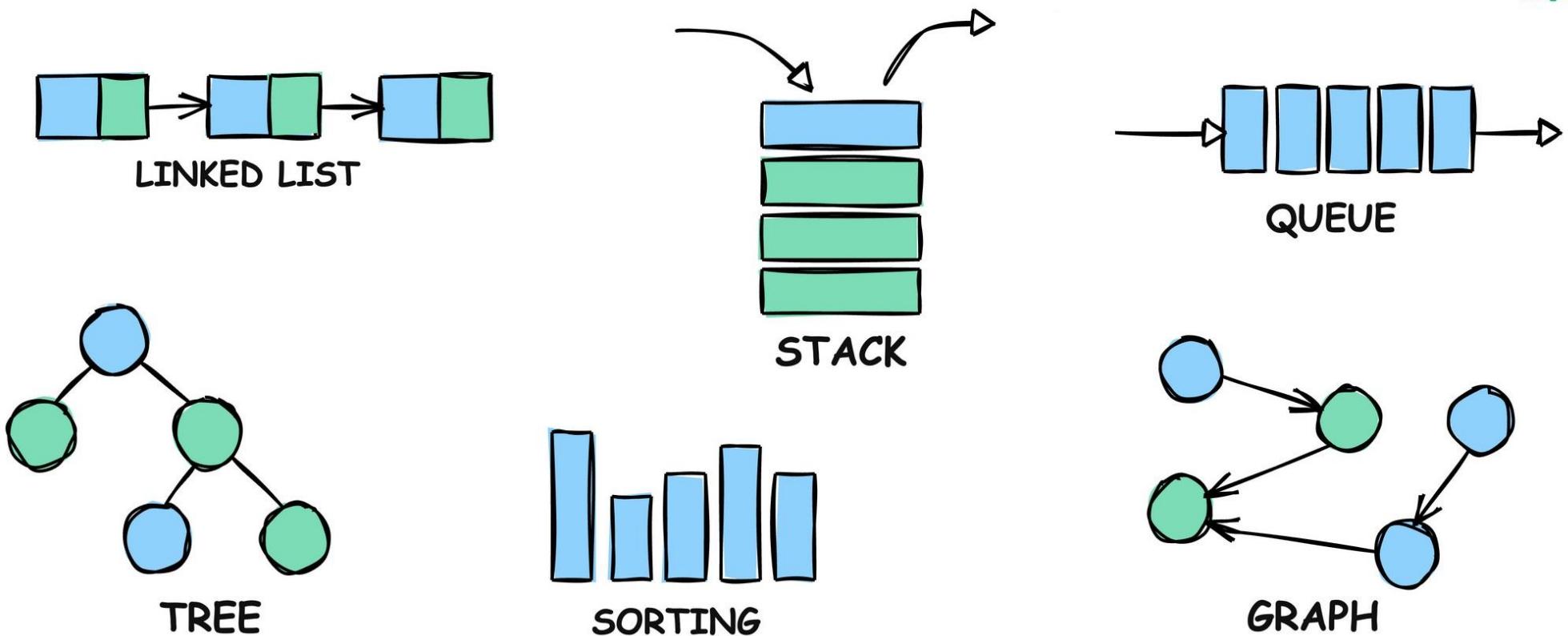
QUICKSORT

GIT  
MERGE

SELF-  
DRIVING  
CAR

GOOGLE  
SEARCH  
BACKEND

SPRAWLING EXCEL SPREADSHEET  
BUILT UP OVER 20 YEARS BY A  
CHURCH GROUP IN NEBRASKA TO  
COORDINATE THEIR SCHEDULING



# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Warming up

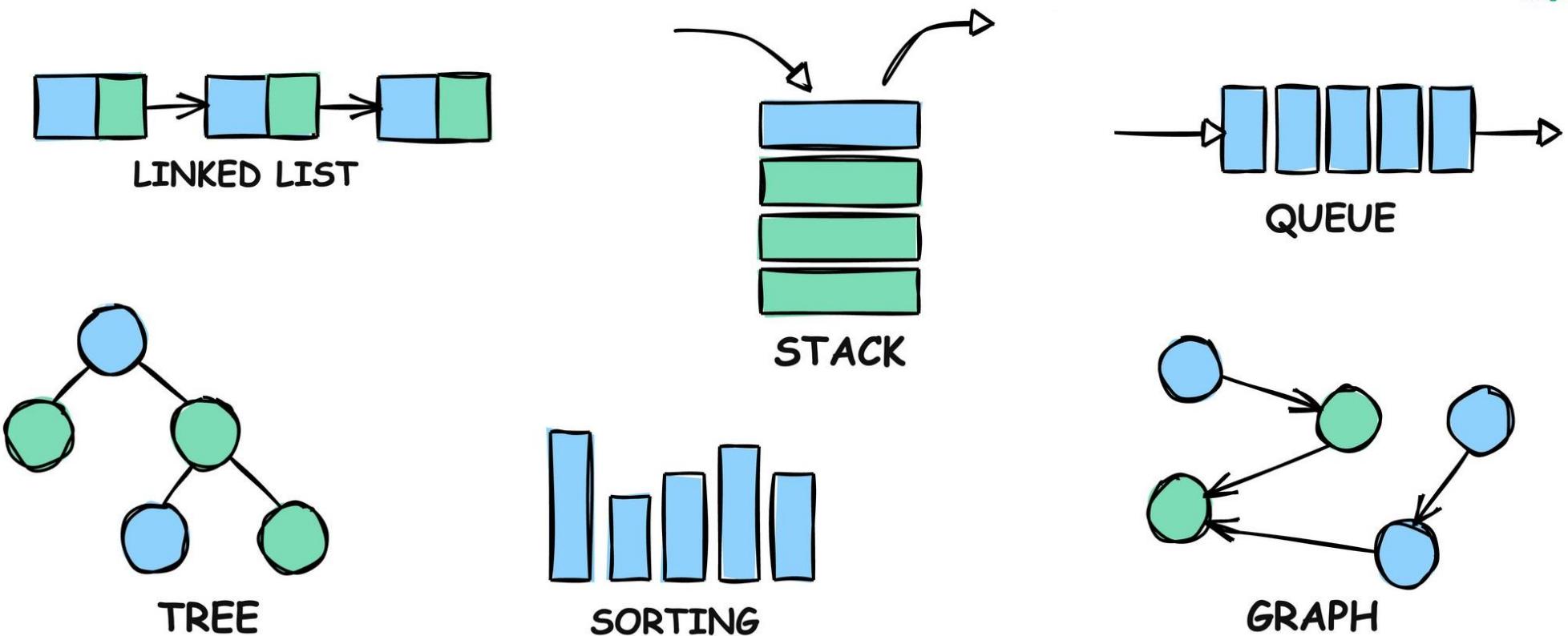
8:45-8:50

- Discuss how you are doing with the course
  - What did you learn in the preparation of this lecture?
  - What do you expect from this lecture?
  - What do you want to have learned after this lecture?
- Discuss how you are doing with life?
  - How was your weekend?
  - What do you feel right now?
  - What are you looking forward to?
- Forgive yourself for coming slightly late
  - (and have the intention to come on time next lecture)

**Focus**

**30 sec**

- Breath in and out 3 times



# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Lab

- This afternoon!
- The structure depends on your mentor group (look at your timetable)
- In Acing ADS, you will solve a challenging problem, with help if you need it
- In the lab, you can work on your homework and ask the TA's questions about it
- In later weeks, you also receive feedback during the lab

# Homework deadline

- Friday 23:59
- You will get the answers to the analysis questions if you have enough points in the remarkable reward register
- If you hand it in before the deadline, you can enqueue to get feedback next week
- You can check your implementation assignments with spec-tests

# Big-Oh notation in predicate logic

A function  $f(n)$  is  $O(n)$  if and only if we can put a factor before  $n$  such that the result will always be greater than  $f(n)$  for values of  $n$  above some value. Write in predicate logic when a function  $f(n)$  is  $O(g(n))$ !

- A.  $\exists c, n_0 (0 < c \wedge 0 < n_0 \wedge (\forall n (n \geq n_0 \Rightarrow g(n) \leq c \cdot n)))$
- B.  $\exists c, n_0 (0 < c \wedge 0 < n_0 \wedge (\forall n (n \geq n_0 \Rightarrow f(n) \leq c \cdot g(n))))$
- C.  $\exists c, n_0 (0 < c \wedge 0 < g(n_0) \wedge (\forall n (g(n) \geq g(n_0) \Rightarrow f(n) \leq c \cdot g(n))))$
- D.  $\exists c, g(n_0) (0 < c \wedge 0 < g(n_0) \wedge (\forall g(n) (g(n) \geq g(n_0) \Rightarrow f(g(n)) \leq c \cdot g(n))))$

# Proving the asymptotic runtime

How would a proof that  $f(n)$  is  $O(g(n))$  look like?

- A. We would show that if there exists a  $c > 0, n_0 > 0$ , then for all  $n, f(n) \leq g(n)$
- B. We would show that if  $c > 0, n_0 > 0$  and  $n \geq n_0$ , then  $f(n) \leq c \cdot g(n)$
- C. We would take an arbitrary  $c > 0$  and  $n_0 > 0$  and show that for each  $n \geq n_0$ , it holds that  $f(n) \leq c \cdot g(n)$
- D. We would choose a  $c > 0$  and an  $n_0 > 0$ , take an arbitrary  $n \geq n_0$ , and show that  $f(n) \leq c \cdot g(n)$

# Proving the asymptotic runtime

What  $c$  and  $n_0$  can we choose to prove that  $2n + 10$  is  $O(n)$ ?

- A.  $c = 0, n_0 = 100$
- B.  $c = 1, n_0 = 35$
- C.  $c = 2, n_0 = 15$
- D.  $c = 3, n_0 = 11$

# Proving the asymptotic runtime

What  $c$  and  $n_0$  can we choose to prove that  $2n^2 + 5n + 49$  is  $O(n^2)$ ?

- A.  $c = 3, n_0 = 10$
- B.  $c = 3, n_0 = 15$
- C.  $c = 4, n_0 = 7$
- D.  $c = 4, n_0 = 10$

# Proving the asymptotic runtime

Prove that  $2n^2 + 5n + 49$  is  $O(n^2)$ !

Proof

Take  $c = 3$ ,  $n_0 = 10$ . Now, we want to prove that for all  $n \geq n_0$ ,  $f(n) \leq c \cdot g(n)$ . In this particular case, we need to show that for all  $n \geq 10$ ,  $2n^2 + 5n + 49 \leq 3n^2$ . We can rewrite this statement to  $2 + \frac{5}{n} + \frac{49}{n^2} \leq 3$  and then to  $\frac{5}{n} + \frac{49}{n^2} \leq 1$ . This is true for  $n = 10$  because  $\frac{5}{10} + \frac{49}{100} \leq 1$ , and for any value of  $n > 10$  because the left side of the equation will only get smaller. Therefore, for all  $n \geq 10$ ,  $2n^2 + 5n + 49 \leq 3n^2$ , so  $2n^2 + 5n + 49$  is  $O(n^2)$ .

# The first important ADS skill!

The screenshot shows a learning platform interface with a central navigation bar and several modules:

- Big-Oh**:  
**Big-Oh proof**  
20 ✘ Helpful Homework: Time Complexity - Proof ⓘ
- Induction & Recursion**:  
**Induction on Recursion**  
This is a link to a skill in another module.  
External
- Analyzing algorithms**:  
**Time complexity**  
5 ✘ Helpful Homework: Code Analysis: Time Complexity I ⓘ  
15 ✘ Helpful Homework: Code Analysis: Time Complexity II ⓘ
- Analyzing algorithms**:  
**Space complexity**  
10 ► Verified Video: Space complexity ⓘ  
5 ✘ Helpful Homework: Space complexity ⓘ  
10 ⓘ Ingenious Implementation: Count repetitions ⓘ
- Analyzing algorithms**:  
**Recursive time complexity**  
16 ► Verified Video: Recursive algorithms ⓘ  
8 ⓘ Written Wisdom: Chapter 5.2  
5 ✘ Helpful Homework: Recurrence equations ⓘ  
30 ✘ Helpful Homework: Manipulating recurrence equations ⓘ  
10 ✘ Helpful Homework: Recursion ⓘ  
5 ✘ Helpful Homework: Code Analysis: Recursion I ⓘ  
5 ✘ Helpful Homework: Code Analysis: Recursion II ⓘ  
10 ⓘ Ingenious Implementation: Recursive Multiplication ⓘ  
5 ⓘ Rapid Reflection: Recursion ⓘ
- Lecture 2**:  
Wed, November 16, 2022 at 10:45–  
Attend Lecture ⓘ  
Post-lecture questions ⓘ
- Analyzing algorithms**:  
**More practice**  
10 ⓘ Ingenious Implementation: Fibonacci 1 ⓘ  
10 ⓘ Ingenious Implementation: Fibonacci 2 ⓘ  
20 ⓘ Ingenious Implementation: Merge two sorted arrays ⓘ  
5 ⓘ Rapid Reflection:  $1 + 1 = 1$  ⓘ
- Analyzing algorithms**:  
**Quadruple Quest**  
80 ⓘ Quadruple Quest ⓘ  
0 ⓘ Input file ⓘ
- Analyzing algorithms**:  
**Challenge**  
This skill has no tasks in this path.

At the bottom, there are two time stamps:

- Lecture 2: Wed, November 16, 2022 at 10:45–
- Homework week 1: Fri, November 18, 2022 at 23:59

# Big-Oh notation

Which of these functions are  $O(n^2)$ ?

- A.  $50n^2 + 100$
- B.  $3n^2 + 5n^3 + 4$
- C.  $n^4$
- D.  $80n + 5$
- E.  $2^n$
- F. 80
- G.  $\log_2 n$

# Tightest bound

What is the tightest bound on  $f(n) = 80$ ?

$O(1)$

# Comparing bounds

Is  $f(n)$  is  $O(n) \Rightarrow f(n)$  is  $O(n^2)$  true for all  $f(n)$ ?

- A. Yes
- B. No

# Relation between bounds

Let  $R$  be a relation such that  $(f, g) \in R$  iff  $f(n)$  is  $O(g(n))$ . Is  $R$  an equivalence relation?

- A. Yes
- B. No,  $R$  is not reflexive
- C. No,  $R$  is not symmetric
- D. No,  $R$  is not transitive
- E. No,  $R$  is not reflexive and not symmetric
- F. No,  $R$  is not reflexive and not transitive
- G. No,  $R$  is not symmetric and not transitive
- H. No,  $R$  is not reflexive, not symmetric and not transitive

# Relation between bounds

Give an order for these bounds, from best to worst.

- A.  $O(n)$
- B.  $O(2n)$
- C.  $O(n^2)$
- D.  $O(n^3)$
- E.  $O(n^2/2)$
- F.  $O(2n)$
- G.  $O(3n)$
- H.  $O(\log_2 n)$
- I.  $O(n \cdot \log_2 n)$
- J.  $O(1)$

# Big-Omega

A function  $f(n)$  is  $\Omega(g(n))$  if and only if we can put a factor before n such that the result will always be smaller than  $f(n)$  for values of n above some value. How can this be written in predicate logic?

- A.  $\exists c, \exists n_0 (0 < c \wedge 0 < n_0 \wedge (\forall n (n \geq n_0 \Rightarrow f(n) \geq c \cdot g(n))))$
- B.  $\exists c, \exists n_0 (0 < c \wedge 0 < n_0 \wedge (\forall n (n \leq n_0 \Rightarrow f(n) \leq c \cdot g(n))))$
- C.  $\exists c, \exists n_0 (0 < c \wedge 0 < n_0 \wedge (\forall n (n \geq n_0 \Rightarrow f(n) \leq c \cdot g(n))))$
- D.  $\forall c, \forall n_0 (0 < c \wedge 0 < n_0 \wedge (\forall n (n < n_0 \Rightarrow f(n) \geq c \cdot g(n))))$

# Big-Omega notation

Which of these functions are  $\Omega(n^2)$ ?

- A.  $50n^2 + 100$
- B.  $3n^2 + 5n^3 + 4$
- C.  $n^4$
- D.  $80n + 5$
- E.  $2^n$
- F. 80
- G.  $\log_2 n$

# Big-Theta notation

Which of these functions are  $\Theta(n^2)$ ?

- A.  $50n^2 + 100$
- B.  $3n^2 + 5n^3 + 4$
- C.  $n^4$
- D.  $80n + 5$
- E.  $2^n$
- F. 80
- G.  $\log_2 n$

# Relation between bounds

Let  $R$  be a relation such that  $(f, g) \in R$  iff  $f(n)$  is  $\Theta(g(n))$ . Is  $R$  an equivalence relation?

- A. Yes
- B. No,  $R$  is not reflexive
- C. No,  $R$  is not symmetric
- D. No,  $R$  is not transitive
- E. No,  $R$  is not reflexive and not symmetric
- F. No,  $R$  is not reflexive and not transitive
- G. No,  $R$  is not symmetric and not transitive
- H. No,  $R$  is not reflexive, not symmetric and not transitive

# Big-Oh in practice

What is the tightest bound on the runtime of the method below?

```
public static int f(int[] x) {  
    int sum;  
    switch(x.length) {  
        case 1:  
            sum = 2;  
            break;  
        default:  
            sum = 0;  
    }  
    for(int i : x) {  
        sum += Math.sqrt(i < 10 ? i : i >>> 1);  
    }  
    return sum;  
}
```

$c_a$   
 $c_b$   
 $c_c$   
 $c_d$   
 $c_e$   
 $c_f$   
 $c_g$   
 $c_h n + c_i$   
 $c_j n$   
 $c_k$

Total:  $c_1 n + c_0$

# The second important ADS skill!

Lecture 2 Wed, November 16, 2022 at 10:45

**Big-Oh**  
**Big-Oh proof**  
20 ✘ Helpful Homework: Time Complexity - Proof ⏺

**Induction & Recursion**  
**Induction on Recursion**  
This is a link to a skill in another module.  
External

**Analyzing algorithms**  
**Time complexity**  
5 ✘ Helpful Homework: Code Analysis: Time Complexity I  
15 ✘ Helpful Homework: Code Analysis: Time Complexity II

**Analyzing algorithms**  
**Space complexity**  
10 ► Verified Video: Space complexity ⏺  
5 ✘ Helpful Homework: Space complexity ⏺  
10 📈 Ingenious Implementation: Count repetitions ⏺

**Analyzing algorithms**  
**Recursive time complexity**  
16 ► Verified Video: Recursive algorithms ⏺  
8 ✑ Written Wisdom: Chapter 5.2  
5 ✘ Helpful Homework: Recurrence equations ⏺  
30 ✘ Helpful Homework: Manipulating recurrence equations  
10 ✘ Helpful Homework: Recursion ⏺  
5 ✘ Helpful Homework: Code Analysis: Recursion I ⏺  
5 ✘ Helpful Homework: Code Analysis: Recursion II ⏺  
10 📈 Ingenious Implementation: Recursive Multiplication  
5 📈 Rapid Reflection: Recursion ⏺

**Analyzing algorithms**  
**Lecture 2**  
90 ✑ Attend Lecture  
10 ✑ Post-lecture questions

**Analyzing algorithms**  
**More practice**  
10 📈 Ingenious Implementation: Fibonacci 1 ⏺  
10 📈 Ingenious Implementation: Fibonacci 2 ⏺  
20 📈 Ingenious Implementation: Merge two sorted arrays  
5 📈 Rapid Reflection:  $1 + 1 = 1$  ⏺

**Analyzing algorithms**  
**Quadruple Quest**  
80 📈 Quadruple Quest ⏺  
0 📈 Input file ⏺

**Analyzing algorithms**  
**Challenge**  
This skill has no tasks in this path

Homework week 1 Fri, November 18, 2022 at 23:59

# An alternative river problem

- River with towns along it over a length of 20 km
- Each town is characterized with the distance to the sea travelling along the river
- List<Integer> containing this distance (between 0 and 20) of all towns (**in a sorted order**)
- The task is to find the number of towns in the first 10 km

Design an algorithm which has as input argument the list of distances as a List<Integer> and outputs the number of towns that have a value of at most 10. Try to find an algorithm that is faster than  $O(n)$ .

Give a tight bound on the runtime of this algorithm.

# If we don't have a sorted list

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32	92	68	92	34	2	50	35	73	11	50	43	49	36	41	11

- If we would have no information about the elements, we would have to look at all of them to be sure that a distance is the first which is at least 10
- Since there are  $n$  elements, a tight bound faster than  $O(n)$  won't be possible

# If we do have a sorted list

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

- There are 16 boxes containing an integer, and these boxes are sorted on their integer
- You have to find the first box with an integer of at least 10
- What box would you first open?

# If we do have a sorted list

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
6	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

- If we open the first box:
  - If it is at least 10, we are lucky and we are done!
  - Otherwise, we only know it is not the first box, but we don't gain any more information
  - Repeating this strategy, we have to open  $n$  boxes in the worst case

# If we do have a sorted list

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
X	X	X	6	?	?	?	?	?	?	?	?	?	?	?	?

- If we open the fourth box:
  - If it is at least 10, it could be this one or one with a lower index
  - Otherwise, it is one with a higher index
  - So in the worst case, we eliminated 4 possibilities (12 left)
  - By opening the middle box (seventh) box, we have the best worst case

# If we do have a sorted list

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
?	?	?	?	?	?	?	18	X	X	X	X	X	X	X	X

- If we open the middle box:
  - If it is at least 10, it could be this one or one with a lower index
  - Otherwise, it is one with a higher index
  - In either case, we eliminated half of the possibilities

If we repeat this strategy, how many boxes do we have to open?

# Binary Search

- `BinarySearch(data, target, low, high):`
  - $\text{mid} \leftarrow (\text{low} + \text{high}) / 2$
  - if  $\text{data}[\text{mid}] < \text{target}$ :
    - return `BinarySearch(data, target, mid + 1, high)`
  - else:
    - return `BinarySearch(data, target, low, mid)`

Aren't we forgetting something?

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	3	3	4	5	5	7	8	9	9	10	10	12	15	17	17

# Binary Search

- `BinarySearch(data, target, low, high):`
  - if `low = high:`
    - return `low`
  - `mid ← (low + high) / 2`
  - if `data[mid] < target:`
    - return `BinarySearch(data, target, mid + 1, high)`
  - else:
    - return `BinarySearch(data, target, low, mid)`

$$\begin{aligned}c_a \\ c_b \\ c_c \\ c_d \\ c_e + T(n/2) \\ c_f + T(n/2)\end{aligned}$$

0	1	2	3	4	5	6	7	8	9	10
1	3	3	4	5	5	7	8	9	9	10

Total:  
 $T(n) = c_1 + T(n/2)$   
 $T(1) = c_0$

# Unfolding a recurrence equation

$$\begin{aligned} T(n) &= c_1 + T(\frac{n}{2}) && \text{if } n > 1 \\ T(n) &= c_0 && \text{else} \end{aligned}$$

$T(n) =$	$c_1 + T(\frac{n}{2})$	by $T(n) = c_1 + T(\frac{n}{2})$
	$c_1 + (c_1 + T(\frac{n}{4}))$	by $T(\frac{n}{2}) = c_1 + T(\frac{n}{4})$
	$2c_1 + T(\frac{n}{4})$	by arithmetic
	$2c_1 + (c_1 + T(\frac{n}{8}))$	by $T(\frac{n}{4}) = c_1 + T(\frac{n}{8})$
	$3c_1 + T(\frac{n}{8})$	by arithmetic
	$3c_1 + (c_1 + T(\frac{n}{16}))$	by $T(\frac{n}{8}) = c_1 + T(\frac{n}{16})$
	$4c_1 + T(\frac{n}{16})$	by arithmetic
	$4c_1 + (c_1 + T(\frac{n}{32}))$	by $T(\frac{n}{2}) = c_1 + T(\frac{n}{4})$
	$5c_1 + T(\frac{n}{32})$	by arithmetic
	$5c_1 + (c_1 + T(\frac{n}{32}))$	by $T(\frac{n}{2}) = c_1 + T(\frac{n}{4})$
	$6c_1 + T(\frac{n}{64})$	by arithmetic
...		
	$k * c_1 + T(\frac{n}{2^k})$	repeat $k$ times
	$\log_2 n * c_1 + T(\frac{n}{2^{\log_2 n}})$	by letting $k = \log_2 n$
	$\log_2 n * c_1 + c_0$	by arithmetic and $T(n) = c_0$

# The third important ADS skill!

Lecture 2 Wed, November 16, 2022 at 10:45

**Big-Oh**  
**Big-Oh proof**  
20 ✘ Helpful Homework: Time Complexity - Proof ⏺

**Induction & Recursion**  
**Induction on Recursion**  
This is a link to a skill in another module.  
External

**Analyzing algorithms**  
**Time complexity**  
5 ✘ Helpful Homework: Code Analysis: Time Complexity I  
15 ✘ Helpful Homework: Code Analysis: Time Complexity II

**Analyzing algorithms**  
**Space complexity**  
10 ► Verified Video: Space complexity ⏺  
5 ✘ Helpful Homework: Space complexity ⏺  
10 📈 Ingenious Implementation: Count repetitions ⏺

**Analyzing algorithms**  
**Recursive time complexity**  
16 ► Verified Video: Recursive algorithms ⏺  
8 📈 Written Wisdom: Chapter 5.2  
5 ✘ Helpful Homework: Recurrence equations ⏺  
30 ✘ Helpful Homework: Manipulating recurrence equations  
10 ✘ Helpful Homework: Recursion ⏺  
5 ✘ Helpful Homework: Code Analysis: Recursion I ⏺  
5 ✘ Helpful Homework: Code Analysis: Recursion II ⏺  
10 📈 Ingenious Implementation: Recursive Multiplication  
5 📈 Rapid Reflection: Recursion ⏺

**Analyzing algorithms**  
**Lecture 2**  
90 📈 Attend Lecture  
10 📈 Post-lecture questions

**Analyzing algorithms**  
**More practice**  
10 📈 Ingenious Implementation: Fibonacci 1 ⏺  
10 📈 Ingenious Implementation: Fibonacci 2 ⏺  
20 📈 Ingenious Implementation: Merge two sorted arrays  
5 📈 Rapid Reflection:  $1 + 1 = 1$  ⏺

**Analyzing algorithms**  
**Quadruple Quest**  
80 📈 Quadruple Quest ⏺  
0 📈 Input file ⏺

**Analyzing algorithms**  
**Challenge**  
This skill has no tasks in this path

Homework week 1 Fri, November 18, 2022 at 23:59

# Space complexity

What is the tightest bound on the space of the method below?

```
public static int[] f(int[] arr) {  
    int[] ret = new int[arr.length];           can + cb  
    for(int i = 0; i < arr.length; i++) {      cc  
        int[] x = new int[arr.length];          cdn + ce  
        for(int b : arr) {                    cf  
            x[i] = arr[i] * b;  
        }  
        Arrays.sort(x);                      ? (but certainly at most linear)  
        if(x[0] > ret[0]) {  
            ret = x;  
        }  
    }  
    return ret;  
}
```

Total:  $c_1n + c_0$

# Binary Search

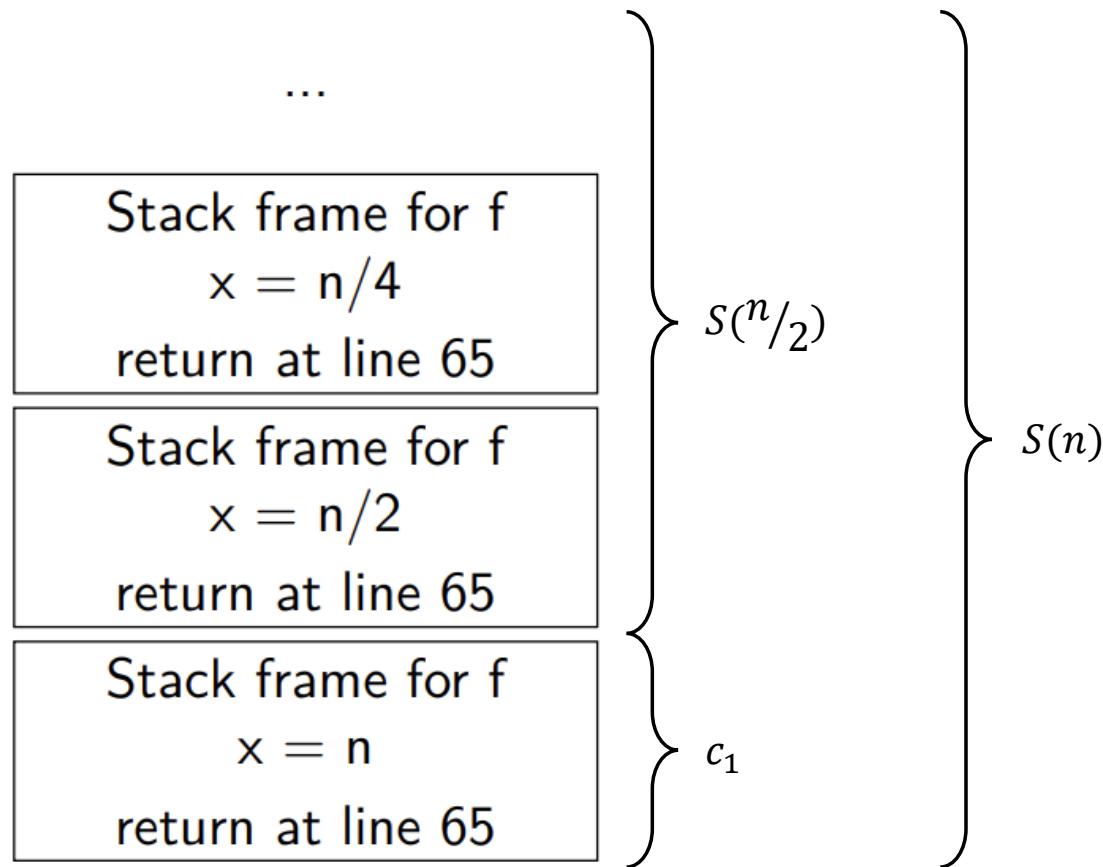
- `BinarySearch(data, target, low, high):`  $c_a$
- if `low = high:`
  - return `low`
- `mid ← (low + high) / 2`  $c_b$
- if `data[mid] < target:`
  - return `BinarySearch(data, target, mid + 1, high)`  $S(n/2)$
- else:
  - return `BinarySearch(data, target, low, mid)`  $S(n/2)$

Total:

$$S(n) = c_1 + T(n/2)$$

$$S(1) = c_0$$

# Visualization of space usage in Binary Search



$$S(n) = c_1 + T(n/2)$$
$$S(1) = c_0$$

# The fourth important ADS skill!

Lecture 2 Wed, November 16, 2022 at 10:45

**Big-Oh**  
**Big-Oh proof**  
20 ✘ Helpful Homework: Time Complexity - Proof ⏺

**Induction & Recursion**  
**Induction on Recursion**  
This is a link to a skill in another module.  
External

**Analyzing algorithms**  
**Time complexity**  
5 ✘ Helpful Homework: Code Analysis: Time Complexity I  
15 ✘ Helpful Homework: Code Analysis: Time Complexity II

**Analyzing algorithms**  
**Space complexity**  
10 ► Verified Video: Space complexity ⏺  
5 ✘ Helpful Homework: Space complexity ⏺  
10 📈 Ingenious Implementation: Count repetitions ⏺

**Analyzing algorithms**  
**Recursive time complexity**  
16 ► Verified Video: Recursive algorithms ⏺  
8 📈 Written Wisdom: Chapter 5.2  
5 ✘ Helpful Homework: Recurrence equations ⏺  
30 ✘ Helpful Homework: Manipulating recurrence equations  
10 ✘ Helpful Homework: Recursion ⏺  
5 ✘ Helpful Homework: Code Analysis: Recursion I ⏺  
5 ✘ Helpful Homework: Code Analysis: Recursion II ⏺  
10 📈 Ingenious Implementation: Recursive Multiplication ⏺  
5 📈 Rapid Reflection: Recursion ⏺

**Analyzing algorithms**  
**Lecture 2**  
90 📈 Attend Lecture  
10 📈 Post-lecture questions

**Analyzing algorithms**  
**More practice**  
10 📈 Ingenious Implementation: Fibonacci 1 ⏺  
10 📈 Ingenious Implementation: Fibonacci 2 ⏺  
20 📈 Ingenious Implementation: Merge two sorted arrays  
5 📈 Rapid Reflection:  $1 + 1 = 1$  ⏺

**Analyzing algorithms**  
**Quadruple Quest**  
80 📈 Quadruple Quest ⏺  
0 📈 Input file ⏺

**Analyzing algorithms**  
**Challenge**  
This skill has no tasks in this path

Homework week 1 Fri, November 18, 2022 at 23:59

# Runtime and space complexity

Is the runtime complexity always the same as the space complexity?

- A. Yes
- B. No, the runtime complexity can be higher than the space complexity
- C. No, the space complexity can be higher than the runtime complexity
- D. No, either of them can be lower than the other

**Focus**

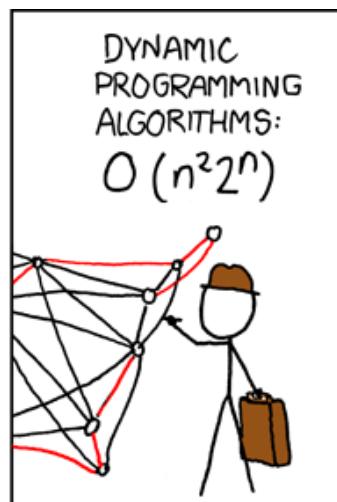
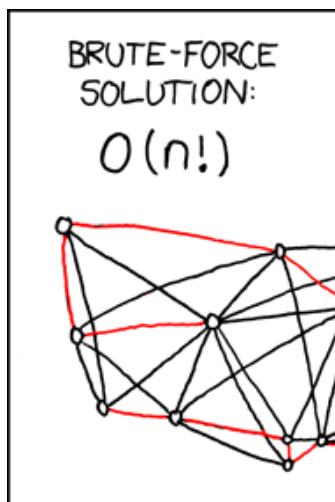
**30 sec**

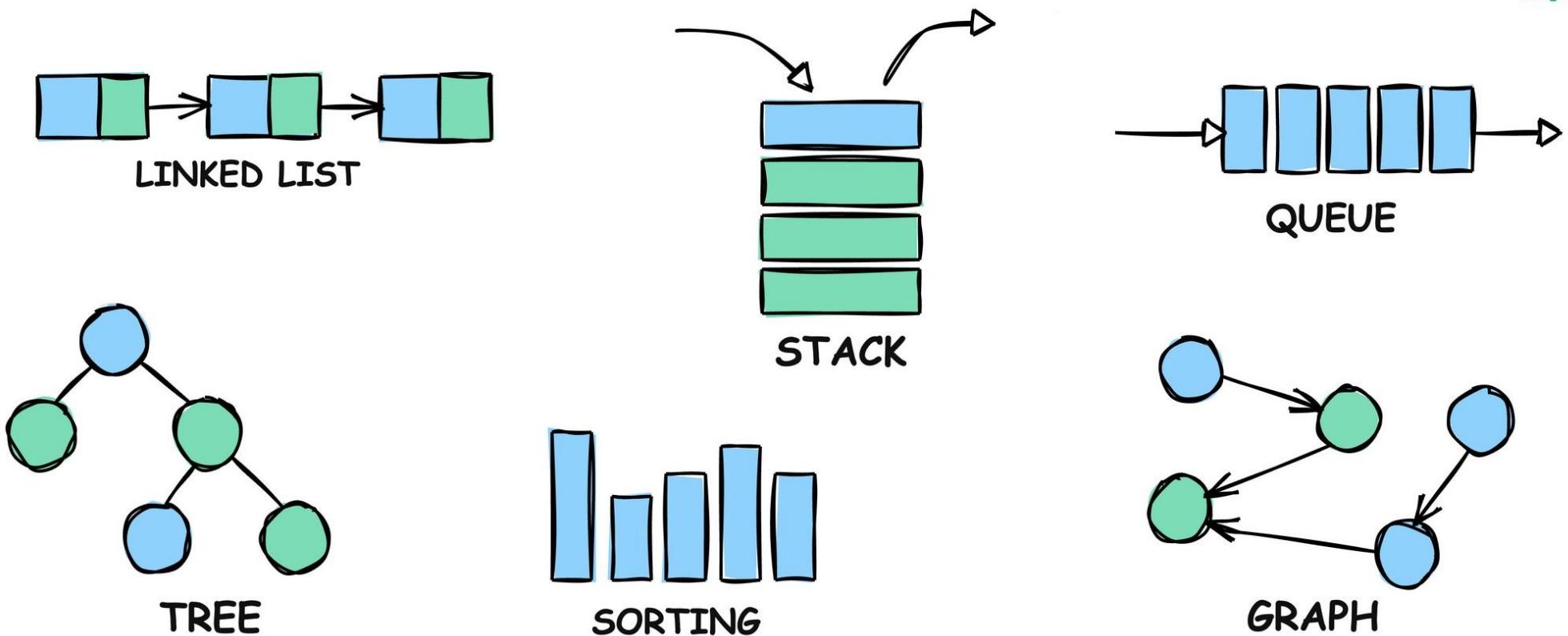
- Breath in and out 3 times

# Cooling down

10:25-10:30

- Discuss how you are doing with the course
  - What did you learn in this lecture?
  - How are you going to prepare for the next lecture?
  - How are you doing in the course in general?
- Discuss how you are doing with life?
  - How was your weekend?
  - What do you feel right now?
  - What are you looking forward to?
- Forgive me for letting the lecture run late





# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Warming up

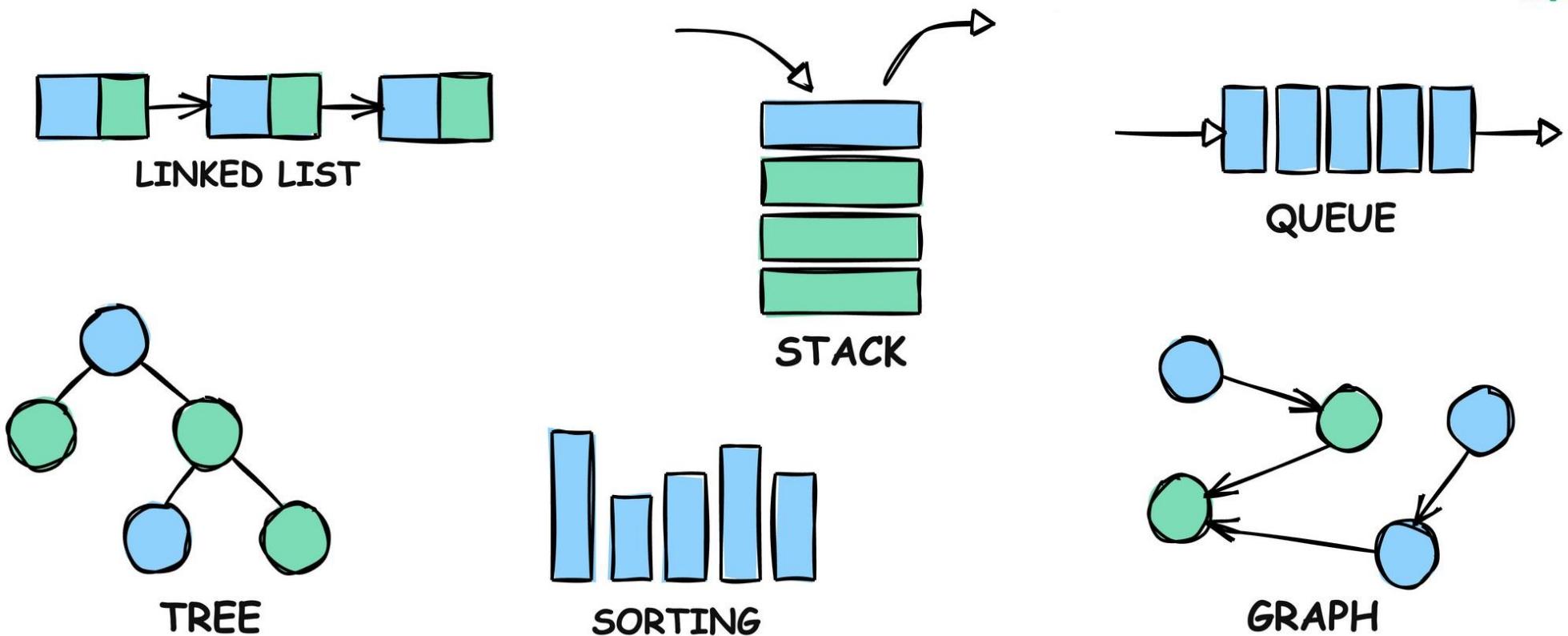
8:45-8:50

- Discuss how you are doing with the course
  - What did you learn in the preparation of this lecture?
  - What do you expect from this lecture?
  - What do you want to have learned after this lecture?
- Discuss how you are doing with life?
  - How was your weekend?
  - What do you feel right now?
  - What are you looking forward to?
- Forgive yourself for coming slightly late
  - (and have the intention to come on time next lecture)

**Focus**

**30 sec**

- Breath in and out 3 times



# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Runtime and space complexity

Is the runtime complexity always the same as the space complexity?

- A. Yes
- B. No, the runtime complexity can be higher than the space complexity
- C. No, the space complexity can be higher than the runtime complexity
- D. No, either of them can be lower than the other

# Space complexity

What is the tightest bound on the space of the method below (you can assume the SLL class is a singly linked list?)

```
public static void f(SLL list) {  
    for(int i = 0; i < list.size(); i++) {  
        SLL newList = new SLL();  
        newList.addLast(list.get(i));  
        list.addLast(list.get(i));  
        list.removeFirst();  
    }  
}
```

# Functionality

What does the algorithm below do?

```
public static void f(int[] arr) {  
    if(arr.length > 1) {  
        int[] newArr = new int[arr.length-1];  
        for(int i = 0; i < newArr.length; i++) {  
            newArr[i] = arr[i];  
        }  
        f(newArr);  
        int i = 0;  
        while(i < newArr.length && newArr[i] < arr[arr.length-1]) {  
            arr[i] = newArr[i];  
            i++;  
        }  
        arr[i] = arr[arr.length-1];  
        while(i+1 < arr.length) {  
            arr[i+1] = newArr[i];  
            i++;  
        }  
    }  
}
```

# Time complexity

What is the time complexity of the algorithm below?

```
public static void f(int[] arr) {  
    if(arr.length > 1) {  
        int[] newArr = new int[arr.length-1];  
        for(int i = 0; i < newArr.length; i++) {  
            newArr[i] = arr[i];  
        }  
        f(newArr);  
        int i = 0;  
        while(i < newArr.length && newArr[i] < arr[arr.length-1]) {  
            arr[i] = newArr[i];  
            i++;  
        }  
        arr[i] = arr[arr.length-1];  
        while(i+1 < arr.length) {  
            arr[i+1] = newArr[i];  
            i++;  
        }  
    }  
}
```

# Insertion sort

Given the list [7, 2, 6, 4, 9, 0, 3, 5, 1, 8]. If you start sorting this list using insertion sort, but stop in the middle of the algorithm, how would the list look like?

- A. [0, 1, 2, 3, 4, 7, 6, 9, 5, 8]
- B. [0, 1, 2, 6, 4, 3, 5, 7, 8, 9]
- C. [2, 4, 6, 7, 9, 0, 1, 3, 5, 8]
- D. [2, 4, 6, 7, 9, 0, 3, 5, 1, 8]

# Insertion sort

Why does the following idea for insertion sort not work?

- Given a list *list*
- Make a new list *newList*
- For each element *e* in *list*:
  - In *newList*, find the highest value that is lower than *e*
  - If there is such value:
    - Insert *e* behind that value
  - Else:
    - Insert *e* at the front of *newList*

# List method complexities

What are the complexities of the following methods for an array, a singly-linked list, a circularly-linked list, and a doubly linked list?

- Access element  $i$
- Changing element  $i$
- Adding an element to the head
- Adding an element to the tail
- Adding an element at index  $i$
- Removing an element at the head
- Removing an element at the tail
- Removing an element at index  $i$
- Concatenating 2 lists
- Cloning the list

1

## Midterm exam 18-19 - MCQ 9 (1 Point)

(1 point) Consider a time-efficient algorithm to remove the middle element of a sequence  $S$  with  $n$  elements. Assume that the middle element is the element at position  $n/2$ , where  $/$  denotes the integer division. Example: if  $S = \{1, 2, 3, 4, 5\}$ , the algorithm should change  $S$  into  $\{1, 2, 4, 5\}$ . What is the tightest worst-case **time** complexity of removing all elements of  $S$  by calling such algorithm repeatedly until  $S$  is empty, if  $S$  is implemented using an array or a singly-linked list?

- A. **array:**  $\mathcal{O}(n)$     **singly-linked list:**  $\mathcal{O}(n)$
- B. **array:**  $\mathcal{O}(n)$     **singly-linked list:**  $\mathcal{O}(n^2)$
- C. **array:**  $\mathcal{O}(n^2)$     **singly-linked list:**  $\mathcal{O}(n)$
- D. **array:**  $\mathcal{O}(n^2)$     **singly-linked list:**  $\mathcal{O}(n^2)$

1

Midterm exam 18-19 - MCQ 9  
(1 Point)

(1 point) Consider a time-efficient algorithm to remove the middle element of a sequence  $S$  with  $n$  elements. Assume that the middle element is the element at position  $n/2$ , where  $/$  denotes the integer division. Example: if  $S = \{1, 2, 3, 4, 5\}$ , the algorithm should change  $S$  into  $\{1, 2, 4, 5\}$ . What is the tightest worst-case **time** complexity of removing all elements of  $S$  by calling such algorithm repeatedly until  $S$  is empty, if  $S$  is implemented using an array or a singly-linked list?

**Idea:** call middle element removal repeatedly.

First look into removing middle element once.

**Idea:** call middle element removal repeatedly.

**What is the time complexity of removing a middle element once?.**

**Array:** remove element at index  $n/2$  :

- **Find:** calculate index of first element to be shifted Done in  $O(1)$  time.
- **Remove:** shift elements in indices  $n/2 + 1$  to  $n$  backwards by one position. Done in  $O(n)$  time.

Remove by shifting

{1, 2, 3, 4, 5, 6}

<- <- <-      Shift  $n - n/2 - 1$  elements backwards      by one position. This is approx.  $n/2$ .

**Idea:** call middle element removal repeatedly.

## What is the time complexity of removing a middle element once?.

**Array:** remove element at index  $n/2$ :

- **Find:** calculate index of first element to be shifted  
Done in  $O(1)$  time.
- **Remove:** shift elements in indices  $n/2 + 1$  to  $n$  backwards by one position. Done in  $O(n)$  time.

Remove by shifting

{1, 2, 3, 4, 5, 6}

<- <- <-      Shift  $n - n/2 - 1$  elements  
backwards      by one position. This is approx.  $n/2$ .

**SLL:** remove element at position  $n/2$ :

- **Find:** traverse list while counting until we find node at position  $n/2 - 1$ . Done in  $O(n)$  time.
- **Remove:** update next reference of node at position  $n/2 - 1$  to point to node at  $n/2 + 1$ .  $O(1)$  time.

Find by traversal

head -> 1 -> 2 -> 3 -> 4 -> 5 -> 6

-> ->

Follow  $n/2$  references

(head +  $n/2 - 1$  next  
references. Approx.  $n/2$ .)

**Idea:** call middle element removal repeatedly.

**What is the time complexity of removing a middle element once from a sequence of size  $n$ ?  $O(n)$  for both array and SLL**

**Array:** remove element at index  $n/2$ :

- **Find:** calculate index of first element to be shifted  
Done in  $O(1)$  time.
- **Remove:** shift elements in indices  $n/2 + 1$  to  $n$  backwards by one position. Done in  $O(n)$  time.

Remove by shifting

{1, 2, 3, 4, 5, 6}

<- <- <-      Shift  $n - n/2 - 1$  elements  
backwards      by one position. This is approx.  $n/2$ .

**SLL:** remove element at position  $n/2$ :

- **Find:** traverse list while counting until we find node at position  $n/2 - 1$ . Done in  $O(n)$  time.
- **Remove:** update next reference of node at position  $n/2 - 1$  to point to node at  $n/2 + 1$ .  $O(1)$  time.

Find by traversal

head -> 1 -> 2 -> 3 -> 4 -> 5 -> 6

-> ->

Follow  $n/2$  references

(head +  $n/2 - 1$  next  
references. Approx.  $n/2$ .)

Midterm exam 18-19 - MCQ 9  
(1 Point)

(1 point) Consider a time-efficient algorithm to remove the middle element of a sequence  $S$  with  $n$  elements. Assume that the middle element is the element at position  $n/2$ , where  $/$  denotes the integer division. Example: if  $S = \{1, 2, 3, 4, 5\}$ , the algorithm should change  $S$  into  $\{1, 2, 4, 5\}$ . What is the tightest worst-case time complexity of removing all elements of  $S$  by calling such algorithm repeatedly until  $S$  is empty, if  $S$  is implemented using an array or a singly-linked list?

**Array:** remove element at index  $k/2$ :

**Runtime of one middle element removal:  $O(k)$ .**

For multiple removals:

Remove by shifting

$\{1, 2, 3, 4, 5, 6\}$

$<- <- <-$

$\{1, 2, 4, 5, 6\}$

$<- <-$

$\{1, 2, 5, 6\}$

$<- <-$

...

Iteration 1: list size  $n$

Iteration 2: list size  $n - 1$

Iteration 3: list size  $n - 2$

...

Iteration  $n$ : list size 1

**Runtime of all removals together:**

Find by traversal

head  $\rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$

$\rightarrow \rightarrow$

head  $\rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6$

$\rightarrow \rightarrow$

head  $\rightarrow 1 \rightarrow 2 \rightarrow 5 \rightarrow 6$

$\rightarrow$

...

$$\sum_{k=1}^n k/2 = \frac{n^2 + n}{8}$$

(ignoring constants)

**Runtime**  
 **$O(n^2)$**

What is the tightest worst-case time complexity of removing an element from an SLL?

**Scenario 1:** We are given a reference to the node to be removed.

**Scenario 2:** We are given the element to be removed.

What is the tightest worst-case time complexity of removing an element from an SLL with  $n$  elements?

Scenario 1: We are given a reference to the node to be removed.

Scenario 2: We are given the element to be removed.

Example: removal of node 5 from the list below, if we already know where to remove.

head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6

To remove node 5 we need a reference to its preceding node (4),  
| v so we can update its next reference to point to node 6 instead of 5.

head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6      How do we normally find a node in an SLL? By traversal.

What is the tightest worst-case time complexity of removing an element from an SLL with  $n$  elements?

Scenario 1: We are given a reference to the node to be removed.

Scenario 2: We are given the element to be removed.

Example: removal of node 5 from the list below, if we already know where to remove.

head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6

| \_\_\_\_\_ To remove node 5 we need a reference to its preceding node (4),  
| v so we can update its next reference to point to node 6 instead of 5.

head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6 How do we normally find a node in an SLL? By traversal.

**Scenario 1:** remove node pointed to by toRemove reference.

toRemove  
v Doesn't help, because we cannot move backwards to get pointer  
to  
head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6 preceding node. How do we find the preceding node? Traversal.

What is the tightest worst-case time complexity of removing an element from an SLL with  $n$  elements?

Scenario 1: We are given a reference to the node to be removed.

Scenario 2: We are given the element to be removed.

Example: removal of node 5 from the list below, if we already know where to remove.

head -> 1 -> 2 -> 3 -> 4 -> 5 -> 6

| \_\_\_\_\_ To remove node 5 we need a reference to its preceding node (4),  
| v so we can update its next reference to point to node 6 instead of 5.

head -> 1 -> 2 -> 3 -> 4 ~~-> 5 ->~~ 6 How do we normally find a node in an SLL? By traversal.

**Scenario 1:** remove node pointed to by toRemove reference.

toRemove  
v  
head -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 Doesn't help, because we cannot move backwards to get pointer to preceding node. How do we find the preceding node? Traversal.

**Scenario 2:** remove node with element 5.

head -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 Don't know which node has element 5, so we need to find it along preceding node (so that we can remove). How? Traversal.  
with

What is the tightest worst-case time complexity of removing an element from an SLL with  $n$  elements?

Scenario 1: We are given a reference to the node to be removed.

Scenario 2: We are given the element to be removed.

Example: removal of node 5 from the list below, if we already know where to remove.

head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6

| \_\_\_\_\_ To remove node 5 we need a reference to its preceding node (4),  
| v so we can update its next reference to point to node 6 instead of 5.

head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6 How do we normally find a node in an SLL? By traversal.

**Scenario 1:** remove node pointed to by toRemove reference.

toRemove  
v Doesn't help, because we cannot move backwards to get pointer to  
head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6 preceding node. How do we find the preceding node? Traversal.

**Scenario 2:** remove node with element 5.

head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6 Don't know which node has element 5, so we need to find it along with  
preceding node (so that we can remove). How? Traversal.

**Both scenarios:**

**Traverse list to find the node in  $O(n)$  time, then remove in  $O(1)$  time. Combined, these steps take  $O(n)$  time.**

4

How many changes to next and prev references are needed to perform an insertion into a doubly-linked list?

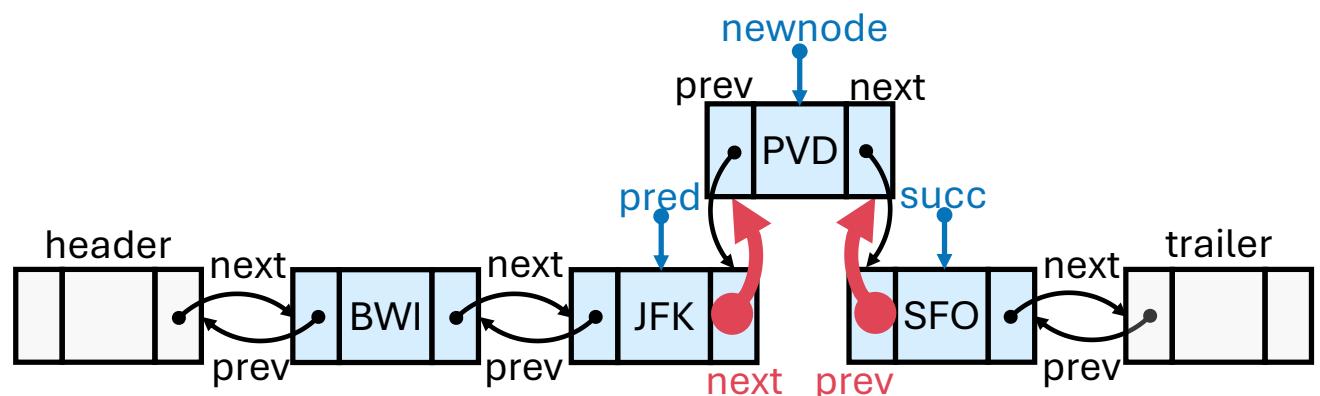
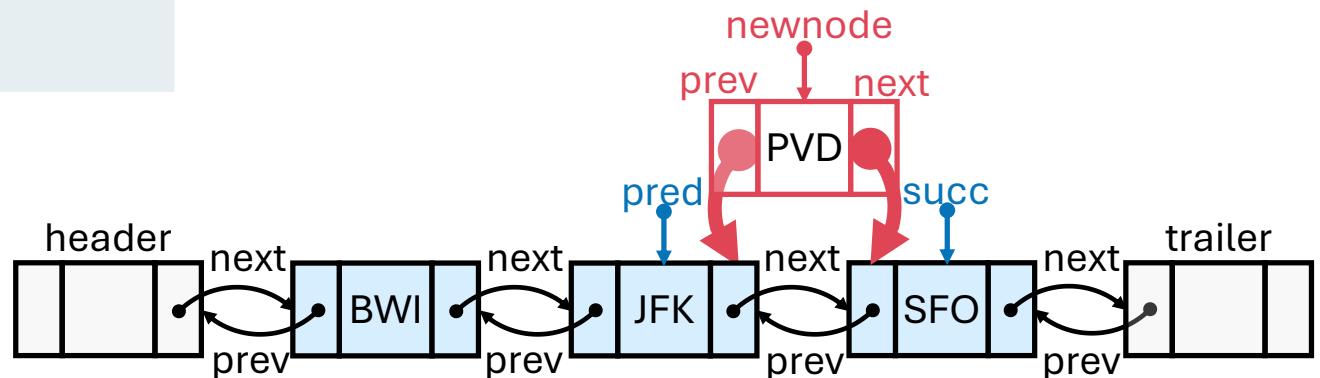
(1 Point)

- 1 prev, 1 next
- 2 prev, 2 next
- 3 prev, 3 next
- 4 prev, 4 next

How many changes to next and prev references are needed to perform an insertion into a doubly-linked list?

(1 Point)

- 1 prev, 1 next
- 2 prev, 2 next
- 3 prev, 3 next
- 4 prev, 4 next



- Update prev and next of new node. Total: 1 prev, 1 next.
- Update next of predecessor node. Total: 1 prev, 2 next.
- Update prev of successor node. **Total: 2 prev, 2 next.**

5

If a list m uses header and trailer nodes, which of the following denotes list m when it contains one element?

(1 Point)

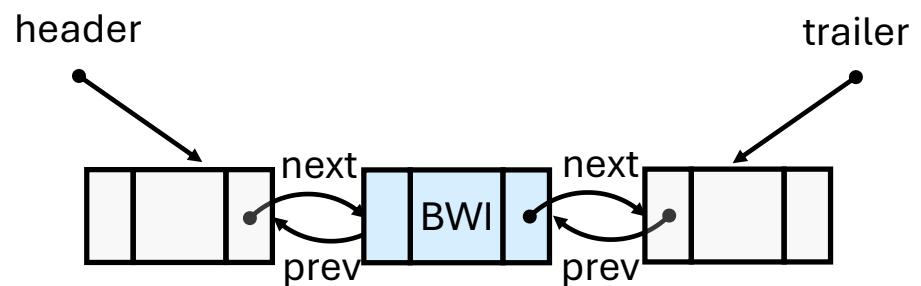
- m.header.next = null
- m.header.next = trailer
- m.header.next != null and m.header.next.next != null
- m.header.next != null and m.header.next.next = null

5

If a list m uses header and trailer nodes, which of the following denotes list m when it contains one element?

(1 Point)

- m.header.next = null
- m.header.next = trailer
- m.header.next != null and m.header.next.next != null
- m.header.next != null and m.header.next.next = null



Valid expressions for m.header.next and m.header.next.next:

**m.header.next != null**  
m.header.next != trailer  
m.header.next.next = trailer  
**m.header.next.next != null**

## 6

Final exam 18-19 - MCQ9  
(1 Point)

(1 point) Consider an algorithm to move the second and the last but one elements of a sequence  $S$  with  $n$  elements to the middle of that sequence. Example: if  $S$  has elements  $(1, 2, 3, 4, 5, 6, 7, 8)$ , the algorithm should change  $S$  into  $(1, 3, 4, 2, 7, 5, 6, 8)$ . What is the complexity of the most time-efficient algorithm for this operation when  $S$  is implemented by an array or a singly-linked list?

- A. array:  $\mathcal{O}(1)$     singly-linked list:  $\mathcal{O}(1)$
- B. array:  $\mathcal{O}(1)$     singly-linked list:  $\mathcal{O}(n)$
- C. array:  $\mathcal{O}(n)$     singly-linked list:  $\mathcal{O}(1)$
- D. array:  $\mathcal{O}(n)$     singly-linked list:  $\mathcal{O}(n)$

## Final exam 18-19 - MCQ9

(1 Point)

(1 point) Consider an algorithm to move the second and the last but one elements of a sequence  $S$  with  $n$  elements to the middle of that sequence. Example: if  $S$  has elements  $(1, 2, 3, 4, 5, 6, 7, 8)$ , the algorithm should change  $S$  into  $(1, 3, 4, 2, 7, 5, 6, 8)$ . What is the complexity of the most time-efficient algorithm for this operation when  $S$  is implemented by an array or a singly-linked list?

- A. array:  $\mathcal{O}(1)$  singly-linked list:  $\mathcal{O}(1)$
- B. array:  $\mathcal{O}(1)$  singly-linked list:  $\mathcal{O}(n)$
- C. array:  $\mathcal{O}(n)$  singly-linked list:  $\mathcal{O}(1)$
- D. array:  $\mathcal{O}(n)$  singly-linked list:  $\mathcal{O}(n)$

**Array**

$\{1, \textcolor{red}{2}, 3, 4, 5, 6, 7, 8, \textcolor{red}{9}, 10\}$  Access and store 2<sup>nd</sup> and 2<sup>nd</sup> last elements for later,  $\mathcal{O}(1)$  time.

$\{1, \textcolor{red}{2}, 3, 4, 5, 6, 7, 8, \textcolor{red}{9}, 10\}$  Shift elements left/right of middle resp. to left/right by 1 position.  
 <- <- <-|-> -> Moving is  $\mathcal{O}(1)$  per element, we move  $n - 4$  elements, so  $\mathcal{O}(n)$ .

$\{1, 3, 4, 5, 5, 6, 6, 7, 8, 10\}$  Status after shifting. Note the repetition of middle elements.

$\{1, 3, 4, 5, \textcolor{red}{2}, \textcolor{red}{9}, 6, 7, 8, 10\}$  Replace middle elements by the 2<sup>nd</sup> and 2<sup>nd</sup> last elements stored earlier.

Assignment of 2 elements takes  $\mathcal{O}(1)$  time.  
**Complete procedure takes  $\mathcal{O}(n)$  time.**

Move second and second last elements to middle.

## Final exam 18-19 - MCQ9

(1 Point)

(1 point) Consider an algorithm to move the second and the last but one elements of a sequence  $S$  with  $n$  elements to the middle of that sequence. Example: if  $S$  has elements  $(1, 2, 3, 4, 5, 6, 7, 8)$ , the algorithm should change  $S$  into  $(1, 3, 4, 2, 7, 5, 6, 8)$ . What is the complexity of the most time-efficient algorithm for this operation when  $S$  is implemented by an array or a singly-linked list?

- A. array:  $\mathcal{O}(1)$  singly-linked list:  $\mathcal{O}(1)$
- B. array:  $\mathcal{O}(1)$  singly-linked list:  $\mathcal{O}(n)$
- C. array:  $\mathcal{O}(n)$  singly-linked list:  $\mathcal{O}(1)$
- D. array:  $\mathcal{O}(n)$  singly-linked list:  $\mathcal{O}(n)$

**SLL**

head  $\rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10$  (second)

$\rightarrow$

head  $\rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10$  (middle)

$\rightarrow \rightarrow \rightarrow$

head  $\rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10$  (2<sup>nd</sup> last)

$\rightarrow \rightarrow \rightarrow$

**s = 2      toInsert    l = 9**

**v**

head  $\rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 9 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 10$  (insert)

Insert 2<sup>nd</sup> and 2<sup>nd</sup> last in the middle.

Move second and second last elements to middle.

Using a single traversal:

Find 1<sup>st</sup> node (from: head), remove 2<sup>nd</sup> node.

Find middle (from: 1<sup>st</sup> node), store middle.

Find 3<sup>rd</sup> last (from: middle), remove 2<sup>nd</sup> last.

All finds together take  $\mathcal{O}(n)$  time, since every find starts where the previous one stopped.

Remove, store, and insert take  $\mathcal{O}(1)$  time, since we have the positions. **Everything combined takes  $\mathcal{O}(n)$  time.**

## Final exam 19-20 - MCQ 10

(1 Point)

(1 point) Consider the `clone` method below for cloning objects of class `SinglyLinkedList`.

```
1 public SinglyLinkedList<E> clone() throws CloneNotSupportedException {
2     SinglyLinkedList<E> other = (SinglyLinkedList<E>) super.clone();
3     if (size > 0) {
4         other.head = new Node<E>(head.getElement(), null);
5         Node<E> walk = head.getNext();
6         Node<E> otherTail = other.head;
7         while (walk != null) {
8             Node<E> newest = new Node<E>(walk.getElement(), null);
9             otherTail.setNext(newest);
10            otherTail = newest;
11            walk = walk.getNext();
12        }
13    }
14    return other;
15 }
```

Which of the following statements is **true** about method `clone`?

- A. If a **node** of the clone list is replaced, the corresponding node in the original list will also change.
- B. If an **element** of the clone list is replaced, the corresponding element in the original list will also change.
- C. If an **instance field** of an element in the clone list is changed, then the field of the corresponding element in the original list will also change.
- D. If the **head** of the clone list is changed to point to the tail of the clone list, then the head of the original list will also point to the tail of the original list.

Which **one/two** line(s) of code contain(s) good evidence to help us answer this question?

Final exam 19-20 - MCQ 10  
(1 Point)

(1 point) Consider the `clone` method below for cloning objects of class `SinglyLinkedList`.

```

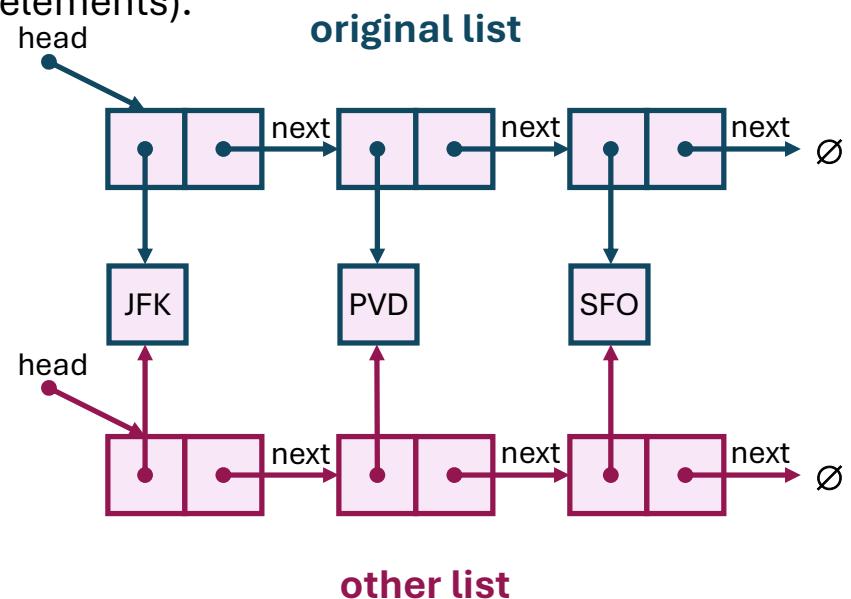
1 public SinglyLinkedList<E> clone() throws CloneNotSupportedException {
2     SinglyLinkedList<E> other = (SinglyLinkedList<E>) super.clone();
3     if (size > 0) {
4         other.head = new Node<E>(head.getElement(), null); ←
5         Node<E> walk = head.getNext();
6         Node<E> otherTail = other.head;
7         while (walk != null) {
8             Node<E> newest = new Node<E>(walk.getElement(), null); ←
9             otherTail.setNext(newest);
10            otherTail = newest;
11            walk = walk.getNext();
12        }
13    }
14    return other;
15 }
```

Which of the following statements is **true** about method `clone`?

- A. If a **node** of the clone list is replaced, the corresponding node in the original list will also change.
- B. If an **element** of the clone list is replaced, the corresponding element in the original list will also change.
- C. If an **instance field** of an element in the clone list is changed, then the field of the corresponding element in the original list will also change.
- D. If the **head** of the clone list is changed to point to the tail of the clone list, then the head of the original list will also point to the tail of the original list.

Nodes in new list **other** are newly created nodes.

But elements of new nodes in list **other** are the same as the ones in the original list (new nodes just contain references to the original elements).

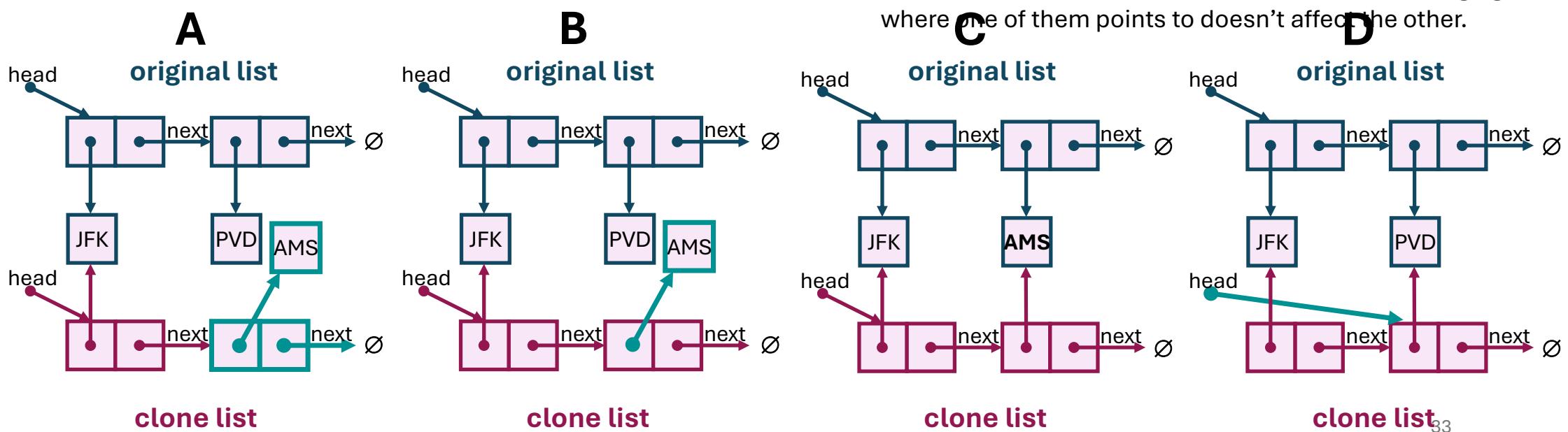


Final exam 19-20 - MCQ 10  
(1 Point)

Which of the following statements is **true** about method `clone`?

- A. If a **node** of the clone list is replaced, the corresponding node in the original list will also change.
- B. If an **element** of the clone list is replaced, the corresponding element in the original list will also change.
- C. If an **instance field** of an element in the clone list is changed, then the field of the corresponding element in the original list will also change.
- D. If the **head** of the clone list is changed to point to the tail of the clone list, then the head of the original list will also point to the tail of the original list.

- A. False. Node of clone replaced -> **no change** to original list since each list has its own nodes and references between them.
- B. False. Element of clone replaced -> no change to original, since the nodes of each list contain their own references to the elements (and we only change the reference of the clone).
- C. True.** Instance field of clone's element changes -> original also changes (because nodes of both lists point to same elements).
- D. False. Each list has its own head reference, so changing where one of them points to doesn't affect the other.



### Resit exam 17-18 - Q26 (adapted from)

*What does method "operateOnMatrix" do? What is its tightest worst-case time complexity in big-Oh notation? Would the method generate the same output if it was performed only on the upper triangle of the matrix? Describe a faster solution to perform the same operation only on the upper triangle of the matrix, assuming that the elements of each row are sorted.*

Method `operateOnMatrix` below performs an operation on a symmetric  $n$  by  $n$  matrix (row x column).

```

public class SymmetricMatrix {
    private int[][] matrix;
    /* ... */
    public static int[] operateOnMatrix(int x) {
        for (int i = 0; i < matrix.length; i++) {
            int j = operateOnArray(matrix[i], i, x);
            if (j > -1)
                return new int[]{i,j};
        }
        return null;
    }

    private static int operateOnArray(int[] row, int startIdx, int x) {
        for (int j = startIdx; j < row.length; j++)
            if (row[j] == x)
                return j;
        return -1;
    }
}

```

Entries in a symmetric matrix are symmetric with respect to the main diagonal. Example:  $M = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{pmatrix}$

## Resit exam 17-18 - Q26 (adapted from)

What does method "operateOnMatrix" do? What is its tightest worst-case time complexity in big-Oh notation? Would the method generate the same output if it was performed only on the upper triangle of the matrix? Describe a faster solution to perform the same operation only on the upper triangle of the matrix, assuming that the elements of each row are sorted.

Method operateOnMatrix below performs an operation on a symmetric  $n$  by  $n$  matrix (row x column).

```

public class SymmetricMatrix {
    private int[][] matrix;
    /* ... */
    public static int[] operateOnMatrix(int x) {
        for (int i = 0; i < matrix.length; i++) {
            int j = operateOnArray(matrix[i], i, x);
            if (j > -1)
                return new int[]{i,j};
        }
        return null;
    }

    private static int operateOnArray(int[] row, int startIdx, int x) {
        for (int j = startIdx; j < row.length; j++)
            if (row[j] == x)
                return j;
        return -1;
    }
}

```

Entries in a symmetric matrix are symmetric with respect to the main diagonal. Example:  $M = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{pmatrix}$

### What does it do?

Searches for element  $x$  in the matrix. Returns coordinate indices (row, column) if  $x$  is found, or  $-1$  otherwise.

### Tightest worst-case time complexity (big-Oh)?

Time is  $O(n^2)$ , since algorithm performs 2 nested for loops that iterate over rows and columns. Gauss' sum.

Same output if operation is performed on upper triangle?

Yes, it is already operating only on the upper triangle, because index  $j$  always starts from `startIdx`, which is set to  $i$ . Operating on upper triangle has the same big-Oh worst-case time complexity of operating on entire matrix.

Faster solution to perform operation on upper triangle, assuming the elements of each row are sorted.  
 $O(n \log n)$  using binary search on each row.

# Doubly linked list

Spot the mistakes:

```
public DLLList clone() {  
    DLLList result = new DLLList();  
    result.head = head;  
    Node current = head.next;  
    Node previous = current.previous;  
    Node next = current.next;  
    while (current != null) {  
        Node newNode = new Node(current.element, previous, next);  
        current = next;  
        previous = current;  
        next = next.next;  
    }  
    result.tail = previous;  
    return result;  
}
```

**Focus**

**30 sec**

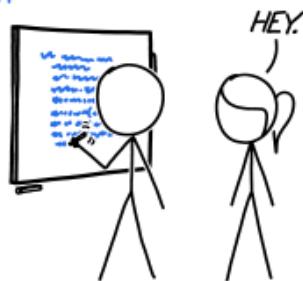
- Breath in and out 3 times

# Cooling down

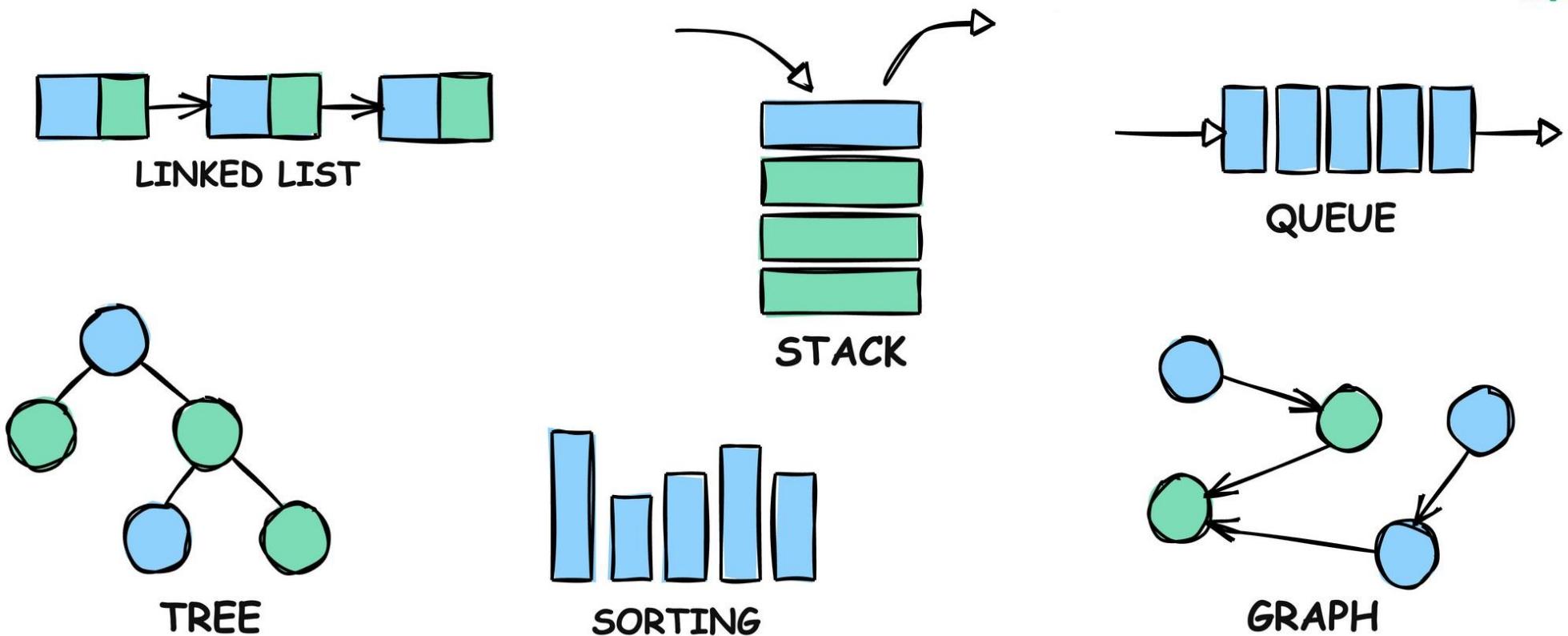
10:25-10:30

- Discuss how you are doing with the course
  - What did you learn in this lecture?
  - How are you going to prepare for the next lecture?
  - How are you doing in the course in general?
- Discuss how you are doing with life?
  - How was your weekend?
  - What do you feel right now?
  - What are you looking forward to?
- Forgive me for letting the lecture run late

```
define traverseLinkedList(headPointer):
    myID = "XXXXXXXXXXXX"
    authToken = "XXXXXXXXXXXXXX"
    museumAddress = "MUSEUM@GMAIL.COM"
    client = MailRestClient(myID, authToken)
    client.messages.send(to=museumAddress,
    subj="Item donation?", body="Thought you
    might be interested: "+str(headPointer))
    return
```



CODING INTERVIEW TIP: INTERVIEWERS GET  
REALY MAD WHEN YOU TRY TO DONATE THEIR  
LINKED LISTS TO A TECHNOLOGY MUSEUM.



# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Warming up

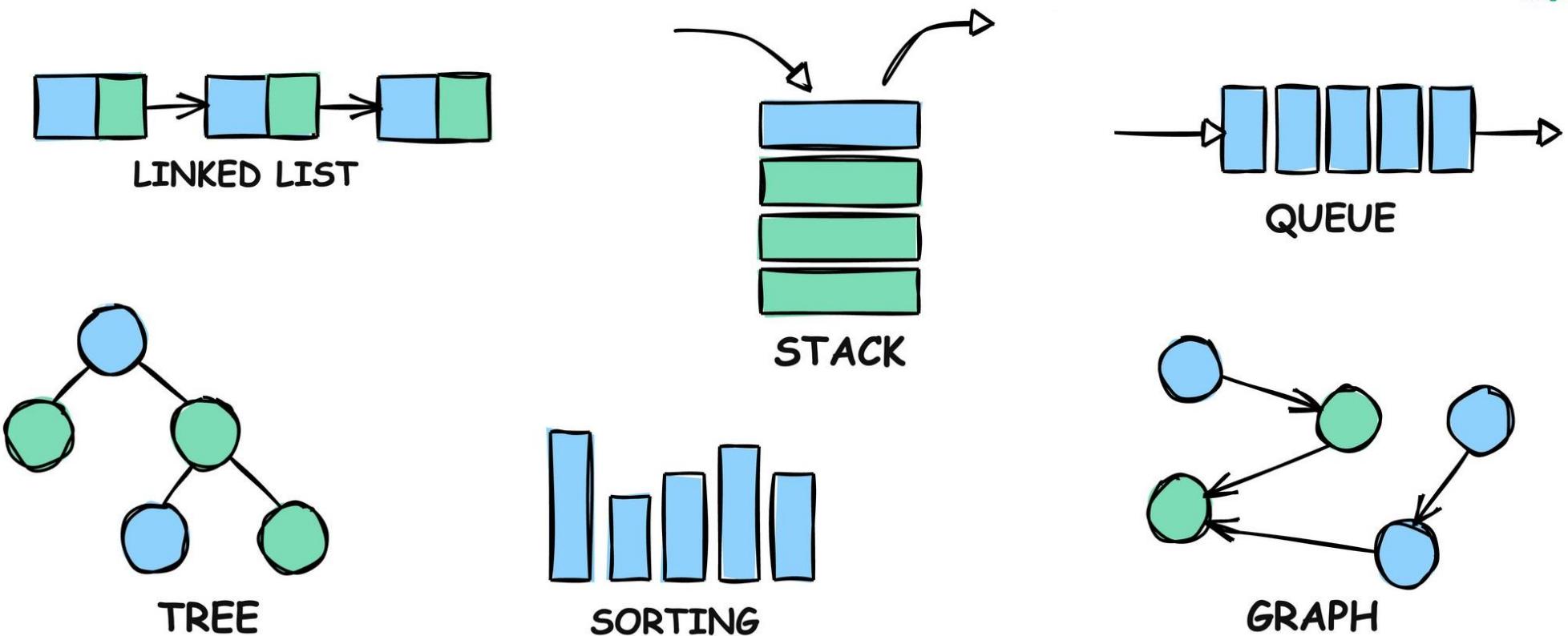
8:45-8:50

- Discuss how you are doing with the course
  - What did you learn in the preparation of this lecture?
  - What do you expect from this lecture?
  - What do you want to have learned after this lecture?
- Discuss how you are doing with life?
  - How was your weekend?
  - What do you feel right now?
  - What are you looking forward to?
- Forgive yourself for coming slightly late
  - (and have the intention to come on time next lecture)

**Focus**

**30 sec**

- Breath in and out 3 times



# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Stacks & Queues

Consider implementations of a stack and a queue data structures using a singly-linked list. Where do you need to add and remove elements to get a correctly working data structure with the optimal time complexities for both methods?

- A. For a stack: add to head
- B. For a stack: add to tail
- C. For a stack: remove from head
- D. For a stack: remove from tail
- E. For a queue: add to head
- F. For a queue: add to tail
- G. For a queue: remove from head
- H. For a queue: remove from tail

# Stacks

How could you implement a stack with a data structure that uses an array to store its data?

# Queue

How could you implement a queue with a data structure that uses an array to store its data?

# Queue from stacks

How can you implement a queue with 2 stacks as the only internal data structures?

What would be the time complexity of the operations of this queue?

# Dynamic arrays

Consider a dynamic array which starts with a capacity of 1. The first time the array reaches its capacity, its current capacity is increased with 3 positions (note, not to 3, but with 3). The second time the array is full, its capacity is increased with 5 positions. The third time, the capacity is increased with 7, and so on. Each time the array becomes full, the array capacity is increased with additional positions, namely as many as the next odd number. What is the amortized time complexity of adding an element to this dynamic array?

- A.  $O(1)$
- B.  $O(\log(n))$
- C.  $O(\sqrt{n})$
- D.  $O(n)$

1

## Midterm exam 18-19 - MCQ 9 (1 Point)

(1 point) Consider a time-efficient algorithm to remove the middle element of a sequence  $S$  with  $n$  elements. Assume that the middle element is the element at position  $n/2$ , where  $/$  denotes the integer division. Example: if  $S = \{1, 2, 3, 4, 5\}$ , the algorithm should change  $S$  into  $\{1, 2, 4, 5\}$ . What is the tightest worst-case **time** complexity of removing all elements of  $S$  by calling such algorithm repeatedly until  $S$  is empty, if  $S$  is implemented using an array or a singly-linked list?

- A. **array:**  $\mathcal{O}(n)$     **singly-linked list:**  $\mathcal{O}(n)$
- B. **array:**  $\mathcal{O}(n)$     **singly-linked list:**  $\mathcal{O}(n^2)$
- C. **array:**  $\mathcal{O}(n^2)$     **singly-linked list:**  $\mathcal{O}(n)$
- D. **array:**  $\mathcal{O}(n^2)$     **singly-linked list:**  $\mathcal{O}(n^2)$

1

Midterm exam 18-19 - MCQ 9  
(1 Point)

(1 point) Consider a time-efficient algorithm to remove the middle element of a sequence  $S$  with  $n$  elements. Assume that the middle element is the element at position  $n/2$ , where  $/$  denotes the integer division. Example: if  $S = \{1, 2, 3, 4, 5\}$ , the algorithm should change  $S$  into  $\{1, 2, 4, 5\}$ . What is the tightest worst-case **time** complexity of removing all elements of  $S$  by calling such algorithm repeatedly until  $S$  is empty, if  $S$  is implemented using an array or a singly-linked list?

**Idea:** call middle element removal repeatedly.

First look into removing middle element once.

**Idea:** call middle element removal repeatedly.

**What is the time complexity of removing a middle element once?.**

**Array:** remove element at index  $n/2$  :

- **Find:** calculate index of first element to be shifted Done in  $O(1)$  time.
- **Remove:** shift elements in indices  $n/2 + 1$  to  $n$  backwards by one position. Done in  $O(n)$  time.

Remove by shifting

{1, 2, 3, 4, 5, 6}

<- <- <-      Shift  $n - n/2 - 1$  elements backwards      by one position. This is approx.  $n/2$ .

**Idea:** call middle element removal repeatedly.

## What is the time complexity of removing a middle element once?.

**Array:** remove element at index  $n/2$ :

- **Find:** calculate index of first element to be shifted  
Done in  $O(1)$  time.
- **Remove:** shift elements in indices  $n/2 + 1$  to  $n$  backwards by one position. Done in  $O(n)$  time.

Remove by shifting

{1, 2, 3, 4, 5, 6}

<- <- <-      Shift  $n - n/2 - 1$  elements  
backwards      by one position. This is approx.  $n/2$ .

**SLL:** remove element at position  $n/2$ :

- **Find:** traverse list while counting until we find node at position  $n/2 - 1$ . Done in  $O(n)$  time.
- **Remove:** update next reference of node at position  $n/2 - 1$  to point to node at  $n/2 + 1$ .  $O(1)$  time.

Find by traversal

head -> 1 -> 2 -> 3 -> 4 -> 5 -> 6

-> ->

Follow  $n/2$  references

(head +  $n/2 - 1$  next  
references. Approx.  $n/2$ .)

**Idea:** call middle element removal repeatedly.

**What is the time complexity of removing a middle element once from a sequence of size  $n$ ?  $O(n)$  for both array and SLL**

**Array:** remove element at index  $n/2$ :

- **Find:** calculate index of first element to be shifted  
Done in  $O(1)$  time.
- **Remove:** shift elements in indices  $n/2 + 1$  to  $n$  backwards by one position. Done in  $O(n)$  time.

Remove by shifting

{1, 2, 3, 4, 5, 6}

<- <- <-      Shift  $n - n/2 - 1$  elements  
backwards      by one position. This is approx.  $n/2$ .

**SLL:** remove element at position  $n/2$ :

- **Find:** traverse list while counting until we find node at position  $n/2 - 1$ . Done in  $O(n)$  time.
- **Remove:** update next reference of node at position  $n/2 - 1$  to point to node at  $n/2 + 1$ .  $O(1)$  time.

Find by traversal

head -> 1 -> 2 -> 3 -> 4 -> 5 -> 6

-> ->

Follow  $n/2$  references

(head +  $n/2 - 1$  next  
references. Approx.  $n/2$ .)

Midterm exam 18-19 - MCQ 9  
(1 Point)

(1 point) Consider a time-efficient algorithm to remove the middle element of a sequence  $S$  with  $n$  elements. Assume that the middle element is the element at position  $n/2$ , where  $/$  denotes the integer division. Example: if  $S = \{1, 2, 3, 4, 5\}$ , the algorithm should change  $S$  into  $\{1, 2, 4, 5\}$ . What is the tightest worst-case time complexity of removing all elements of  $S$  by calling such algorithm repeatedly until  $S$  is empty, if  $S$  is implemented using an array or a singly-linked list?

**Array:** remove element at index  $k/2$ :

**Runtime of one middle element removal:  $O(k)$ .**

For multiple removals:

Remove by shifting

$\{1, 2, 3, 4, 5, 6\}$

$<- <- <-$

$\{1, 2, 4, 5, 6\}$

$<- <-$

$\{1, 2, 5, 6\}$

$<- <-$

...

Iteration 1: list size  $n$

Iteration 2: list size  $n - 1$

Iteration 3: list size  $n - 2$

...

Iteration  $n$ : list size 1

**Runtime of all removals together:**

Find by traversal

head  $\rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$

$\rightarrow \rightarrow$

head  $\rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6$

$\rightarrow \rightarrow$

head  $\rightarrow 1 \rightarrow 2 \rightarrow 5 \rightarrow 6$

$\rightarrow$

...

$$\sum_{k=1}^n k/2 = \frac{n^2 + n}{8}$$

(ignoring constants)

**Runtime**  
 **$O(n^2)$**

What is the tightest worst-case time complexity of removing an element from an SLL?

**Scenario 1:** We are given a reference to the node to be removed.

**Scenario 2:** We are given the element to be removed.

What is the tightest worst-case time complexity of removing an element from an SLL with  $n$  elements?

Scenario 1: We are given a reference to the node to be removed.

Scenario 2: We are given the element to be removed.

Example: removal of node 5 from the list below, if we already know where to remove.

head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6



To remove node 5 we need a reference to its preceding node (4),  
so we can update its next reference to point to node 6 instead of 5.

head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6

How do we normally find a node in an SLL? By traversal.

What is the tightest worst-case time complexity of removing an element from an SLL with  $n$  elements?

Scenario 1: We are given a reference to the node to be removed.

Scenario 2: We are given the element to be removed.

Example: removal of node 5 from the list below, if we already know where to remove.

head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6

| \_\_\_\_\_ To remove node 5 we need a reference to its preceding node (4),  
| v so we can update its next reference to point to node 6 instead of 5.

head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6 How do we normally find a node in an SLL? By traversal.

**Scenario 1:** remove node pointed to by toRemove reference.

toRemove  
v Doesn't help, because we cannot move backwards to get pointer  
to  
head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6 preceding node. How do we find the preceding node? Traversal.

What is the tightest worst-case time complexity of removing an element from an SLL with  $n$  elements?

Scenario 1: We are given a reference to the node to be removed.

Scenario 2: We are given the element to be removed.

Example: removal of node 5 from the list below, if we already know where to remove.

head -> 1 -> 2 -> 3 -> 4 -> 5 -> 6

| \_\_\_\_\_ To remove node 5 we need a reference to its preceding node (4),  
| v so we can update its next reference to point to node 6 instead of 5.

head -> 1 -> 2 -> 3 -> 4 ~~-> 5 ->~~ 6 How do we normally find a node in an SLL? By traversal.

**Scenario 1:** remove node pointed to by toRemove reference.

toRemove  
v  
head -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 Doesn't help, because we cannot move backwards to get pointer to preceding node. How do we find the preceding node? Traversal.

**Scenario 2:** remove node with element 5.

head -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 Don't know which node has element 5, so we need to find it along preceding node (so that we can remove). How? Traversal.  
with

What is the tightest worst-case time complexity of removing an element from an SLL with  $n$  elements?

Scenario 1: We are given a reference to the node to be removed.

Scenario 2: We are given the element to be removed.

Example: removal of node 5 from the list below, if we already know where to remove.

head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6

| \_\_\_\_\_ To remove node 5 we need a reference to its preceding node (4),  
| v so we can update its next reference to point to node 6 instead of 5.

head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6 How do we normally find a node in an SLL? By traversal.

**Scenario 1:** remove node pointed to by toRemove reference.

toRemove  
v Doesn't help, because we cannot move backwards to get pointer to  
head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6 preceding node. How do we find the preceding node? Traversal.

**Scenario 2:** remove node with element 5.

head  $\rightarrow$  1  $\rightarrow$  2  $\rightarrow$  3  $\rightarrow$  4  $\rightarrow$  5  $\rightarrow$  6 Don't know which node has element 5, so we need to find it along with  
preceding node (so that we can remove). How? Traversal.

**Both scenarios:**

**Traverse list to find the node in  $O(n)$  time, then remove in  $O(1)$  time. Combined, these steps take  $O(n)$  time.**

4

How many changes to next and prev references are needed to perform an insertion into a doubly-linked list?

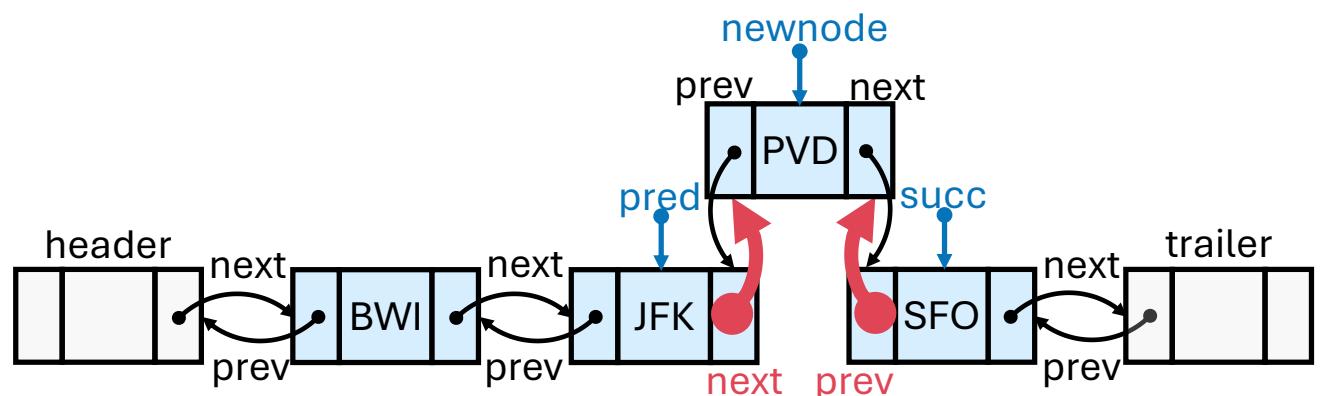
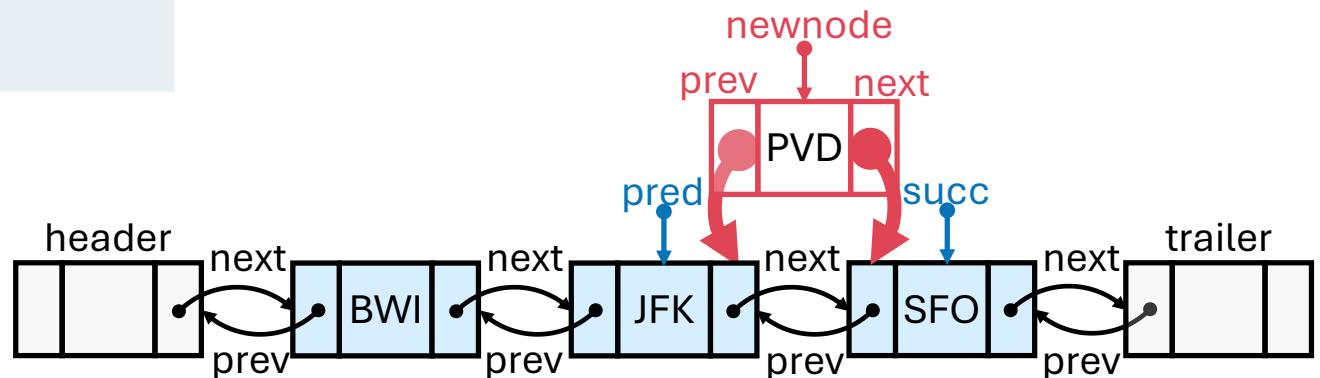
(1 Point)

- 1 prev, 1 next
- 2 prev, 2 next
- 3 prev, 3 next
- 4 prev, 4 next

How many changes to next and prev references are needed to perform an insertion into a doubly-linked list?

(1 Point)

- 1 prev, 1 next
- 2 prev, 2 next
- 3 prev, 3 next
- 4 prev, 4 next



- Update prev and next of new node. Total: 1 prev, 1 next.
- Update next of predecessor node. Total: 1 prev, 2 next.
- Update prev of successor node. **Total: 2 prev, 2 next.**

5

If a list m uses header and trailer nodes, which of the following denotes list m when it contains one element?

(1 Point)

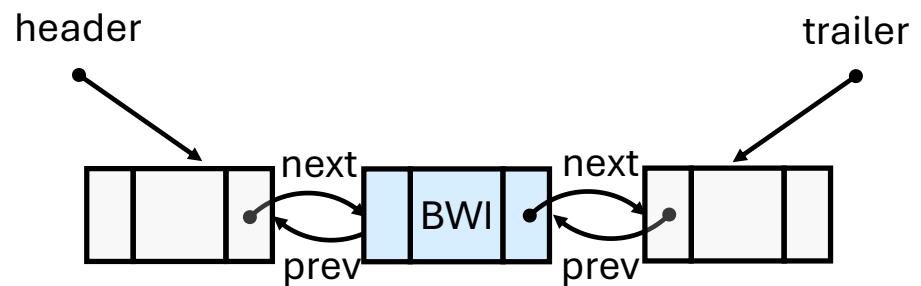
- m.header.next = null
- m.header.next = trailer
- m.header.next != null and m.header.next.next != null
- m.header.next != null and m.header.next.next = null

5

If a list m uses header and trailer nodes, which of the following denotes list m when it contains one element?

(1 Point)

- m.header.next = null
- m.header.next = trailer
- m.header.next != null and m.header.next.next != null
- m.header.next != null and m.header.next.next = null



Valid expressions for m.header.next and m.header.next.next:

**m.header.next != null**  
m.header.next != trailer  
m.header.next.next = trailer  
**m.header.next.next != null**

## 6

## Final exam 18-19 - MCQ9

(1 Point)

(1 point) Consider an algorithm to move the second and the last but one elements of a sequence  $S$  with  $n$  elements to the middle of that sequence. Example: if  $S$  has elements  $(1, 2, 3, 4, 5, 6, 7, 8)$ , the algorithm should change  $S$  into  $(1, 3, 4, 2, 7, 5, 6, 8)$ . What is the complexity of the most time-efficient algorithm for this operation when  $S$  is implemented by an array or a singly-linked list?

- A. array:  $\mathcal{O}(1)$     singly-linked list:  $\mathcal{O}(1)$
- B. array:  $\mathcal{O}(1)$     singly-linked list:  $\mathcal{O}(n)$
- C. array:  $\mathcal{O}(n)$     singly-linked list:  $\mathcal{O}(1)$
- D. array:  $\mathcal{O}(n)$     singly-linked list:  $\mathcal{O}(n)$

## Final exam 18-19 - MCQ9

(1 Point)

(1 point) Consider an algorithm to move the second and the last but one elements of a sequence  $S$  with  $n$  elements to the middle of that sequence. Example: if  $S$  has elements  $(1, 2, 3, 4, 5, 6, 7, 8)$ , the algorithm should change  $S$  into  $(1, 3, 4, 2, 7, 5, 6, 8)$ . What is the complexity of the most time-efficient algorithm for this operation when  $S$  is implemented by an array or a singly-linked list?

- A. array:  $\mathcal{O}(1)$  singly-linked list:  $\mathcal{O}(1)$
- B. array:  $\mathcal{O}(1)$  singly-linked list:  $\mathcal{O}(n)$
- C. array:  $\mathcal{O}(n)$  singly-linked list:  $\mathcal{O}(1)$
- D. array:  $\mathcal{O}(n)$  singly-linked list:  $\mathcal{O}(n)$

**Array**

$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$  Access and store 2<sup>nd</sup> and 2<sup>nd</sup> last elements for later,  $\mathcal{O}(1)$  time.

$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$  Shift elements left/right of middle resp. to left/right by 1 position.  
 <- <- <-|-> -> Moving is  $\mathcal{O}(1)$  per element, we move  $n - 4$  elements, so  $\mathcal{O}(n)$ .

$\{1, 3, 4, 5, 5, 6, 6, 7, 8, 10\}$  Status after shifting. Note the repetition of middle elements.

$\{1, 3, 4, 5, 2, 9, 6, 7, 8, 10\}$  Replace middle elements by the 2<sup>nd</sup> and 2<sup>nd</sup> last elements stored earlier.

Assignment of 2 elements takes  $\mathcal{O}(1)$  time.  
**Complete procedure takes  $\mathcal{O}(n)$  time.**

Move second and second last elements to middle.

## Final exam 18-19 - MCQ9

(1 Point)

(1 point) Consider an algorithm to move the second and the last but one elements of a sequence  $S$  with  $n$  elements to the middle of that sequence. Example: if  $S$  has elements  $(1, 2, 3, 4, 5, 6, 7, 8)$ , the algorithm should change  $S$  into  $(1, 3, 4, 2, 7, 5, 6, 8)$ . What is the complexity of the most time-efficient algorithm for this operation when  $S$  is implemented by an array or a singly-linked list?

- A. array:  $\mathcal{O}(1)$  singly-linked list:  $\mathcal{O}(1)$
- B. array:  $\mathcal{O}(1)$  singly-linked list:  $\mathcal{O}(n)$
- C. array:  $\mathcal{O}(n)$  singly-linked list:  $\mathcal{O}(1)$
- D. array:  $\mathcal{O}(n)$  singly-linked list:  $\mathcal{O}(n)$

**SLL**

head  $\rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10$  (second)

$\rightarrow$

head  $\rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10$  (middle)

$\rightarrow \rightarrow \rightarrow$

head  $\rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10$  (2<sup>nd</sup> last)

$\rightarrow \rightarrow \rightarrow$

**s = 2      toInsert    l = 9**

**v**

head  $\rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 9 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 10$  (insert)

Insert 2<sup>nd</sup> and 2<sup>nd</sup> last in the middle.

Move second and second last elements to middle.

Using a single traversal:

Find 1<sup>st</sup> node (from: head), remove 2<sup>nd</sup> node.

Find middle (from: 1<sup>st</sup> node), store middle.

Find 3<sup>rd</sup> last (from: middle), remove 2<sup>nd</sup> last.

All finds together take  $\mathcal{O}(n)$  time, since every find starts where the previous one stopped.

Remove, store, and insert take  $\mathcal{O}(1)$  time, since we have the positions. **Everything combined takes  $\mathcal{O}(n)$  time.**

Final exam 19-20 - MCQ 10  
(1 Point)

(1 point) Consider the `clone` method below for cloning objects of class `SinglyLinkedList`.

```
1 public SinglyLinkedList<E> clone() throws CloneNotSupportedException {
2     SinglyLinkedList<E> other = (SinglyLinkedList<E>) super.clone();
3     if (size > 0) {
4         other.head = new Node<E>(head.getElement(), null);
5         Node<E> walk = head.getNext();
6         Node<E> otherTail = other.head;
7         while (walk != null) {
8             Node<E> newest = new Node<E>(walk.getElement(), null);
9             otherTail.setNext(newest);
10            otherTail = newest;
11            walk = walk.getNext();
12        }
13    }
14    return other;
15 }
```

Which of the following statements is **true** about method `clone`?

- A. If a **node** of the clone list is replaced, the corresponding node in the original list will also change.
- B. If an **element** of the clone list is replaced, the corresponding element in the original list will also change.
- C. If an **instance field** of an element in the clone list is changed, then the field of the corresponding element in the original list will also change.
- D. If the **head** of the clone list is changed to point to the tail of the clone list, then the head of the original list will also point to the tail of the original list.

Which **one/two** line(s) of code contain(s) good evidence to help us answer this question?

Final exam 19-20 - MCQ 10  
(1 Point)

(1 point) Consider the `clone` method below for cloning objects of class `SinglyLinkedList`.

```

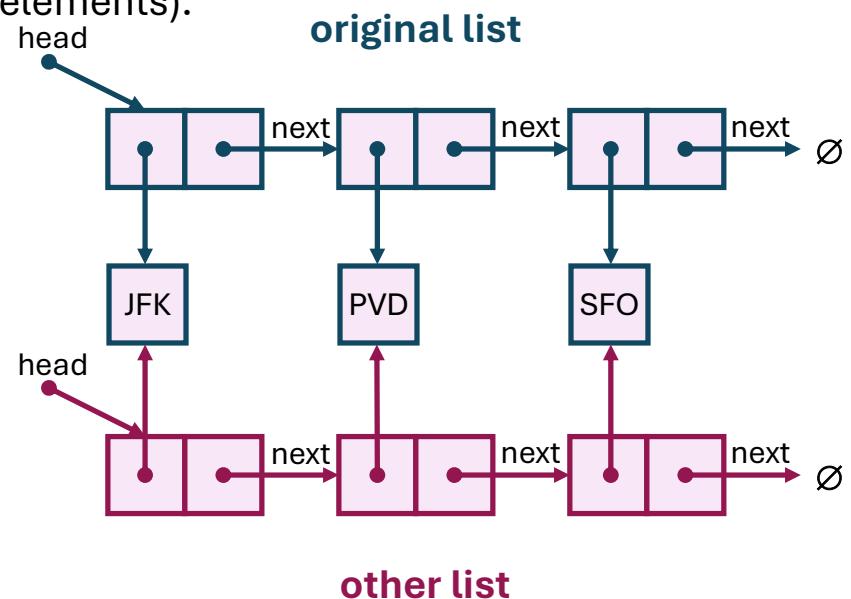
1 public SinglyLinkedList<E> clone() throws CloneNotSupportedException {
2     SinglyLinkedList<E> other = (SinglyLinkedList<E>) super.clone();
3     if (size > 0) {
4         other.head = new Node<E>(head.getElement(), null); ←
5         Node<E> walk = head.getNext();
6         Node<E> otherTail = other.head;
7         while (walk != null) {
8             Node<E> newest = new Node<E>(walk.getElement(), null); ←
9             otherTail.setNext(newest);
10            otherTail = newest;
11            walk = walk.getNext();
12        }
13    }
14    return other;
15 }
```

Which of the following statements is **true** about method `clone`?

- A. If a **node** of the clone list is replaced, the corresponding node in the original list will also change.
- B. If an **element** of the clone list is replaced, the corresponding element in the original list will also change.
- C. If an **instance field** of an element in the clone list is changed, then the field of the corresponding element in the original list will also change.
- D. If the **head** of the clone list is changed to point to the tail of the clone list, then the head of the original list will also point to the tail of the original list.

Nodes in new list **other** are newly created nodes.

But elements of new nodes in list **other** are the same as the ones in the original list (new nodes just contain references to the original elements).

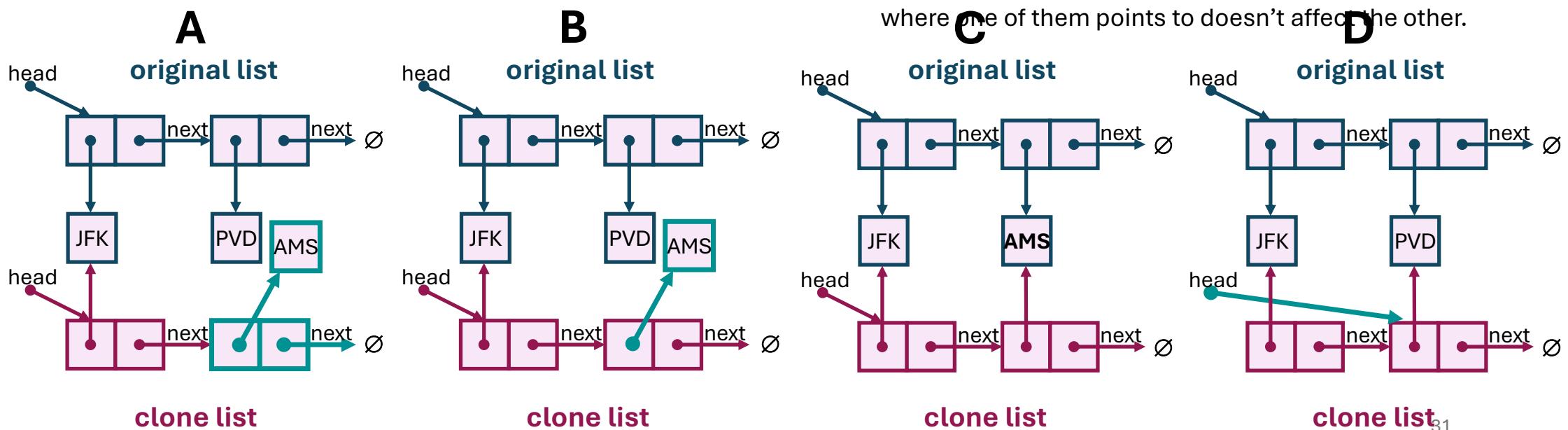


Final exam 19-20 - MCQ 10  
(1 Point)

Which of the following statements is **true** about method `clone`?

- A. If a **node** of the clone list is replaced, the corresponding node in the original list will also change.
- B. If an **element** of the clone list is replaced, the corresponding element in the original list will also change.
- C. If an **instance field** of an element in the clone list is changed, then the field of the corresponding element in the original list will also change.
- D. If the **head** of the clone list is changed to point to the tail of the clone list, then the head of the original list will also point to the tail of the original list.

- A. False. Node of clone replaced -> **no change** to original list since each list has its own nodes and references between them.
- B. False. Element of clone replaced -> no change to original, since the nodes of each list contain their own references to the elements (and we only change the reference of the clone).
- C. True.** Instance field of clone's element changes -> original also changes (because nodes of both lists point to same elements).
- D. False. Each list has its own head reference, so changing where one of them points to doesn't affect the other.



### Resit exam 17-18 - Q26 (adapted from)

*What does method "operateOnMatrix" do? What is its tightest worst-case time complexity in big-Oh notation? Would the method generate the same output if it was performed only on the upper triangle of the matrix? Describe a faster solution to perform the same operation only on the upper triangle of the matrix, assuming that the elements of each row are sorted.*

Method `operateOnMatrix` below performs an operation on a symmetric  $n$  by  $n$  matrix (row x column).

```

public class SymmetricMatrix {
    private int[][] matrix;
    /* ... */
    public static int[] operateOnMatrix(int x) {
        for (int i = 0; i < matrix.length; i++) {
            int j = operateOnArray(matrix[i], i, x);
            if (j > -1)
                return new int[]{i,j};
        }
        return null;
    }

    private static int operateOnArray(int[] row, int startIdx, int x) {
        for (int j = startIdx; j < row.length; j++)
            if (row[j] == x)
                return j;
        return -1;
    }
}

```

Entries in a symmetric matrix are symmetric with respect to the main diagonal. Example:  $M = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{pmatrix}$

### Resit exam 17-18 - Q26 (adapted from)

What does method "operateOnMatrix" do? What is its tightest worst-case time complexity in big-Oh notation? Would the method generate the same output if it was performed only on the upper triangle of the matrix? Describe a faster solution to perform the same operation only on the upper triangle of the matrix, assuming that the elements of each row are sorted.

Method operateOnMatrix below performs an operation on a symmetric  $n$  by  $n$  matrix (row x column).

```

public class SymmetricMatrix {
    private int[][] matrix;
    /* ... */
    public static int[] operateOnMatrix(int x) {
        for (int i = 0; i < matrix.length; i++) {
            int j = operateOnArray(matrix[i], i, x);
            if (j > -1)
                return new int[]{i,j};
        }
        return null;
    }

    private static int operateOnArray(int[] row, int startIdx, int x) {
        for (int j = startIdx; j < row.length; j++)
            if (row[j] == x)
                return j;
        return -1;
    }
}

```

Entries in a symmetric matrix are symmetric with respect to the main diagonal. Example:  $M = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{pmatrix}$

### What does it do?

Searches for element  $x$  in the matrix. Returns coordinate indices (row, column) if  $x$  is found, or  $-1$  otherwise.

### Tightest worst-case time complexity (big-Oh)?

Time is  $O(n^2)$ , since algorithm performs 2 nested for loops that iterate over rows and columns. Gauss' sum.

Same output if operation is performed on upper triangle?

Yes, it is already operating only on the upper triangle, because index  $j$  always starts from `startIdx`, which is set to  $i$ . Operating on upper triangle has the same big-Oh worst-case time complexity of operating on entire matrix.

Faster solution to perform operation on upper triangle, assuming the elements of each row are sorted.  
 $O(n \log n)$  using binary search on each row.

# Doubly linked list

Spot the mistakes:

```
public DLLList clone() {  
    DLLList result = new DLLList();  
    result.head = head;  
    Node current = head.next;  
    Node previous = current.previous;  
    Node next = current.next;  
    while (current != null) {  
        Node newNode = new Node(current.element, previous, next);  
        current = next;  
        previous = current;  
        next = next.next;  
    }  
    result.tail = previous;  
    return result;  
}
```

**Focus**

**30 sec**

- Breath in and out 3 times

# Cooling down

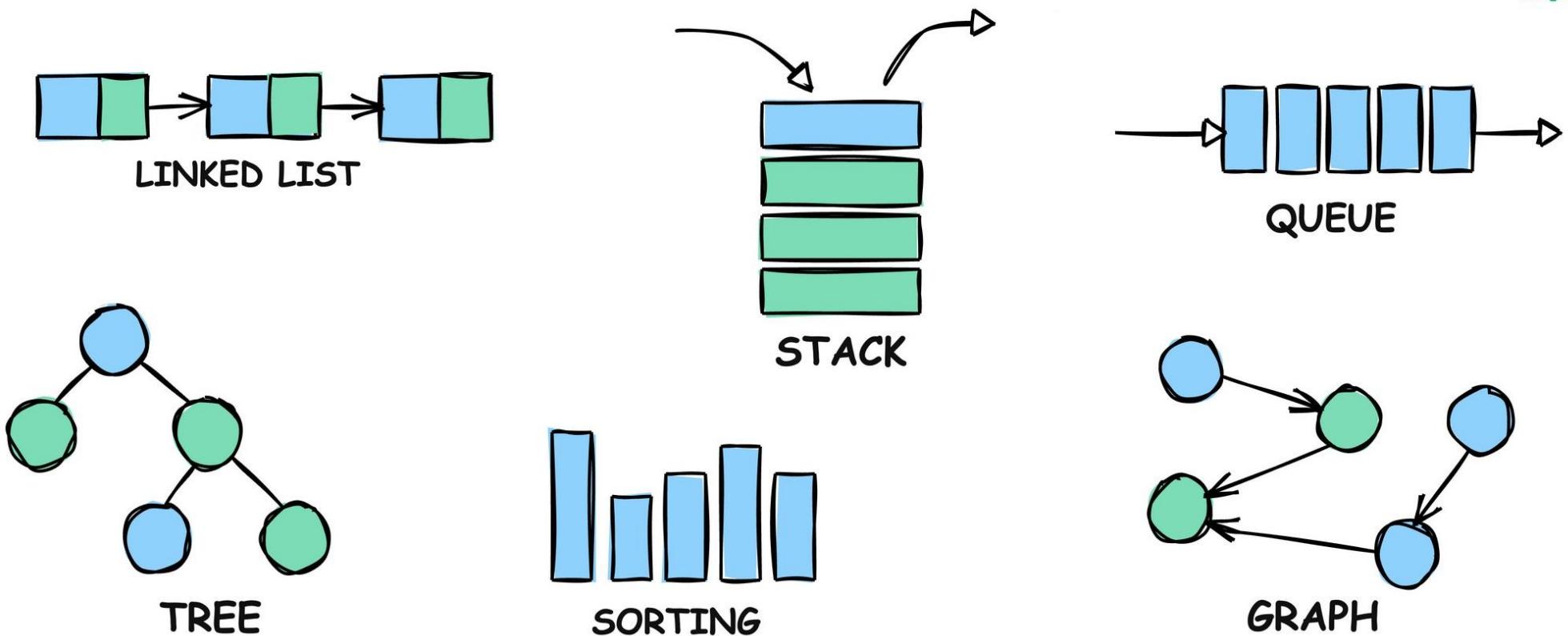
10:25-10:30

- Discuss how you are doing with the course
  - What did you learn in this lecture?
  - How are you going to prepare for the next lecture?
  - How are you doing in the course in general?
- Discuss how you are doing with life?
  - How was your weekend?
  - What do you feel right now?
  - What are you looking forward to?
- Forgive me for letting the lecture run late

## MY HOBBY: EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS

CHOTCHKIES RESTAURANT	
~~ APPETIZERS ~~	
MIXED FRUIT	2.15
FRENCH FRIES	2.75
SIDE SALAD	3.35
HOT WINGS	3.55
MOZZARELLA STICKS	4.20
SAMPLER PLATE	5.80
~~ SANDWICHES ~~	
BARBECUE	6.55





# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Warming up

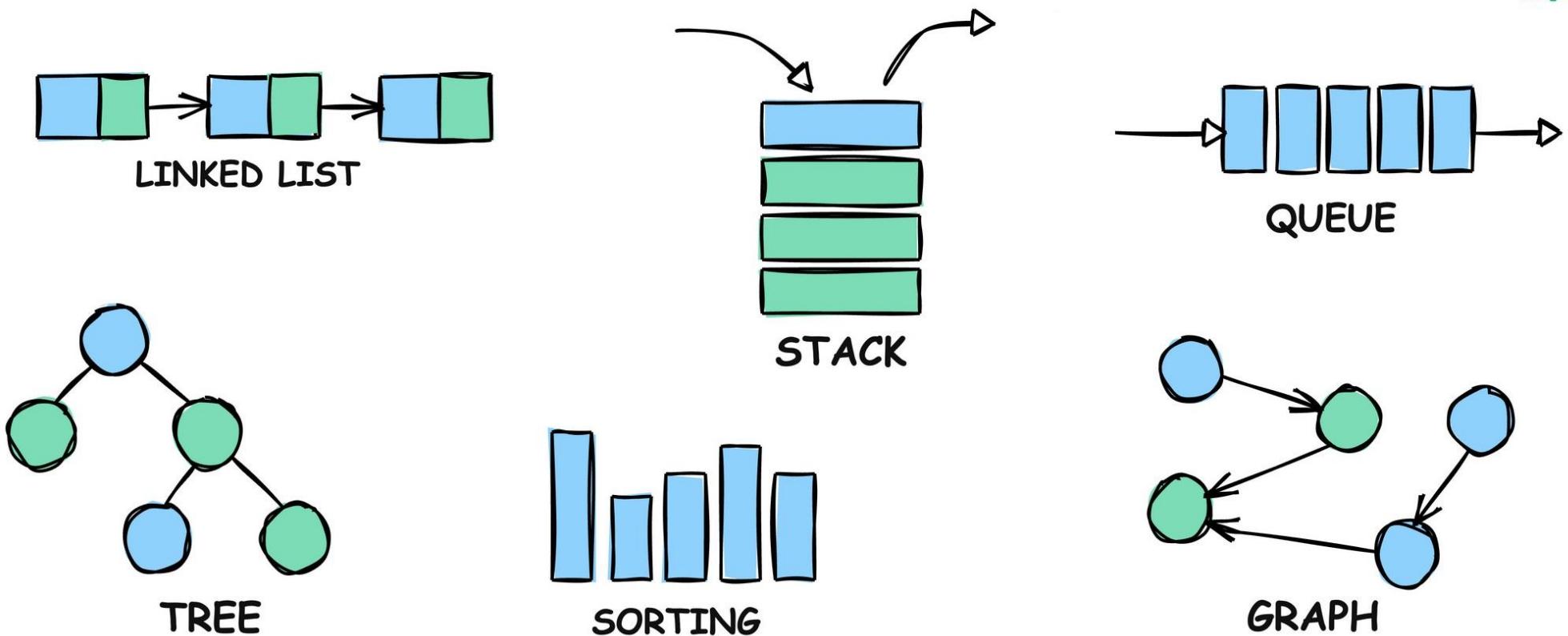
8:45-8:50

- Discuss how you are doing with the course
  - What did you learn in the preparation of this lecture?
  - What do you expect from this lecture?
  - What do you want to have learned after this lecture?
- Discuss how you are doing with life?
  - How was your weekend?
  - What do you feel right now?
  - What are you looking forward to?
- Forgive yourself for coming slightly late
  - (and have the intention to come on time next lecture)

**Focus**

**30 sec**

- Breath in and out 3 times



# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Positional lists

Which of the following statements is true about positional lists?

- A. A Position p of a linked positional list enables direct access to the previous and next positions using p.prev and p.next
- B. The methods that add nodes to a linked positional list return nodes of a list
- C. A Position is an abstraction used to encapsulate the actual nodes and prevent unauthorized modifications of the list
- D. Array-based positional lists are more space-efficient than standard array-based lists.

# Iterators

Which of the following methods is usually O(1)?

- A. The construction of a lazy iterator
- B. The next method of a lazy iterator
- C. The construction of a snapshot iterator
- D. The next method of a snapshot iterator

# Iterators

What do each of the following methods print?

- A. 1: 10, 10, 10
- B. 1: 10, 50, 100
- C. 1: 10, 100, 1000
- D. 1: 10, 50, 100, 500, 1000
- E. 2: 10, 10, 10
- F. 2: 10, 50, 100
- G. 2: 10, 100, 1000
- H. 2: 10, 50, 100, 500, 1000

(1 point) Consider the implementation of methods `printElements1` and `printElements2` below and assume that the list `elements` given as input contains the following elements: {10, 50, 100, 500, 1000}.

```
1 public static void printElements1(List<Integer> elements) {
2     int i = 0;
3     while(elements.iterator().hasNext() && i < 3) {
4         System.out.println(elements.iterator().next());
5         i++;
6     }
7 }
8
9 public static void printElements2(List<Integer> elements) {
10    int i = 0;
11    Iterator<Integer> iterator = elements.iterator();
12    while(iterator.hasNext() && i < 3) {
13        System.out.println(iterator.next());
14        i++;
15    }
16 }
```

# Tree definitions

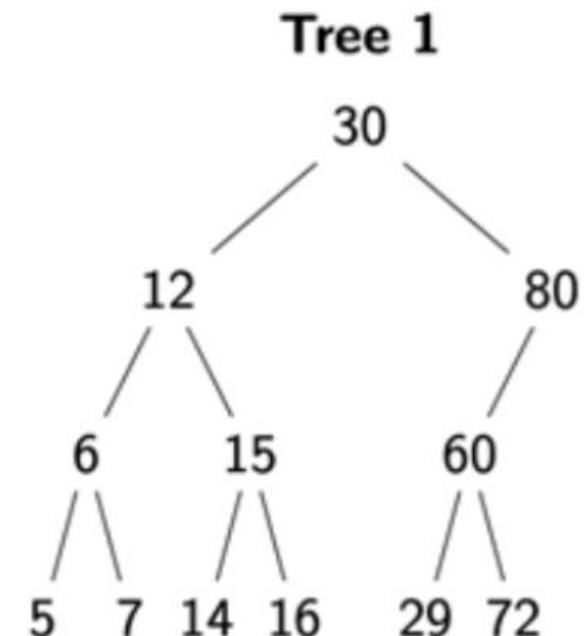
Given a node in a tree with depth 6 and height 6. Which depth and height could its parent node possibly have?

- A. Depth: 3
- B. Depth: 5
- C. Depth: 6
- D. Depth: 7
- E. Depth: 9
- F. Height: 3
- G. Height: 5
- H. Height: 6
- I. Height: 7
- J. Height: 9

# Tree traversals

If the output is [5, 7, 6, 14, 16, 15, 12, 29, 72, 60, 80, 30], what traversal is performed on the tree below?

- A. Breadth-first traversal
- B. In-order traversal
- C. Post-order traversal
- D. Pre-order traversal



# Tree traversals

There exists a tree with 3 nodes for which three of the four traversals (preorder, inorder, postorder, and breadth-first) result in the same order of nodes. For such a tree, what is the traversal that does not produce the same order of nodes than the other three?

- A. Breadth-first traversal
- B. In-order traversal
- C. Post-order traversal
- D. Pre-order traversal

# Tree traversals

Give pseudo-code for a depth-first traversal of a tree.

Give pseudo-code for a breadth-first traversal of a tree.

# Tree representations

In a binary tree represented using an array, how would you calculate the index of the children of a node with an index  $i$ ? And how would you calculate the index of the parent?

# Tree representations

When is it inefficient to use an array-based implementation of a tree?

- A. When the tree is very high
- B. When the tree has many nodes
- C. When nodes of the tree have many children
- D. When the root of the tree has a high depth

# Tree properties

In what trees is the height of the tree guaranteed to be  $O(\log(n))$ ?

- A. A proper binary tree (all nodes have either 0 or 2 children)
- B. A balanced binary tree (the height of 2 siblings differs by at most 1)
- C. A complete binary tree (all levels of the tree are full, except the lowest, and all nodes in the lowest level are as far left as possible)
- D. A perfect binary tree (all levels of the tree are full)

# Tree properties

In what trees would an array-representation of the tree have no empty elements in the array?

- A. A proper binary tree (all nodes have either 0 or 2 children)
- B. A balanced binary tree (the height of 2 siblings differs by at most 1)
- C. A complete binary tree (all levels of the tree are full, except the lowest, and all nodes in the lowest level are as far left as possible)
- D. A perfect binary tree (all levels of the tree are full)

# Tree properties

Given a tree of integers where for any leaf, the ancestors of that leaf are sorted (if an ancestor has a lower depth than another ancestor, it must have a higher value). How could you add an integer to this tree while maintaining this property?

What if the tree should be a complete tree in addition to the above property?

**Focus**

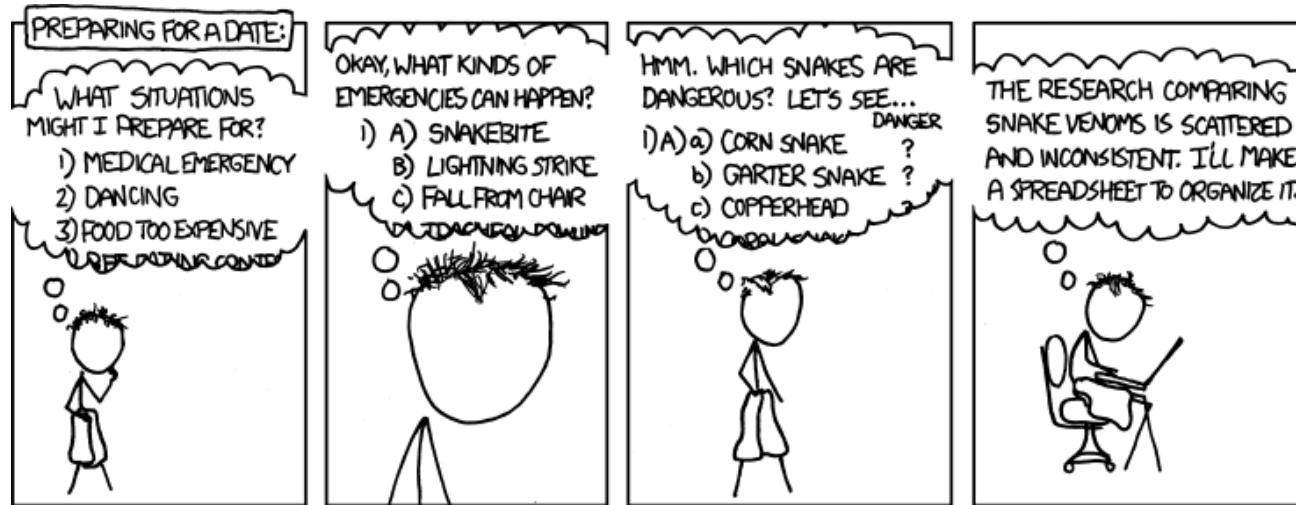
**30 sec**

- Breath in and out 3 times

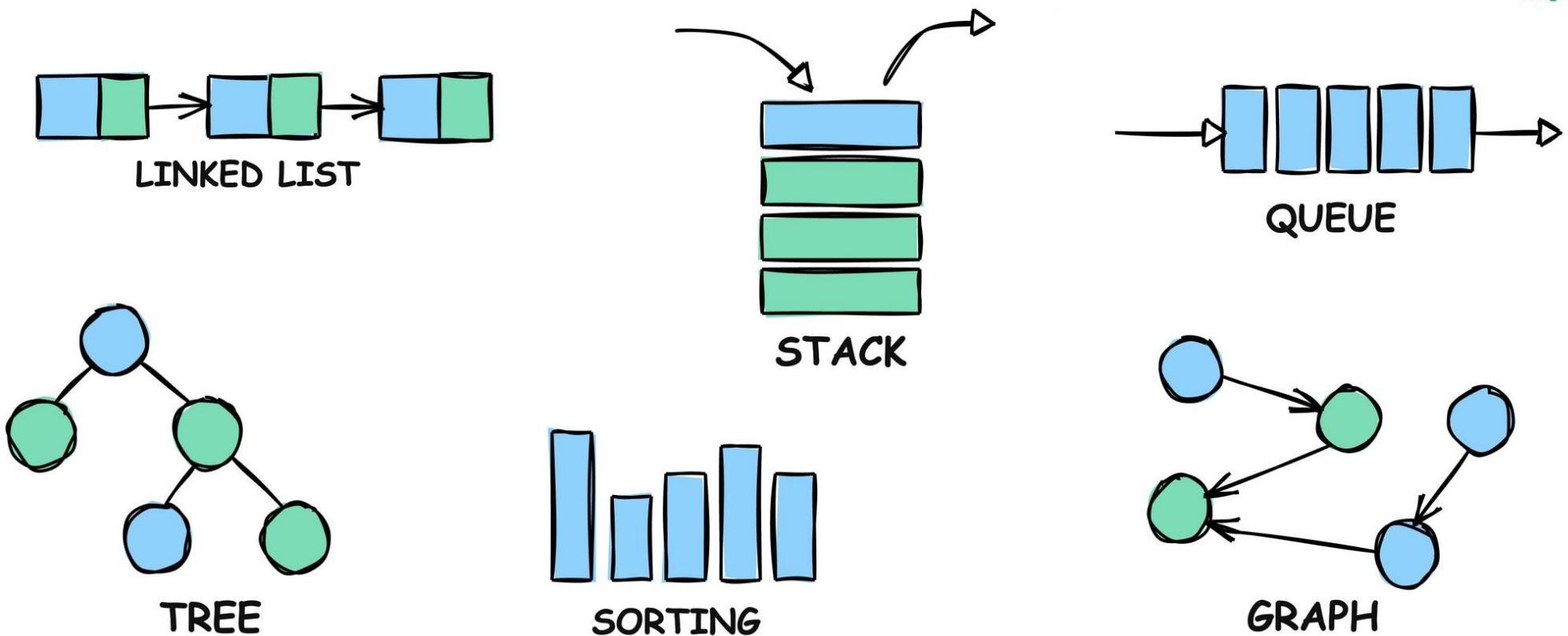
# Cooling down

10:25-10:30

- Discuss how you are doing with the course
  - What did you learn in this lecture?
  - How are you going to prepare for the next lecture?
  - How are you doing in the course in general?
- Discuss how you are doing with life?
  - How was your weekend?
  - What do you feel right now?
  - What are you looking forward to?
- Forgive me for letting the lecture run late



I REALLY NEED TO STOP USING DEPTH-FIRST SEARCHES.



# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Warming up

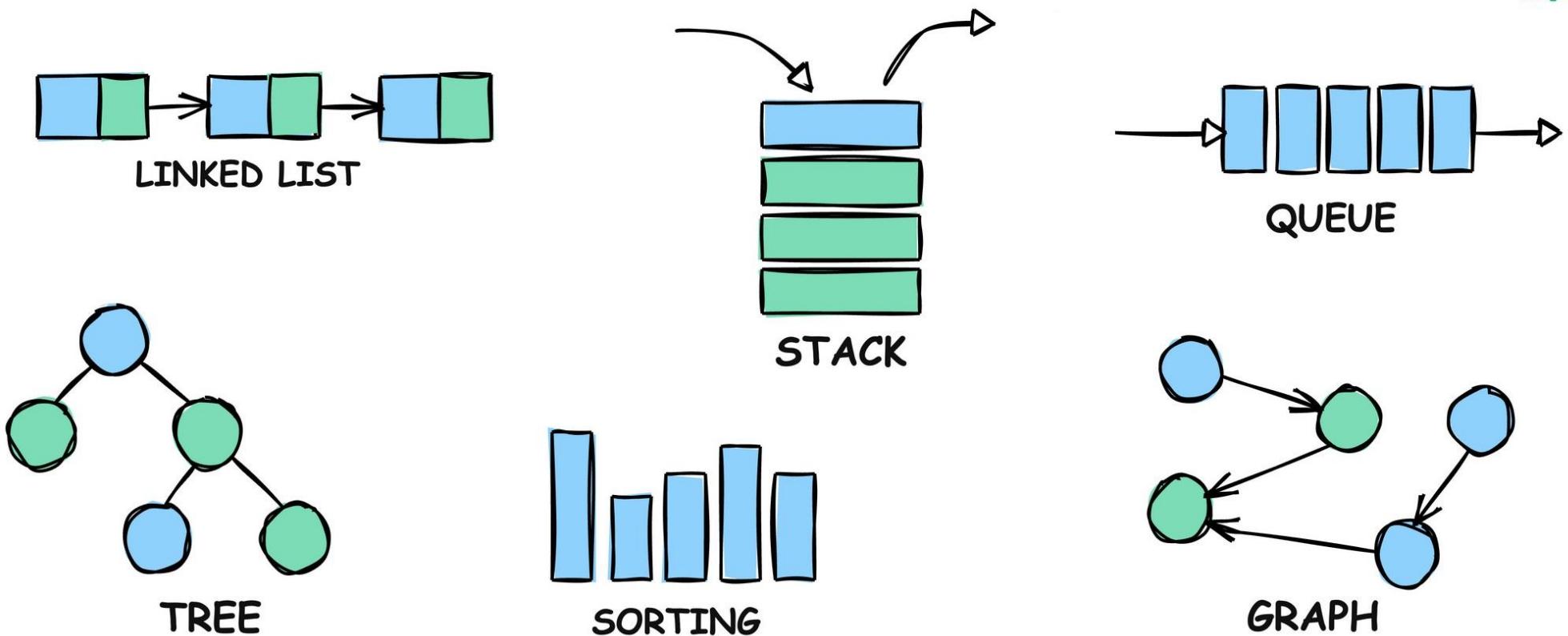
8:45-8:50

- Discuss how you are doing with the course
  - What did you learn in the preparation of this lecture?
  - What do you expect from this lecture?
  - What do you want to have learned after this lecture?
- Discuss how you are doing with life?
  - How were your last few days?
  - What do you feel right now?
  - What are you looking forward to?
- Forgive yourself for coming slightly late
  - (and have the intention to come on time next lecture)

**Focus**

**30 sec**

- Breath in and out 3 times



# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Tree traversals

Which traversal do you think is most used in practice?

- A. Breadth-first traversal
- B. Depth-first traversal

# Priority Queues

Which of the following methods is O(1)?

- A. The insert method of a priority queue implemented as an unsorted list
- B. The insert method of a priority queue implemented as a sorted list
- C. The removeMin method of a priority queue implemented as an unsorted list
- D. The removeMin method of a priority queue implemented as a sorted list

# Priority Queues

For the insert method of a priority queue implemented as a sorted list, why can we not insert the element in  $O(\log(n))$  time using Binary Search?

# Priority Queue: list-based implementations

## Unsorted list

**Insertion:** New entries are added at the end of the list, in  $O(1)$  time.

**Removal:** Access (**min**) and removal (**removeMin**) of minimal key is done by traversing list,  $O(n)$  time.

```
public class UnsortedPriorityQueue<K,V> extends AbstractPriorityQueue<K,V> { ... } Code 9.6, p. 335
```

## Sorted list

**Insertion:** Entries kept in non-decreasing order of keys. Insertion needs traversal to find position:  $O(n)$ .

**Removal:** Minimal key is always first in the list. Access (**min**) and removal (**removeMin**) are thus  $O(1)$ .

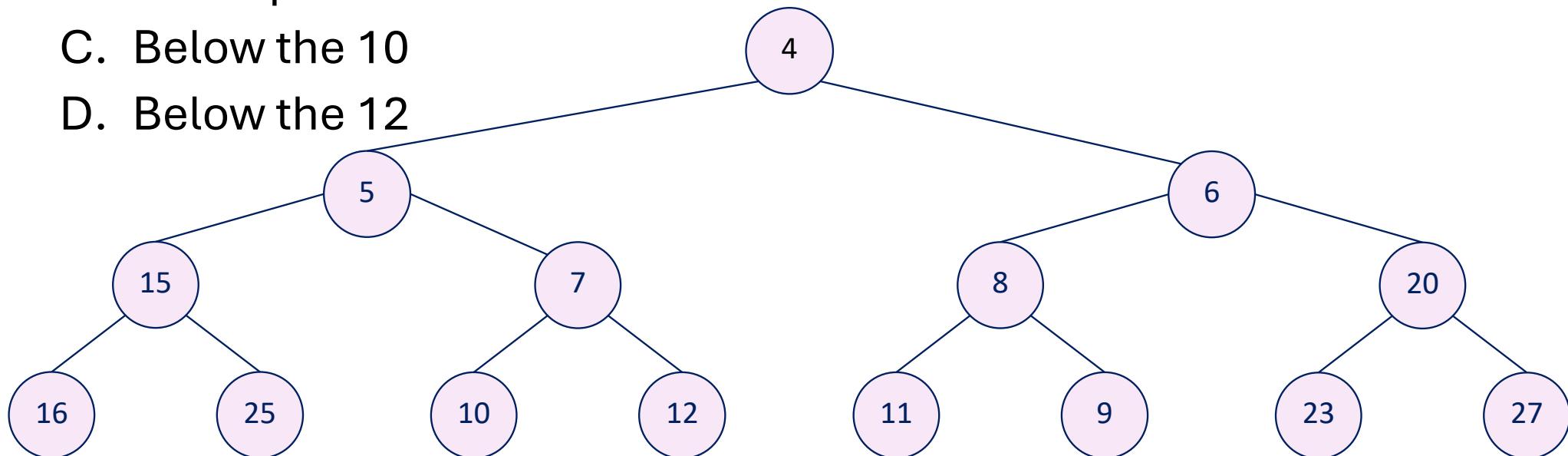
```
public class SortedPriorityQueue<K,V> extends AbstractPriorityQueue<K,V> { ... } Code 9.7, p. 337
```

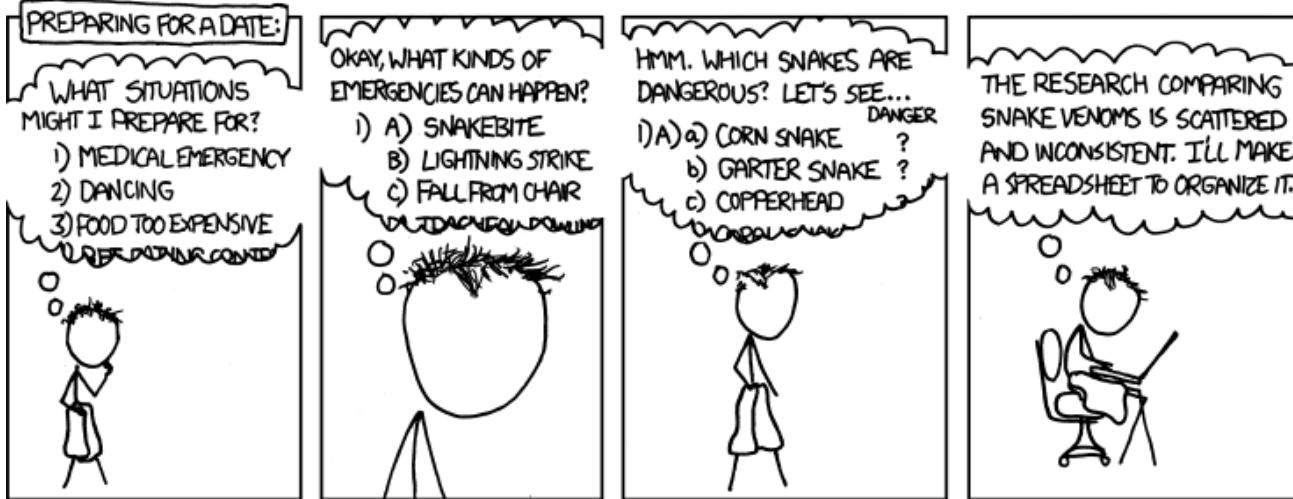
	Unsorted	Sorted
insert( <i>k,v</i> )	$O(1)$	$O(n)$
min( )	$O(n)$	$O(1)$
removeMin( )	$O(n)$	$O(1)$
size( )	$O(1)$	$O(1)$
isEmpty( )	$O(1)$	$O(1)$

# Priority Queues insertion

Consider the heap below. If the value 13 would be added to this heap, where would that value end up?

- A. At the place where the value 15 is now
- B. At the place where the value 20 is now
- C. Below the 10
- D. Below the 12

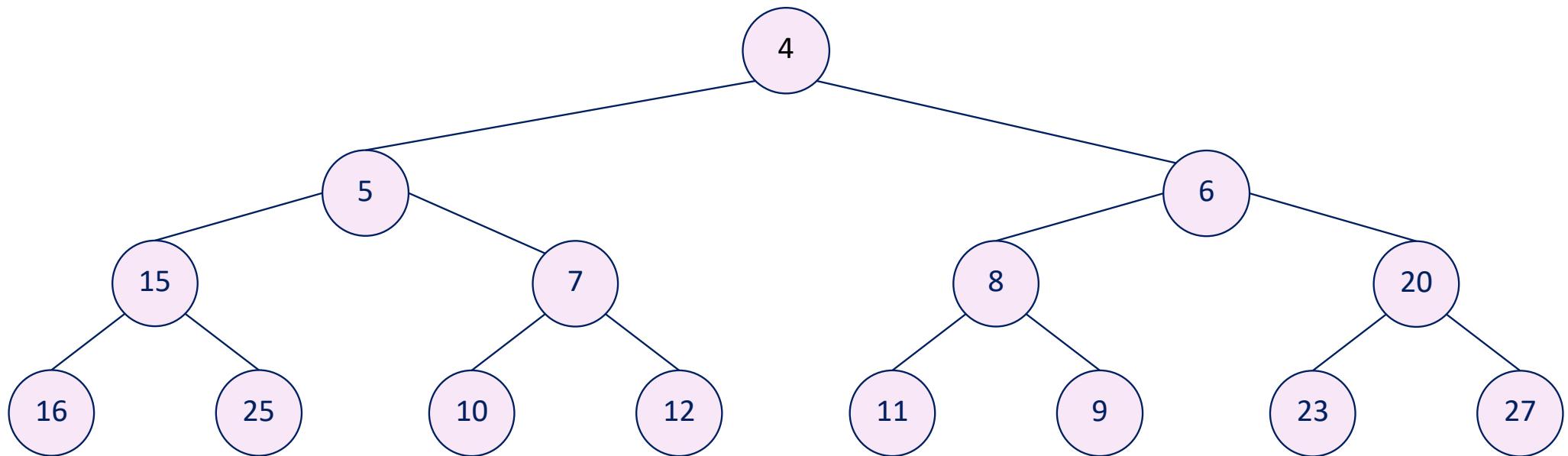




I REALLY NEED TO STOP USING DEPTH-FIRST SEARCHES.

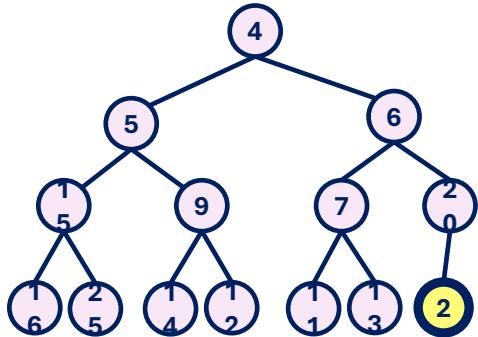
# Priority Queues deletion

Consider the heap below. If the minimum value would be removed from this heap, which values would change position?

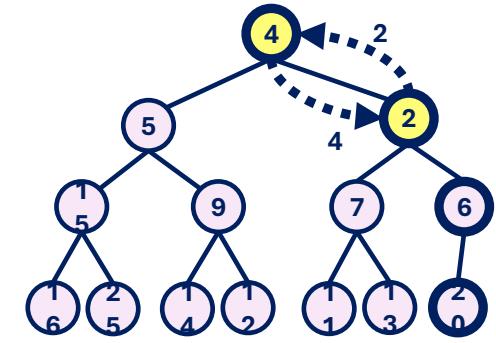
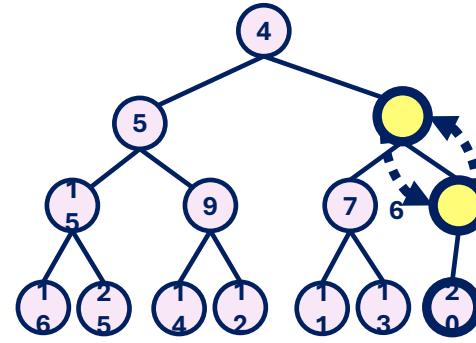
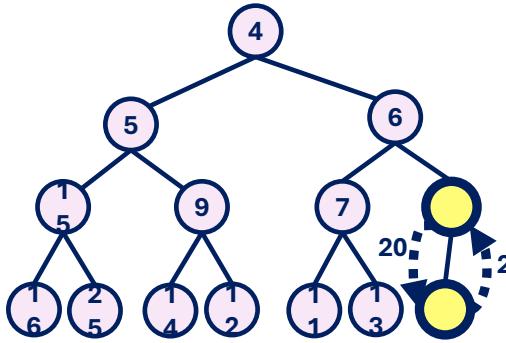


## insert

insert new element in last position

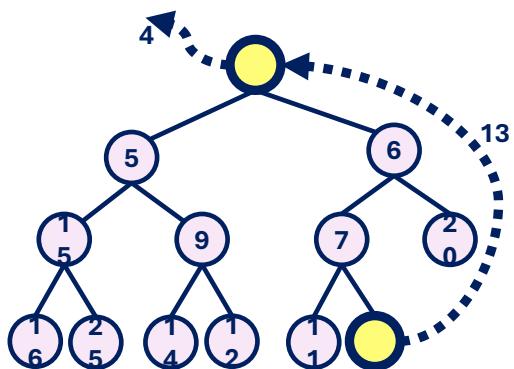


bubble up when necessary

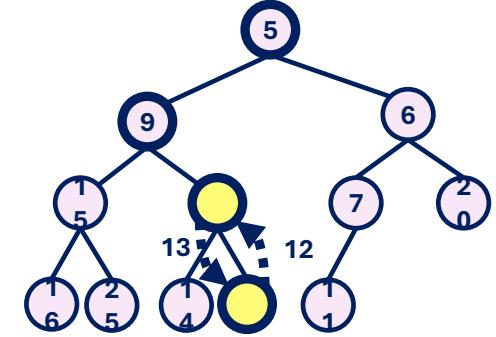
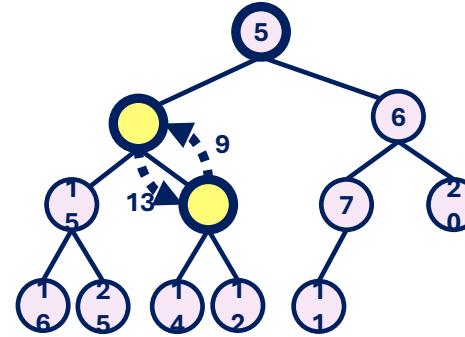
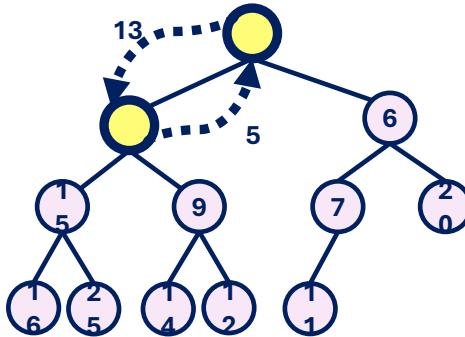


## removeMin

swap minimal element with last



bubble down when necessary



# Taxi service

- You are organizing a taxi service to bring  $n$  passengers from a fixed start location to their homes
- You are given a list of  $n$  integers, each representing the time it takes for a taxi to drive that passenger home and return to the start location
- There are  $m$  taxis available to help with this
- Passengers cannot share a taxi
- Passengers need to be helped in a fixed order, given by the list
- Calculate the minimum time it will take to bring all passengers home and return all taxis to the start location
- Your solution should run in  $O((n + m) * \log(n))$  time.

# Priority queue implementations

Consider an application using a variant of a minimum-oriented priority queue `vpx`, which initially contains  $n$  elements. This variant supports the standard insertion operation, and a custom “remove half” operation (see below). The application performs the following two steps:

- It performs  $n$  insertion operations on `vpx`, where each insertion operation adds an element to `vpx`.
- It performs one “remove half” operation on `vpx`. This “remove half” operation of the variant priority queue removes and returns the  $n$  smallest elements from `vpx`. The removed elements should be returned as a list, but the order of the elements within that list doesn’t matter.

What implementations of this variant priority queue would be best suited for this application?

- A. An unsorted list
- B. A sorted list
- C. A heap

# Priority queue implementations

Consider that we want to perform the following two types of operations on a heap:

Operation 1 - Build the heap from  $n$  given elements, we have two options:

- Initialize an empty heap and add the  $n$  elements to it.
- Alternative: initialize an unsorted list, add the  $n$  elements to it, then convert the unsorted list into a heap.

Operation 2 - Remove the minimum element from this heap  $n$  times, we also have two options:

- Remove the minimum element from the heap  $n$  times.
- Alternative: convert the heap into a sorted list, then remove the minimum element from the sorted list  $n$  times.

For which of these two operations (build the heap & remove the minimum  $n$  times) does the big-Oh time complexity improve if we use the alternative option (provided we implement it optimally)? For each alternative, you should take into account the time it takes to convert one implementation into another.

- A. The big-Oh time complexity does not improve for any of the two operations.
- B. For operation 1 (build the heap from  $n$  elements), the alternative is faster in big-Oh.
- C. For operation 2 (remove the minimum element  $n$  times), the alternative is faster in big-Oh.
- D. For both operations 1 and 2, the alternatives are faster in big-Oh.

# Bottom-up heap construction

Consider the bottom-up construction of an array-based heap. What happens after the first phase in which all entries are added to the array?

- A. Down-heap bubbling from the root node.
- B. Down-heap bubbling from all nodes except the ones in the last level, starting from the root node and ending at the last node in the last but one level.
- C. Down-heap bubbling from all nodes except the ones in the last level, starting from the last node in the last but one level and ending at the root node.
- D. Up-heap bubbling from the root node.

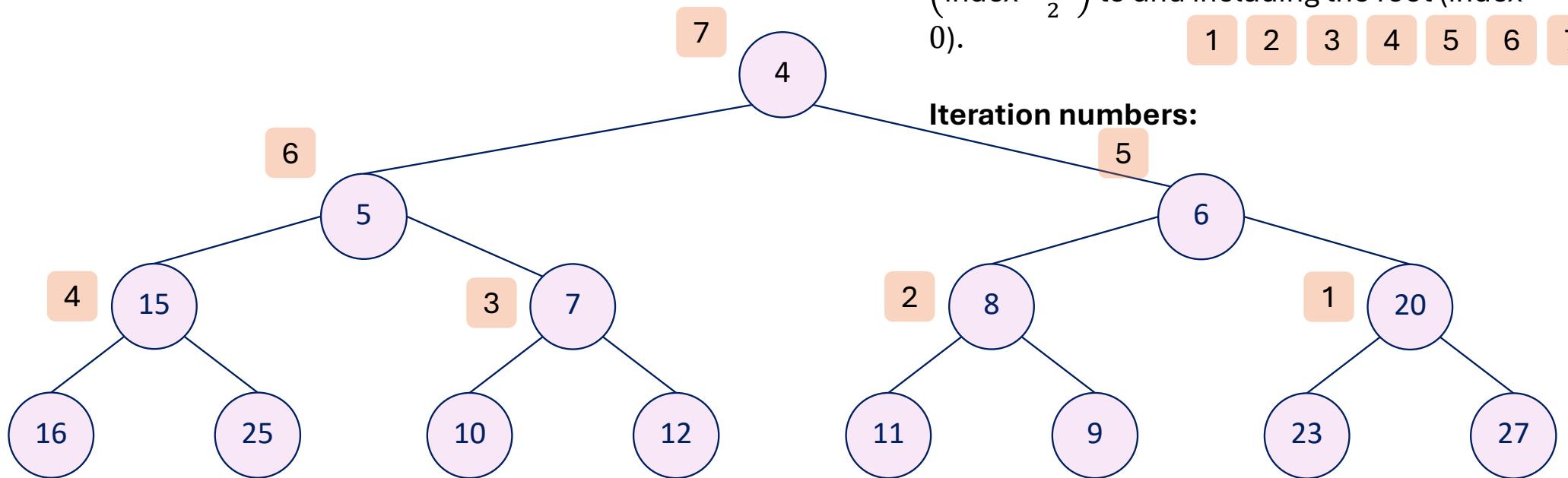
# Heapify (bottom-up array-heap construction)

**Downheap from individual nodes in this order:**

Backwards from the parent of the last node  
 $\left(\text{index } \frac{n-1}{2}\right)$  to and including the root (index 0).

1 2 3 4 5 6 7

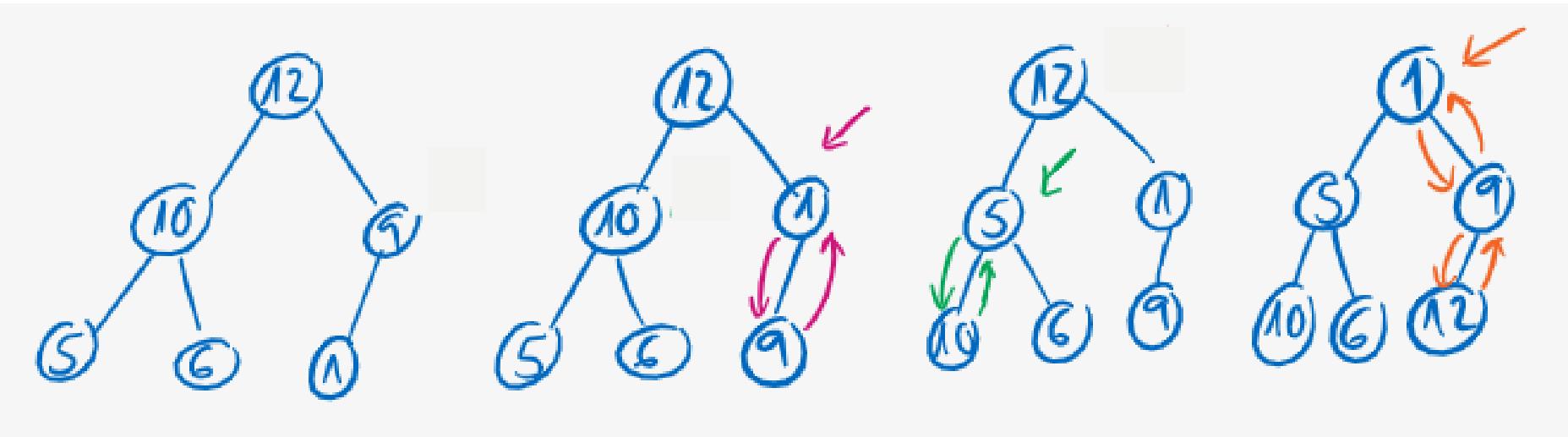
**Iteration numbers:**



# Bottom-up heap construction

Consider the array of elements [12, 10, 9, 5, 6, 1] in the given order. What is the content of the array after applying the heapify algorithm to build a min-heap?

- A. [1, 5, 6, 9, 10, 12]
- B. [1, 5, 12, 10, 6, 9]
- C. [1, 5, 9, 10, 6, 12]
- D. [1, 5, 6, 10, 9, 12]



[12, 10, 9, 5, 6, 1]

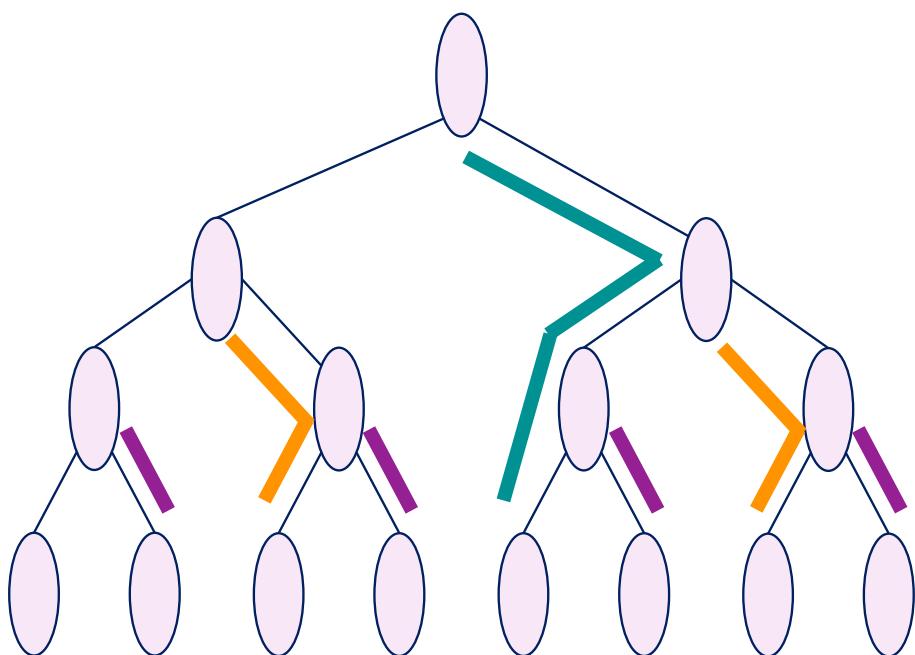
[12, 10, 1, 5, 6, 9]

[12, 5, 1, 10, 6, 9]

[1, 5, 9, 10, 6, 12]

# Bottom-up heap construction

Time complexity  $O(n)$



**Cost of downheap per node at height  $i$ .**

Cost is proportional to the height of the node's subtree, since downheap will perform swaps along a single path until it reaches a leaf.

$$\sum_{i=0}^{\lfloor \log_2 n \rfloor} \frac{n+1}{2^{i+1}} i = \sum_{i=0}^{\lfloor \log_2 n \rfloor} \frac{n+1}{2^i \cdot 2} i = \sum_{i=0}^{\lfloor \log_2 n \rfloor} \frac{n+1}{2^i} i$$

Number of nodes at height  $i$ .

Same as multiplying by  $\frac{1}{2}$ : a constant, can be dropped for big-Oh purposes.

$$= (n+1) \sum_{i=0}^{\lfloor \log_2 n \rfloor} \frac{i}{2^i}$$

converges to a constant

# Bottom-up heap construction

Time complexity  $O(n)$

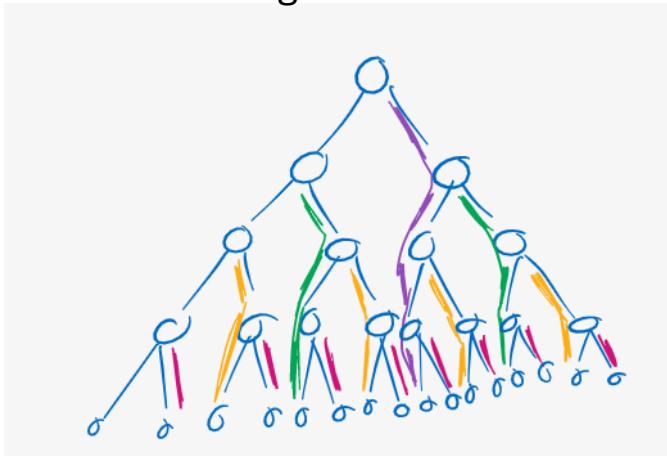
If we draw one downheap path of maximal length for every node in the tree, so that we take unused edges first...

Each edge of the tree is used by a downheap path at most once!

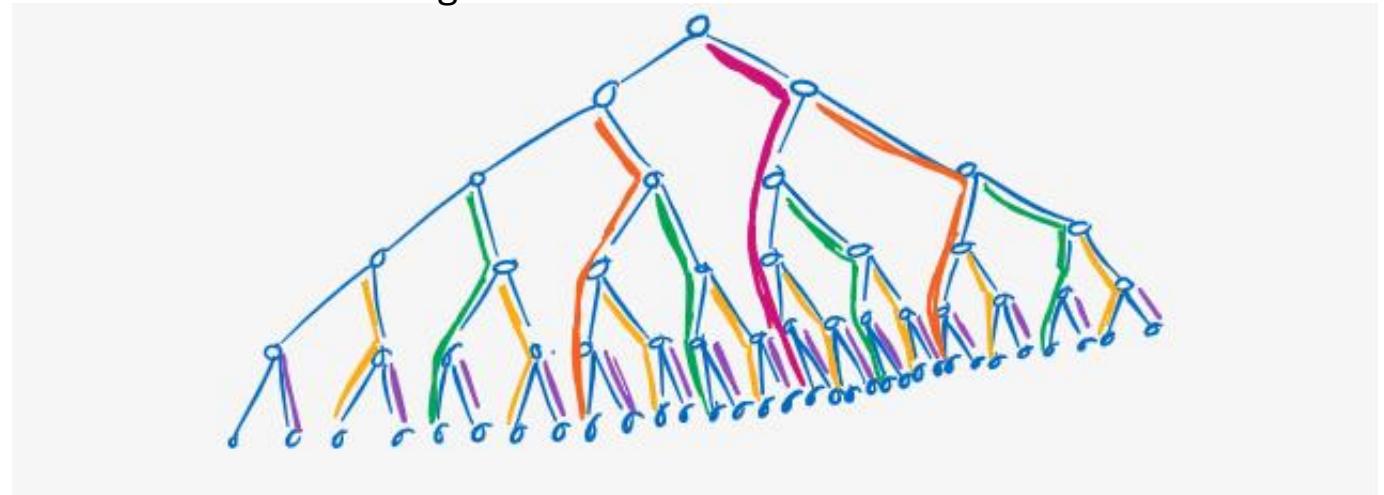
Since there are  $O(n)$  edges\* in the tree and each is used at most once, bottom-up heap construction takes  $O(n)$  time.

\* Note: Why are there  $O(n)$  edges? There are  $n$  nodes in the tree. Every node has one edge linking it to its parent, except for the root (which has no parent).

Full tree of height 4



Full tree of height 5



**Focus**

**30 sec**

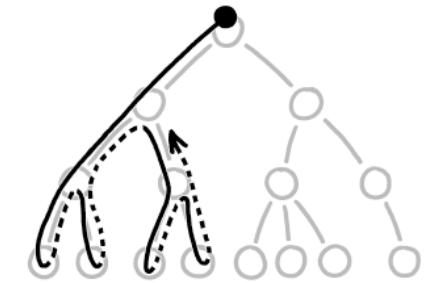
- Breath in and out 3 times

# Cooling down

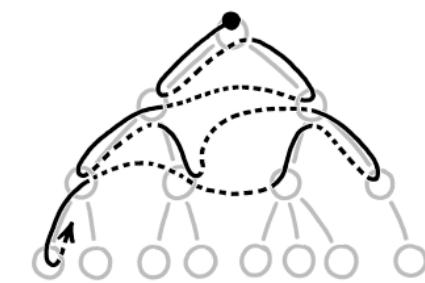
10:25-10:30

- Discuss how you are doing with the course
  - What did you learn in this lecture?
  - How are you going to prepare for the next lecture?
  - How are you doing in the course in general?
- Discuss how you are doing with life?
  - How were your last few days?
  - What do you feel right now?
  - What are you looking forward to?
- Forgive me for letting the lecture run late

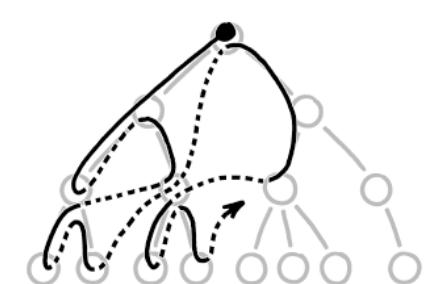
DEPTH-FIRST SEARCH



BREADTH-FIRST SEARCH



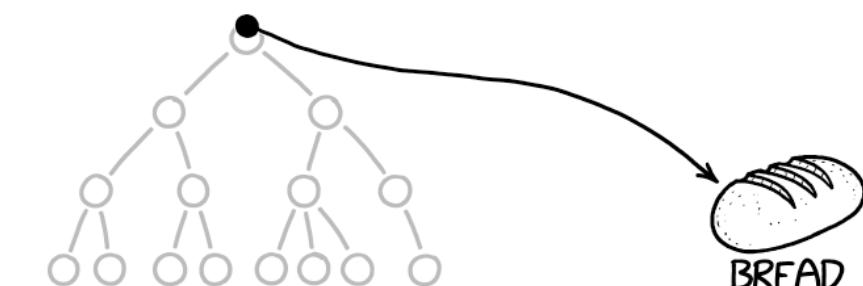
BREPTH-FIRST SEARCH

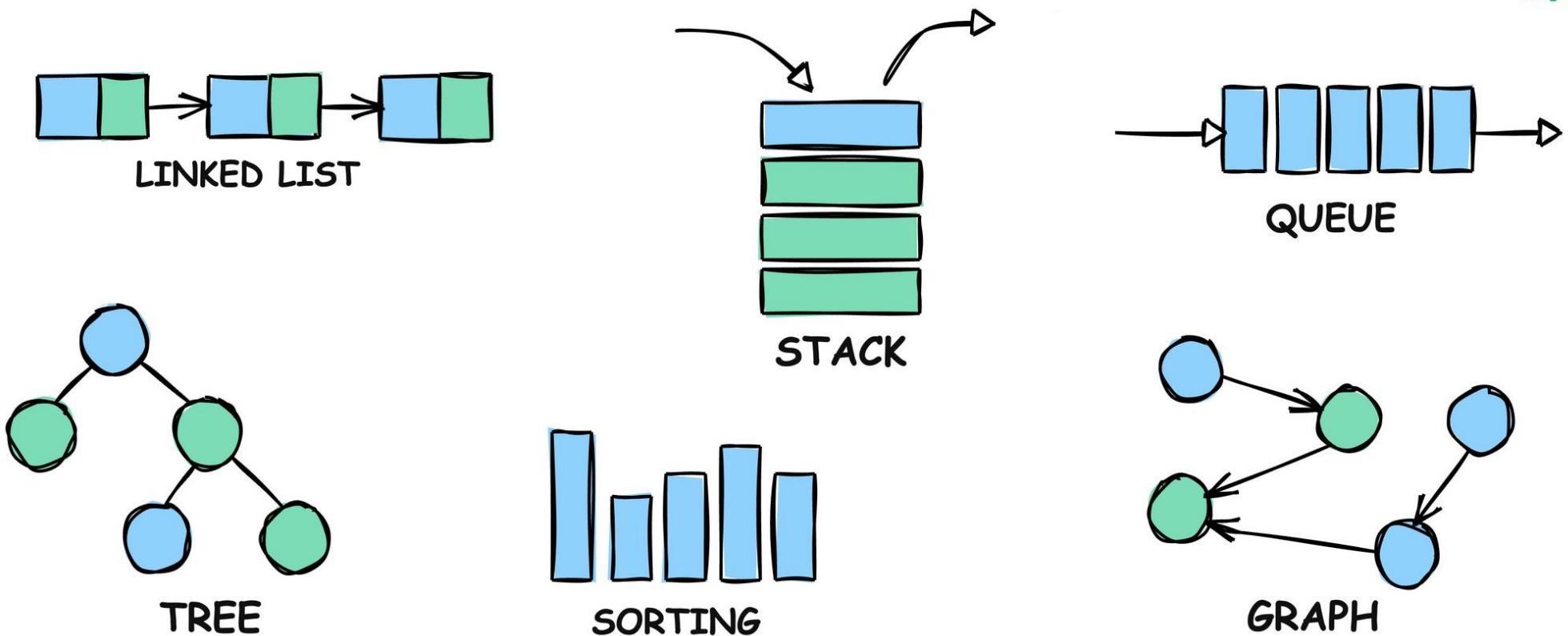


DEADTH-FIRST SEARCH



BREAD-FIRST SEARCH





# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Warming up

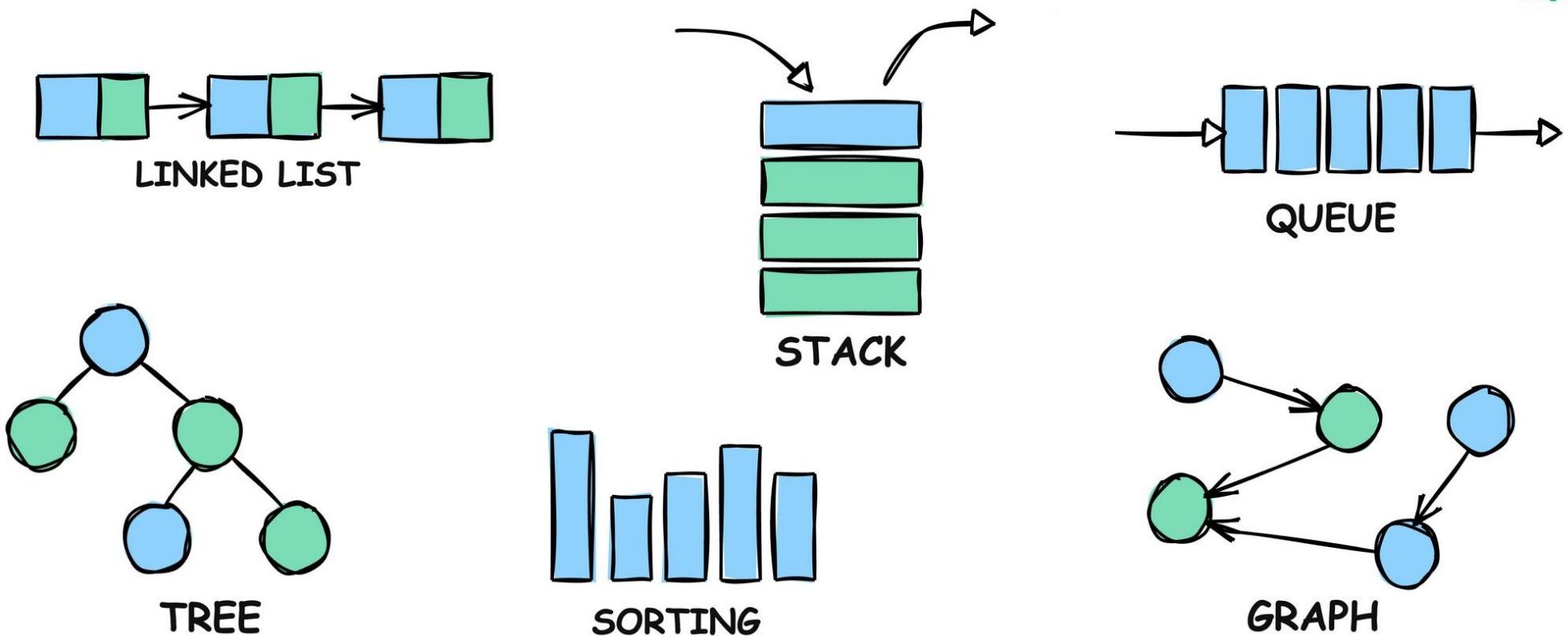
8:45-8:50

- Discuss how you are doing with the course
  - What did you learn in the preparation of this lecture?
  - What do you expect from this lecture?
  - What do you want to have learned after this lecture?
- Discuss how you are doing with life?
  - How was your weekend?
  - What do you feel right now?
  - What are you looking forward to?
- Forgive yourself for coming slightly late
  - (and have the intention to come on time next lecture)

**Focus**

**30 sec**

- Breath in and out 3 times



# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Sorting algorithms

Which sorting algorithm is implemented in the method sort?

- A. Heap sort
- B. Insertion sort
- C. Merge sort
- D. Selection sort

```
1 public void sort(int a[]) {  
2     for (int i = 0; i < a.length-1; i++) {  
3         int idx = i;  
4         for (int j = i+1; j < a.length; j++)  
5             if (a[j] < a[idx])  
6                 idx = j;  
7         swap(a, idx, i);  
8     }  
9 }
```

# Insertion sort

If insertion sort is implemented with 2 for loops, in what direction do both for loops loop over the list?

- A. Outer loop: right, inner loop: right
- B. Outer loop: right, inner loop: left
- C. Outer loop: left, inner loop: right
- D. Outer loop: left, inner loop: left

# Insertion/selection sort

Consider an unsorted list, which is input in an in-place selection or insertion sort algorithm. In the middle of this algorithm, it is suddenly stopped, and only the first  $k$  elements are sorted. You can assume that  $k \ll n$ . What can we say about the time complexity of getting to this state, and about this state in general?

- A. The insertion sort algorithm has a faster time complexity to get to this state than the selection sort algorithm
- B. The selection sort algorithm has a faster time complexity to get to this state than the insertion sort algorithm
- C. If the insertion sort algorithm was used, the first  $k$  positions are the smallest  $k$  values of the list
- D. If the selection sort algorithm was used, the first  $k$  positions are the smallest  $k$  values of the list

# Sorting with a Priority Queue

If you add  $n$  elements to a priority queue and then remove  $n$  elements from a priority queue and add them to a list, the list is sorted. Which sorting algorithm resembles each implementation of the priority queue?

- A. Unsorted list: Heap sort
- B. Unsorted list: Insertion sort
- C. Unsorted list: Selection sort
- D. Sorted list: Heap sort
- E. Sorted list: Insertion sort
- F. Sorted list: Selection sort
- G. Heap: Heap sort
- H. Heap: Insertion sort
- I. Heap: Selection sort

# Heap sort

What is true about the heap sort algorithm?

- A. Each element is swapped at most  $n$  times
- B. If the algorithm is stopped halfway while it is running, the first  $n/2$  elements are sorted
- C. It can be implemented in-place
- D. Its time complexity is  $O(n \log(n))$

# Merge sort

What is true about the merge sort algorithm?

- A. Each element is accessed/moved at most  $n$  times
- B. If the algorithm is stopped halfway while it is running, the first  $n/2$  elements are sorted
- C. It can be implemented in-place
- D. Its time complexity is  $O(n \log(n))$

# Merge sort

Write down pseudocode for the merge sort algorithm.

# Merge sort

Spot the mistakes:

```
public static PriorityQueue<Integer> mergeSort(PriorityQueue<Integer> list) {  
    PriorityQueue<Integer> list1 = new PriorityQueue<>();  
    PriorityQueue<Integer> list2 = new PriorityQueue<>();  
    for(int i = 0; i < list.size()/2; i++) {  
        list1.offer(list.poll());  
    }  
    for(int i = 0; i < list.size()/2; i++) {  
        list2.offer(list.poll());  
    }  
    mergeSort(list1);  
    mergeSort(list2);  
    list = new PriorityQueue<>();  
    while(!list1.isEmpty() || !list2.isEmpty()) {  
        if(list1.peek() < list2.peek()) {  
            list.offer(list1.poll());  
        } else {  
            list.offer(list2.poll());  
        }  
    }  
    return list;  
}
```

**Focus**

**30 sec**

- Breath in and out 3 times

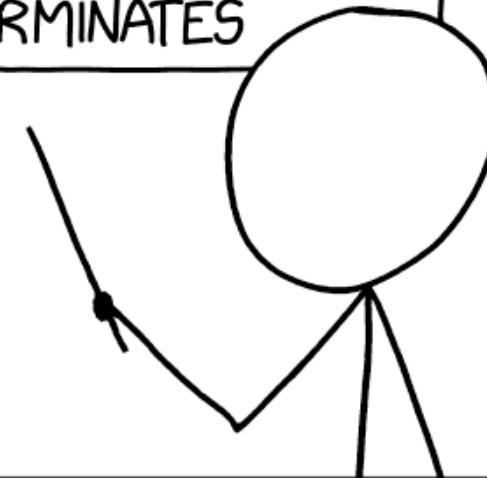
# Cooling down

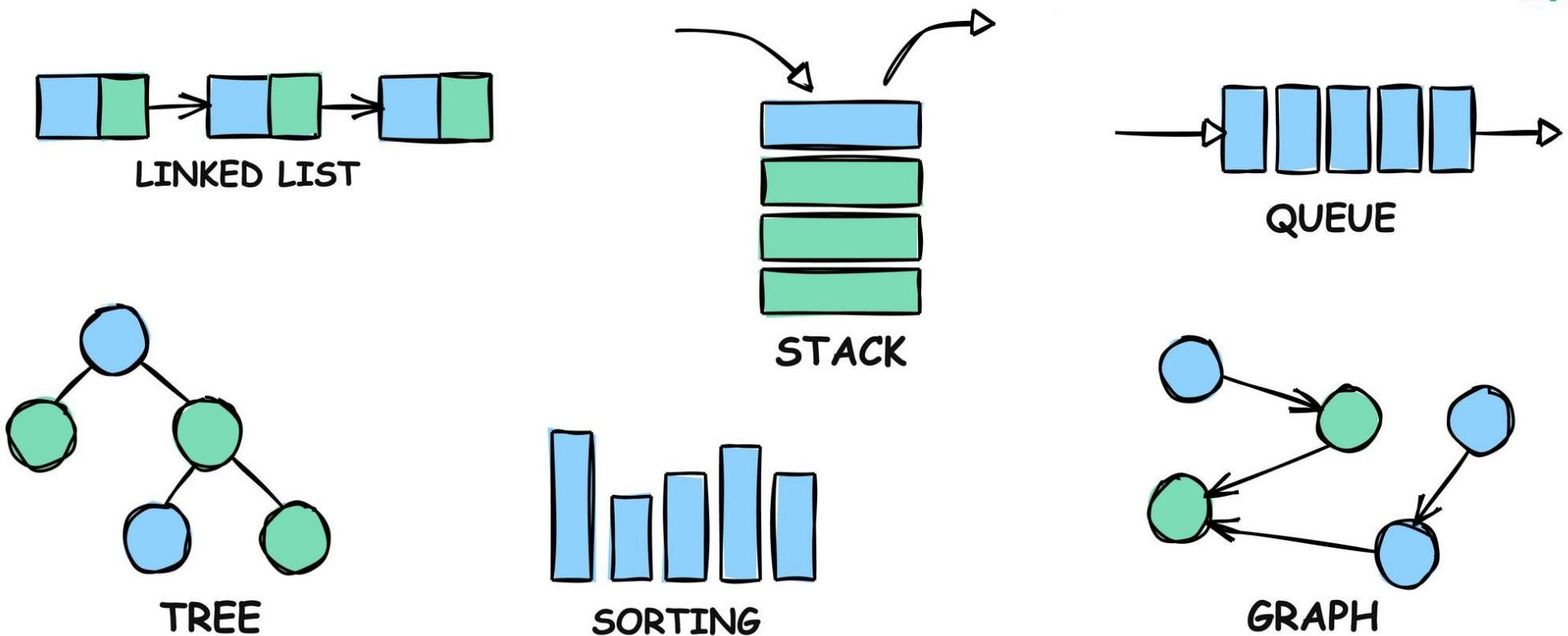
10:25-10:30

- Discuss how you are doing with the course
  - What did you learn in this lecture?
  - How are you going to prepare for the next lecture?
  - How are you doing in the course in general?
- Discuss how you are doing with life?
  - How was your weekend?
  - What do you feel right now?
  - What are you looking forward to?
- Forgive me for letting the lecture run late

## RESULTS OF ALGORITHM COMPLEXITY ANALYSIS:

AVERAGE CASE	$O(N \log N)$
BEST CASE	ALGORITHM TURNS OUT TO BE UNNECESSARY AND IS HALTED, THEN CONGRESS ENACTS SURPRISE DAYLIGHT SAVING TIME AND WE GAIN AN HOUR
WORST CASE	TOWN IN WHICH HARDWARE IS LOCATED ENTERS A GROUNDHOG DAY SCENARIO, ALGORITHM NEVER TERMINATES





# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Warming up

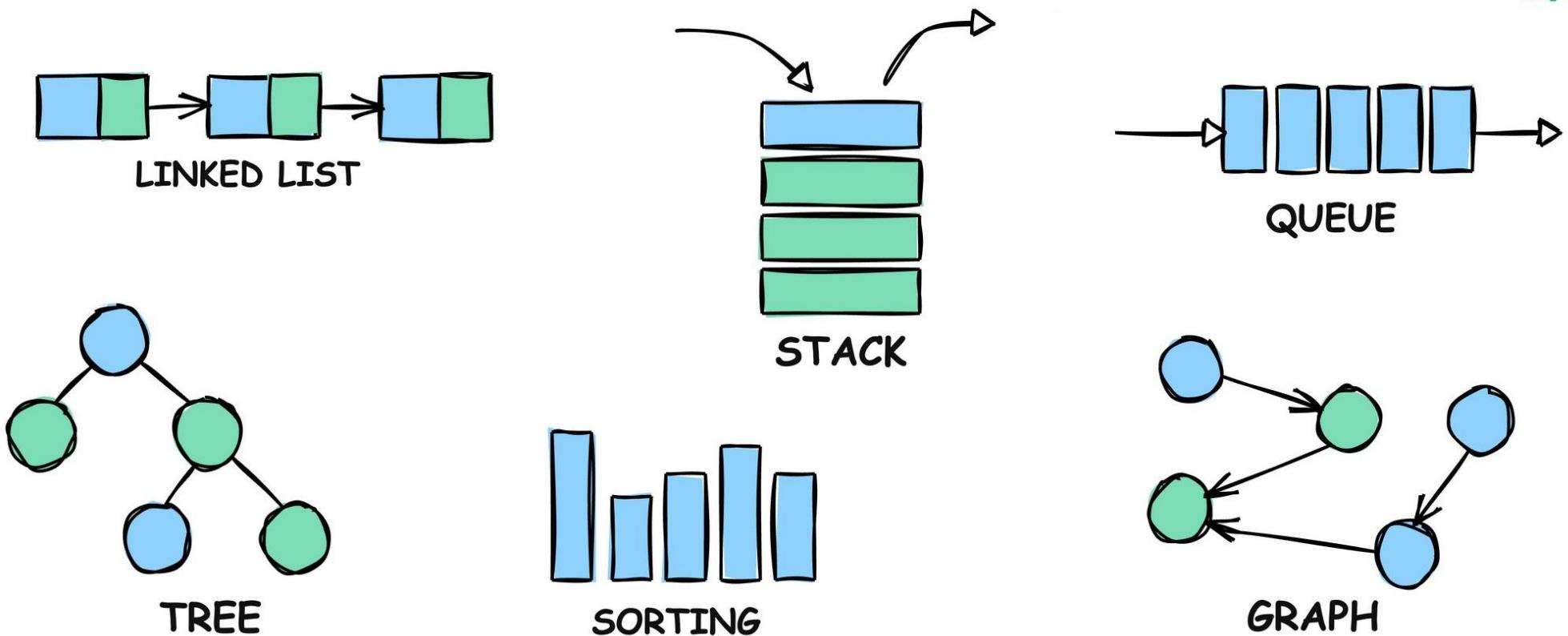
8:45-8:50

- Discuss how you are doing with the course
  - What did you learn in the preparation of this lecture?
  - What do you expect from this lecture?
  - What do you want to have learned after this lecture?
- Discuss how you are doing with life?
  - How were your last few days?
  - What do you feel right now?
  - What are you looking forward to?
- Forgive yourself for coming slightly late
  - (and have the intention to come on time next lecture)

**Focus**

**30 sec**

- Breath in and out 3 times



# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Quick sort

What is true about the quick sort algorithm?

- A. Each element is accessed/moved at most  $O(n)$  times
- B. If the algorithm is stopped halfway while it is running, the first  $n/2$  elements are sorted
- C. It can be implemented in-place
- D. Its time complexity is  $O(n \log(n))$

# Randomized quick sort

In what applications would randomized quick sort be faster than normal quick sort?

# Time complexity of sorting algorithms

Which sorting algorithms can be implemented in  $O(n \log(n))$ ?

- A. Heap sort
- B. Insertion sort
- C. Merge sort
- D. Quick sort
- E. Selection sort

# Quick sort time complexity

Analyze the runtime complexity of the quicksort algorithm by analyzing the pseudocode of a recursive quick sort algorithm and unfolding the recurrence equation.

# In-place

Which sorting algorithms can be implemented in-place?

- A. Heap sort
- B. Insertion sort
- C. Merge sort
- D. Quick sort
- E. Selection sort

# Almost sorted lists

Which sorting algorithms run in  $O(n)$  if the input list is almost sorted (there are  $O(n)$  swaps of adjacent neighbors needed to sort the list)?

- A. Heap sort
- B. Insertion sort
- C. Merge sort
- D. Quick sort
- E. Selection sort

# Stability

For which sorting algorithm is it true that if elements have the same value, the elements that are earlier in the list stay earlier in the list?

- A. Heap sort
- B. Insertion sort
- C. Merge sort
- D. Quick sort
- E. Selection sort

# Quicker quick sort

If we adjust the quick sort algorithm where we skip one of the recursive calls (which call is chosen randomly), what can you say about the list after the algorithm is done? Which of these lists can be the output of such an algorithm?

- A. [0, 4, 6, 8, 1, 3, 7, 2, 5, 9]
- B. [2, 3, 4, 6, 9, 0, 1, 5, 7, 8]
- C. [3, 1, 2, 0, 4, 5, 9, 7, 6, 8]
- D. [3, 4, 0, 2, 1, 5, 6, 7, 8, 9]

# Quick select

Analyze the runtime complexity of this adjusted quicksort algorithm by analyzing the pseudocode of a recursive quick select algorithm and unfolding the recurrence equation.

**Focus**

**30 sec**

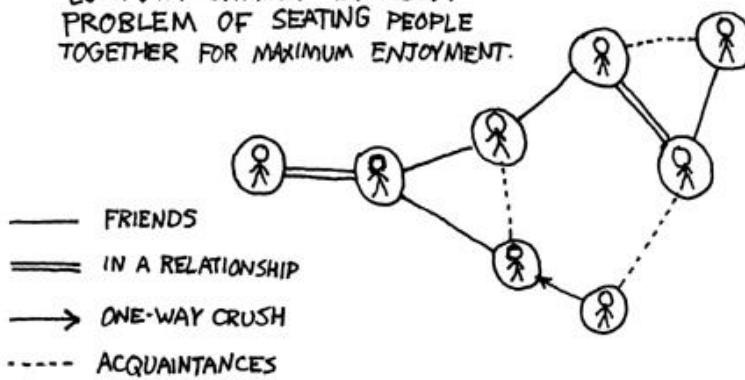
- Breath in and out 3 times

# Cooling down

10:25-10:30

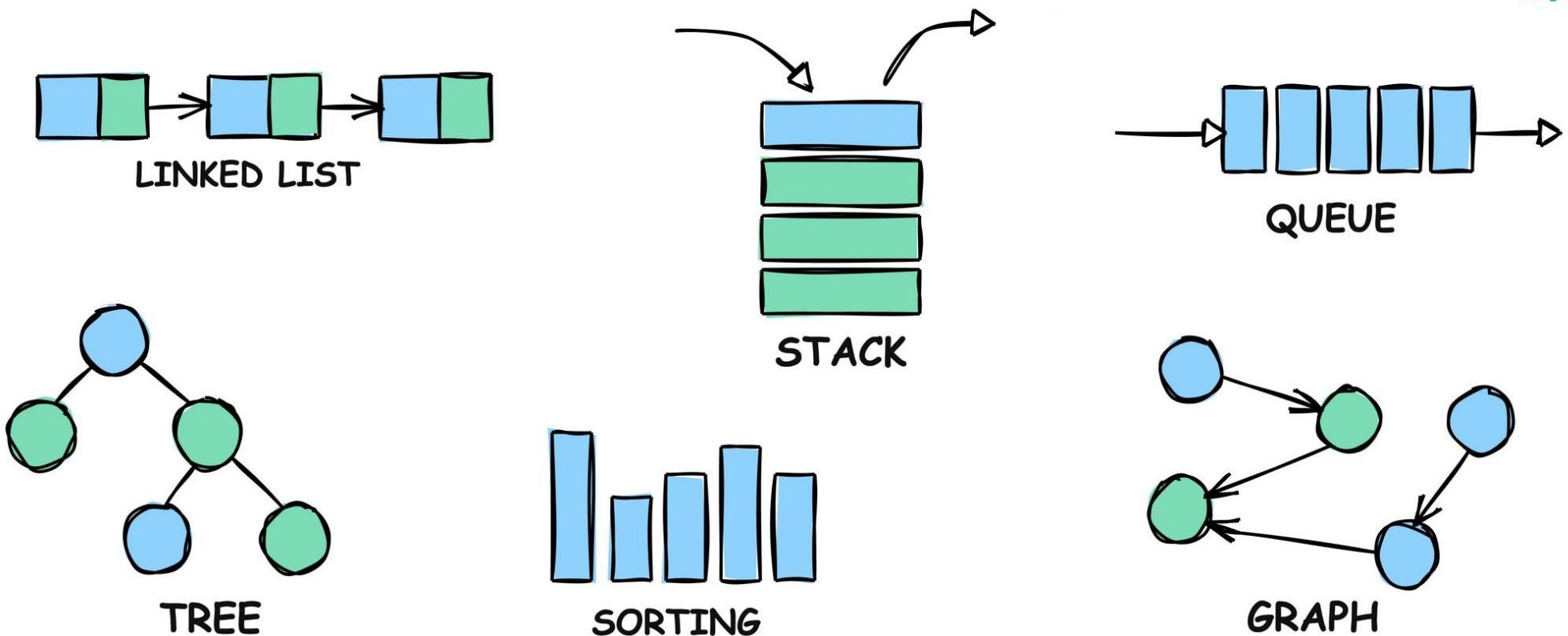
- Discuss how you are doing with the course
  - What did you learn in this lecture?
  - How are you going to prepare for the next lecture?
  - How are you doing in the course in general?
- Discuss how you are doing with life?
  - How were your last few days?
  - What do you feel right now?
  - What are you looking forward to?
- Forgive me for letting the lecture run late

AT THE MOVIES, I GET FRUSTRATED  
WHEN WE FILE INTO OUR ROW  
HAPHAZARDLY, IGNORING THE  
COMPUTATIONALLY DIFFICULT  
PROBLEM OF SEATING PEOPLE  
TOGETHER FOR MAXIMUM ENJOYMENT.



GUYS! THIS IS NOT  
SOCIALLY OPTIMAL!





# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Comparing Sorting Algorithms

- Time Complexity
- Auxiliary space requirements
- Stability

# Time Complexity

You are developing a **flight scheduling system** for an airport. Flight departure times are usually scheduled **in ascending order**, but due to **weather conditions and air traffic delays**, some flights take off **slightly earlier or later than expected**. This results in a **few misordered entries** in the system. Here is an example of recorded departure times:

[06:00, 06:15, 06:30, 06:45, 07:00, 07:30, 07:15, 07:45, 08:00]

- A. Quick Sort
- B. Heap Sort
- C. Insertion Sort
- D. Merge Sort

# Time complexity

You are developing a **flight scheduling system** for an airport. Flight departure times are usually scheduled **in ascending order**, but due to **weather conditions and air traffic delays**, some flights take off **slightly earlier or later than expected**. This results in a **few misordered entries** in the system. Here is an example of recorded departure times:

[06:00, 06:15, 06:30, 06:45, 07:00, 07:30, 07:15, 07:45, 08:00]

- A. Quick Sort
- B. Heap Sort
- C. **Insertion Sort**
- D. Merge Sort

Insertion sort runs in  $O(n+m)$  time (where  $m$  is the number of inversions required) and is linear when  $m$  is small.

All other options need at least  $O(n \log n)$  time.

(1 point) Consider the following method `csort`, which implements a sorting algorithm. The input array is guaranteed to contain integer values between 0 and  $k$ .

```
1 public static void csort(int[] array, int k) {  
2     int temp[] = new int[k + 1];  
3  
4     for (int e : array)  
5         temp[e]++;  
6  
7     int ndx = 0;  
8     for (int i = 0; i < temp.length; i++)  
9         while (temp[i] > 0) {  
10             array[ndx] = i;  
11             ndx++;  
12             temp[i]--;  
13         }  
14 }
```

## Which of the following statements about method `csort` and the sorting algorithm it implements is true?

- A. It is a comparison-based sorting algorithm.
- B. It is an in-place sorting algorithm with  $O(1)$  space complexity.
- C. It runs in  $O(n)$  time, where  $n$  is the length of the input array.
- D. At each index, the array `temp` accumulates the number of elements from the original input array that are identical to such index.

1

Final exam 19-20 - MCQ 14

(1 Point)

(1 point) Consider the following method `csort`, which implements a sorting algorithm. The input array is guaranteed to contain integer values between 0 and  $k$ .

```
1 public static void csort(int[] array, int k) {  
2     int temp[] = new int[k + 1];  
3  
4     for (int e : array)  
5         temp[e]++;  
6  
7     int ndx = 0;  
8     for (int i = 0; i < temp.length; i++)  
9         while (temp[i] > 0) {  
10             array[ndx] = i;  
11             ndx++;  
12             temp[i]--;  
13         }  
14 }
```

Which of the following statements about method `csort` and the sorting algorithm it implements is **true**?

- A. It is a comparison-based sorting algorithm.
- B. It is an in-place sorting algorithm with  $\mathcal{O}(1)$  space complexity.
- C. It runs in  $\mathcal{O}(n)$  time, where  $n$  is the length of the input array.
- D. At each index, the array `temp` accumulates the number of elements from the original input array that are identical to such index.

- A. **False.** There are no comparisons made between elements.
- B. **False.** The space complexity is  $\mathcal{O}(k)$ , since we are declaring an array with  $k+1$  positions (line 2).
- C. **False.** It runs in  $\mathcal{O}(n + k)$ . It could run in  $\mathcal{O}(k)$  if  $k \gg n$ .
- D. True.** Each index contains the count of elements whose value is identical to such index.

Note: this is a simplified version of the **counting sort** algorithm.

1

Final exam 19-20 - MCQ 14  
(1 Point)

(1 point) Consider the following method `csort`, which implements counting sort. The array `array` contains integer values between 0 and  $k$ .

```

1 public static void csort(int[] array, int k) {
2     int temp[] = new int[k + 1];
3
4     for (int e : array)
5         temp[e]++;
6
7     int ndx = 0;
8     for (int i = 0; i < temp.length; i++) {
9         while (temp[i] > 0) {
10            array[ndx] = i;
11            ndx++;
12            temp[i]--;
13        }
14    }

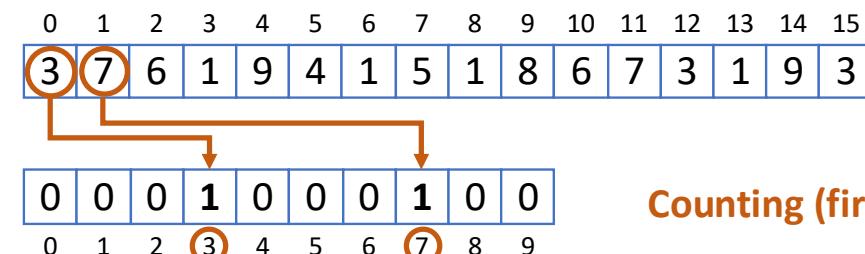
```

array

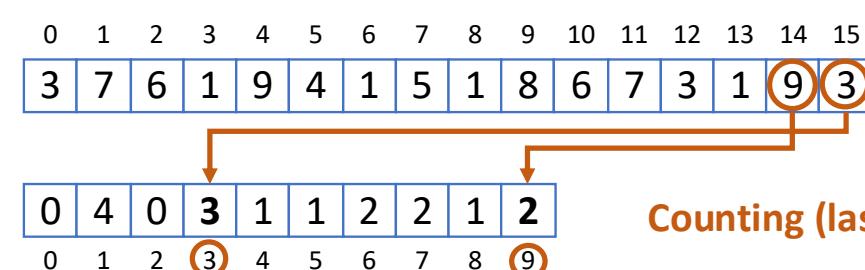
temp (counts)

array

temp (counts)



Counting (first 2 iterations)



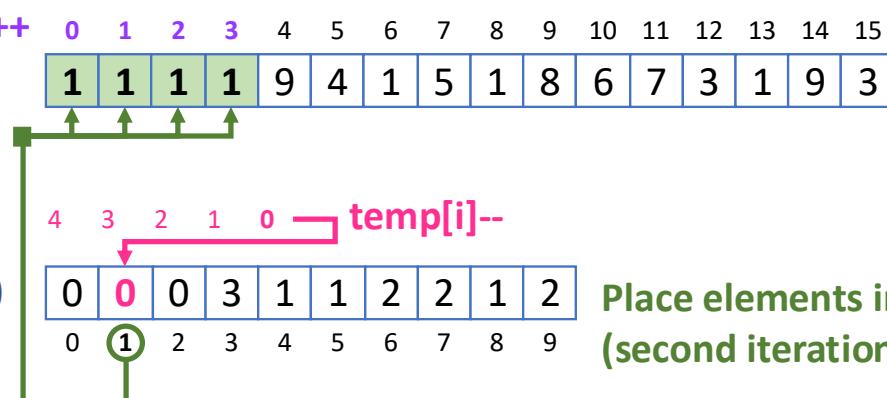
Counting (last 2 iterations)

array

array[ndx] = i

temp (counts)

ndx++



Place elements in final positions  
(second iteration)

# Sorting large data

You are working as a **database engineer** for a financial company. The company needs to **sort billions of transaction records** stored on disk. The dataset is **too large to fit into RAM**, so sorting must be performed using external storage.

Given these constraints, which of the following sorting algorithms would be the **most efficient and feasible choice**?

- A. Quick Sort
- B. Heap Sort
- C. Merge Sort
- D. Insertion Sort

# Sorting large data

You are working as a **database engineer** for a financial company. The company needs to **sort billions of transaction records** stored on disk. The dataset is **too large to fit into RAM**, so sorting must be performed in **batches** using external storage.

Given these constraints, which of the following sorting algorithms would be the **most efficient and feasible choice**?

- A. Quick Sort
- B. Heap Sort
- C. Merge Sort
- D. Insertion Sort

You are developing a **payroll system** for a company. Employee records are stored in a database with **two fields: Employee ID and Salary**. Initially, the records are **sorted by Employee ID**, but now you need to sort them **by Salary** while keeping employees with the same salary in the same order as before.

Which of the following sorting algorithms **cannot be used** to correctly maintain the relative order of employees with the same salary?

- A. Quick Sort
- B. Heap Sort
- C. Merge Sort
- D. Insertion Sort

You are developing a **payroll system** for a company. Employee records are stored in a database with **two fields: Employee ID and Salary**. Initially, the records are **sorted by Employee ID**, but now you need to sort them **by Salary** while keeping employees with the same salary in the same order as before.

Which of the following sorting algorithms **cannot be used** to correctly maintain the relative order of employees with the same salary?

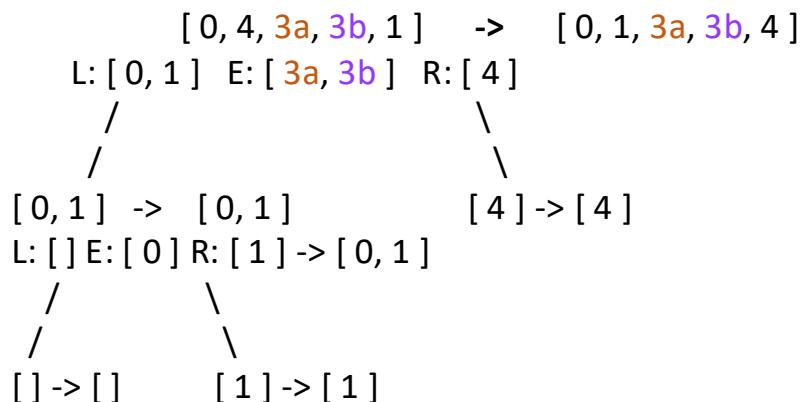
What criterion should your sorting algorithm obey?

# Quick sort (stability)

Quick sort (not in-place)

**Stable.** Partition/combine can keep relative order between entries with equal key by adding and removing elements from the sequences according to the FIFO / queue principle.

[ 0, 4, 3a, 3b, 1 ]



Quick sort (in-place)

**Not stable**, since elements with equal key can end up in either side of the pivot, and they can be swapped with any other element (incl. the pivot itself) during the partition procedure.

[4, 3a, 5, 6, 3b ]

Partition start idx 0, end idx: 4 (pivot: 3a)

[(4, 3a, 5, 6, 3b )]

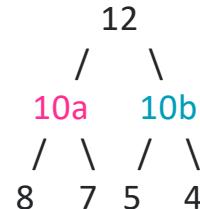
[(3b, 3a, 5, 6, 4 )]

# Heap sort (stability)

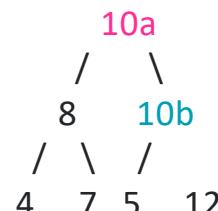
**Not stable.**

Bubbling is not guaranteed to preserve the order between elements with identical keys.

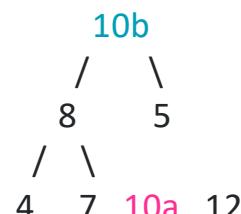
Valid for both in-place and not in-place implementations (both involve bubbling).



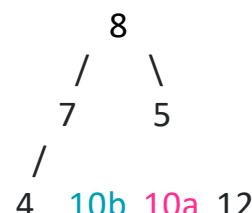
[ 12, 10a, 10b, 8, 7, 5, 4 ]  
(already as max-heap)



first removal (w/ bubbling)  
swap 12 <-> 4 (root <-> last)  
bubble 4 <-> 10a, bubble 4 <-> 8  
[ 10a, 10b, 8, 7, 5, 12 ]



first removal (w/ bubbling)  
swap 10a <-> 5 (root <-> last)  
bubble 5 <-> 10b  
[ 10b, 8, 5, 4, 7, 10a, 12 ]



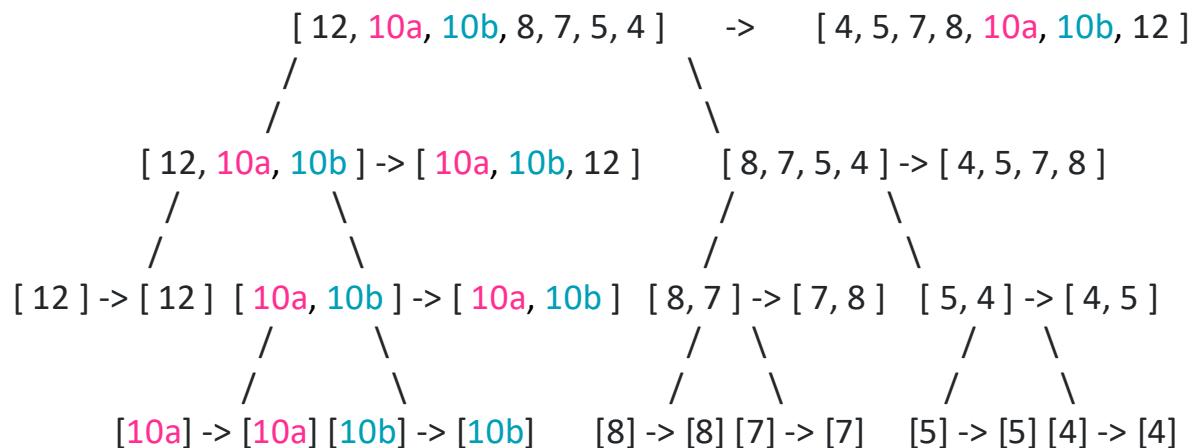
first removal (w/ bubbling)  
swap 10b <-> 7 (root <-> last)  
bubble 7 <-> 8  
[ 8, 7, 5, 4, 10b, 10a, 12 ]

# Merge sort (stability)

**Stable.**

If **a** appears before **b** (left -> right), then **a** will always end up to the left of **b** when partitioning.

When merging, stability is guaranteed as long as we keep the left/right order when the elements from the left and right subsequences have identical keys. Valid for both implementations using queues and arrays.



# Insertion sort (stability)

inserts element at its final position in the sorted part of the sequence (elements processed so far)

**Stable.** When comparing the current element to the previous elements, only shift previous elements to the right if they are strictly larger than the current element. This guarantees that the current element is inserted after (to the right of) any other elements with identical key which have already been processed so far (left to right outer loop), keeping their original order intact.

[ 0, 3a, 3b, 1, 4 ]

current

v

[ 0, 3a, 3b, 1, 4 ] (0 is smaller than 3a, no moves)

v

[ 0, 3a, 3b, 1, 4 ] (3a is equal to 3b, no moves)

v

[ 0, 1, 3a, 3b, 4 ] (3b is larger than 1, shift 3b, 3a is larger than 1, shift 3a, 0 is smaller than 1, insert 1 at index 1)

v

[ 0, 1, 3a, 3b, 4 ] (3b is smaller than 4, no moves)

# Radix Sort

3

(1 Point)

What is the content of the following sequence of elements [329, 457, 657, 839, 436, 720, 355] after two iterations of the LSD radix sort algorithm using digits as individual keys?

- [720, 329, 436, 839, 355, 457, 657]
- [720, 355, 436, 457, 657, 329, 839]
- [329, 355, 457, 436, 657, 720, 839]
- [329, 355, 436, 457, 657, 720, 839]

3

(1 Point)

What is the content of the following sequence of elements [329, 457, 657, 839, 436, 720, 355] after two iterations of the LSD radix sort algorithm using digits as individual keys?

- [720, 329, 436, 839, 355, 457, 657] ✓
- [720, 355, 436, 457, 657, 329, 839]
- [329, 355, 457, 436, 657, 720, 839]
- [329, 355, 436, 457, 657, 720, 839]

3	2	9
4	5	7
6	5	7
8	3	9
4	3	6
7	2	0
3	5	5
Sort		

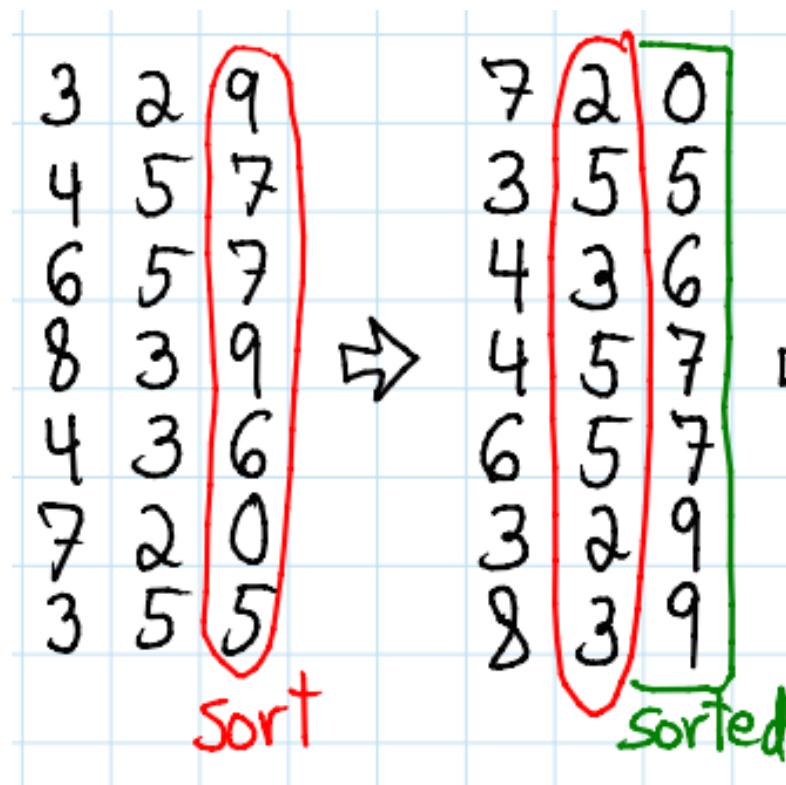
<https://brilliant.org/wiki/radix-sort/>

3

(1 Point)

What is the content of the following sequence of elements [329, 457, 657, 839, 436, 720, 355] after two iterations of the LSD radix sort algorithm using digits as individual keys?

- [720, 329, 436, 839, 355, 457, 657] ✓
- [720, 355, 436, 457, 657, 329, 839]
- [329, 355, 457, 436, 657, 720, 839]
- [329, 355, 436, 457, 657, 720, 839]



<https://brilliant.org/wiki/radix-sort/>

3

(1 Point)

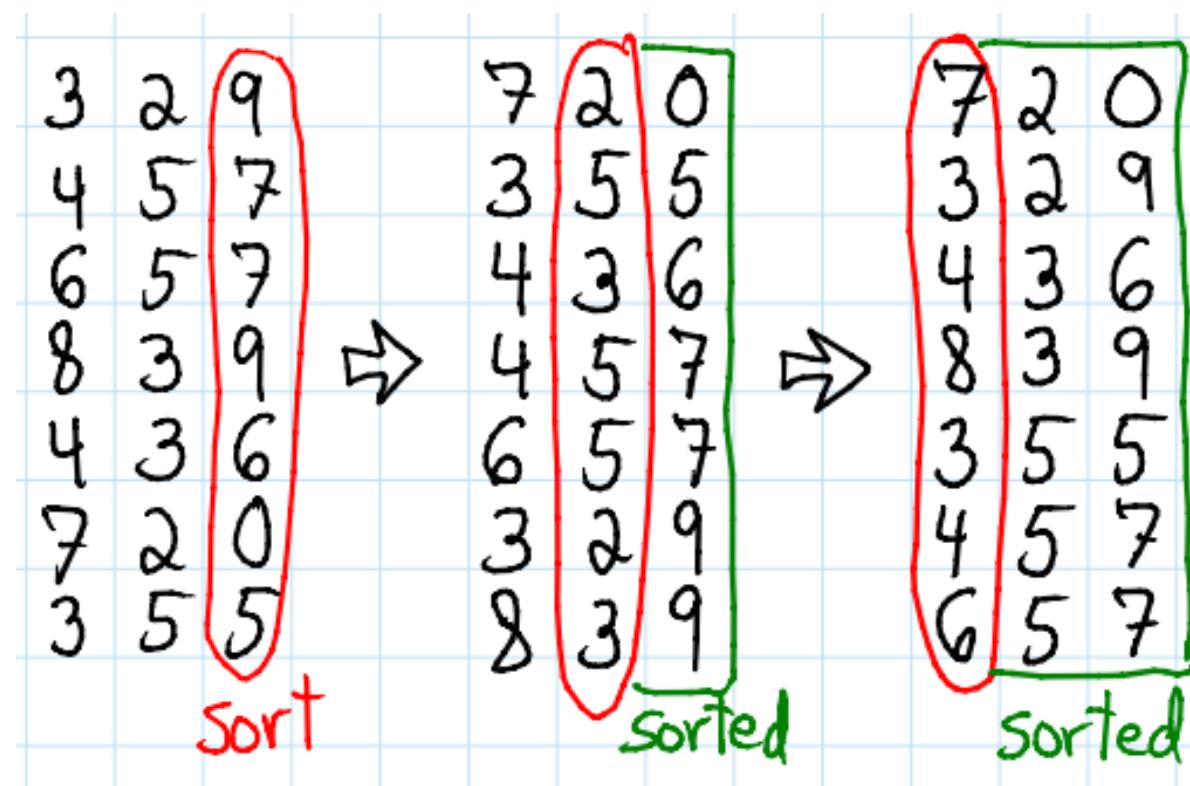
What is the content of the following sequence of elements [329, 457, 657, 839, 436, 720, 355] after two iterations of the LSD radix sort algorithm using digits as individual keys?

[720, 329, 436, 839, 355, 457, 657] ✓

[720, 355, 436, 457, 657, 329, 839]

[329, 355, 457, 436, 657, 720, 839]

[329, 355, 436, 457, 657, 720, 839]



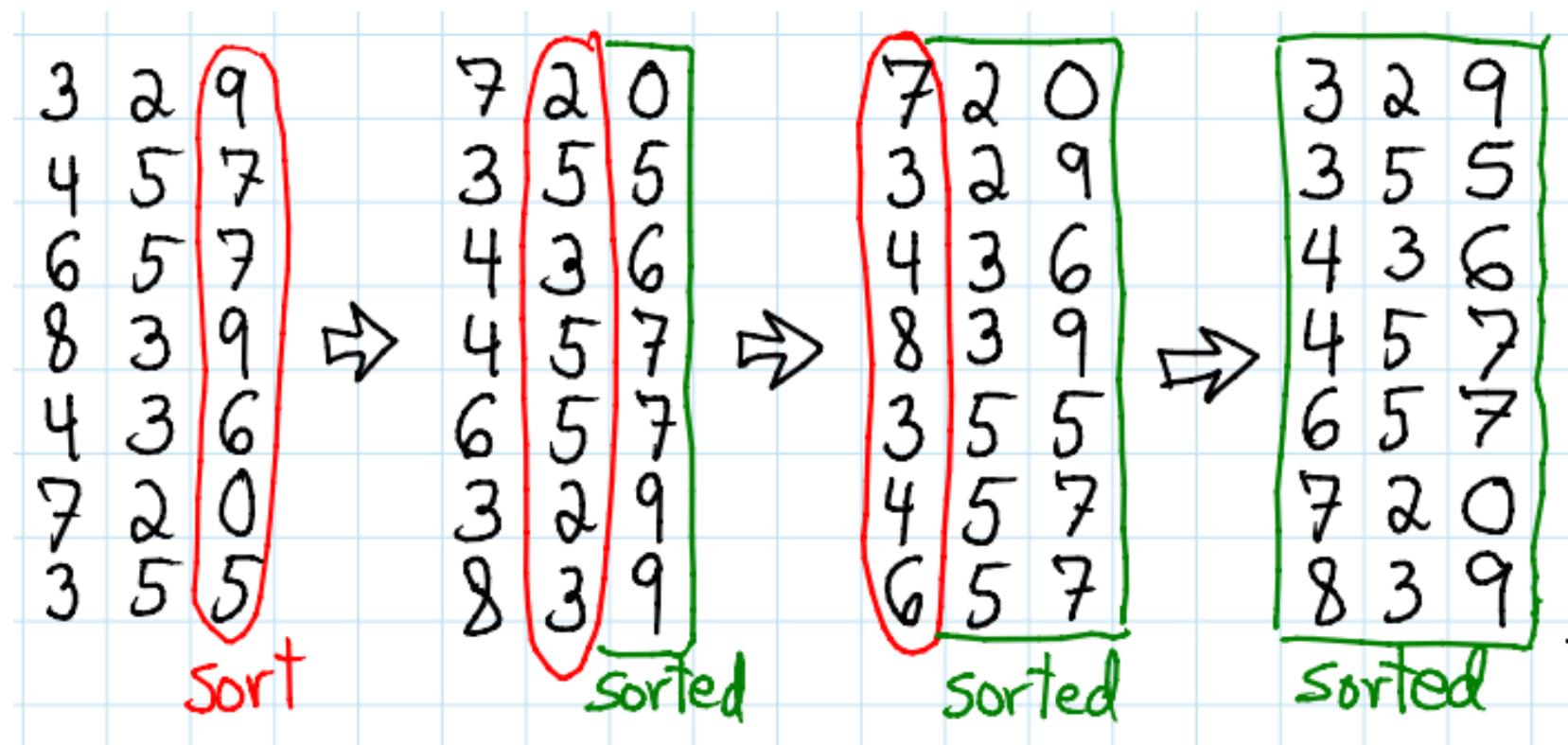
<https://brilliant.org/wiki/radix-sort/>

3

(1 Point)

What is the content of the following sequence of elements [329, 457, 657, 839, 436, 720, 355] after two iterations of the LSD radix sort algorithm using digits as individual keys?

- [720, 329, 436, 839, 355, 457, 657] ✓
- [720, 355, 436, 457, 657, 329, 839]
- [329, 355, 457, 436, 657, 720, 839]
- [329, 355, 436, 457, 657, 720, 839]



<https://brilliant.org/wiki/radix-sort/>

4

(1 Point)

How many non-empty buckets are formed during the first iteration of the MSD radix sort algorithm when applied to the sequence [329, 457, 657, 839, 436, 720, 355] using decimal digits as individual keys?

4

5

6

7

4

(1 Point)

How many non-empty buckets are formed during the first iteration of the MSD radix sort algorithm when applied to the sequence [329, 457, 657, 839, 436, 720, 355] using decimal digits as individual keys?

 4 5 6 7

0	1	2
3	2	9
4	5	7
6	5	7
8	3	9
4	3	6
7	2	0
3	5	5

sort

4

(1 Point)

How many non-empty buckets are formed during the first iteration of the MSD radix sort algorithm when applied to the sequence [329, 457, 657, 839, 436, 720, 355] using decimal digits as individual keys?

- 4
- 5 ✓
- 6
- 7

0	1	2
3	2	9
4	5	7
6	5	7
8	3	9
4	3	6
7	2	0
3	5	5

sort

0	1	2
3	2	9
3	5	5
4	5	7
4	3	6
6	5	7
7	2	0
8	3	9

sorted sort

0	1	2
3	2	9
3	5	5
4	3	6
4	5	7
6	5	7
7	2	0
8	3	9

sorted

0	1	2
3	2	9
3	5	5
4	3	6
4	5	7
6	5	7
7	2	0
8	3	9

sorted

**Which of the following statements is CORRECT?**

- A) Radix Sort is always faster than QuickSort.
- B) The choice of base (radix) has no effect on the space complexity of Radix Sort.
- C) Bucket Sort is not a stable sorting algorithm.
- D) In practice MSD Radix Sort can be faster than LSD Radix Sort for sorting variable length strings.

## Which of the following statements about Radix Sort is CORRECT?

- A) Radix Sort is always faster than QuickSort. (**Incorrect** : Radix sort with runtime  $O(d(n + N))$  is faster than  $O(n \log n)$  time algorithms if  $d$  (number of elementary keys per composite key) and  $N$  (radix or number of possible values per elementary key) are reasonably small such that  $O(d(n + N))$  is  $O(n)$ . For instance, if  $d = n$  and  $N \leq n$ , then radix sort is  $O(n^2)$ .)
- B) A very large base (radix) can lead to excessive memory usage for Radix Sort.  
(**Incorrect**: space complexity of Radix Sort is  $O(n+N)$ .)
- C) Bucket Sort is not a stable sorting algorithm. (**Incorrect**)
- D) In practice MSD Radix Sort can be faster than LSD Radix Sort for sorting variable length strings. (**Correct**: LSD Radix Sort processes all characters whereas a sorting order with MSD can be achieved without processing all keys. LSD requires additional processing to handle variable length strings)

# Selection or Sorting ?

Consider that we would like to find and print duplicate elements in an unsorted sequence of integers. For example, given the sequence (3, 2, 4, 3), we would like to print the number 3.

**Which of the following data structures or algorithms would lead to the most time-efficient solution for this task, considering tightest average or expected time complexity in big-Oh notation?**

- A. Quick sort
- B. Quick select.
- C. Nested loops to make comparisons between all pairs of elements.
- D. List-based priority queue.

Consider that we would like to find and print duplicate elements in an unsorted sequence of integers. For example, given the sequence (3, 2, 4, 3), we would like to print the number 3.

**Sorting (quick sort):** first sort the sequence using quick sort in expected  $O(n \log n)$  time, then traverse it in  $O(n)$  time to find duplicates. This leads to an expected  $O(n \log n)$  time solution.

**Selection (quick select):** repeatedly call quick select to get items based on their rank/order (e.g. retrieve the smallest, the 2nd smallest, and so on) and check for duplicates in the same way as when items are sorted. The expected runtime of quick select is  $O(n)$  per item. Since we run quick select for every rank (as many as the number of elements  $n$ ), we end up with an expected  $O(n^2)$  time solution.

**Comparisons:** making all pairwise comparisons leads to comparing each of the  $n$  elements against the other  $n - 1$  other elements. This solution has average runtime  $O(n^2)$ .

**List-based PQ:** first insert all items, then remove them until the PQ is empty, while checking for duplicates (this is similar to using sorting, since the PQ returns the items in sorted order). Doing this using list-based PQ takes  $O(n^2)$  time, since an unsorted list PQ takes  $O(k)$  per removal and a sorted list PQ takes  $O(k)$  per insertion (where  $k$  is the size of the PQ when removing/inserting).

# Selection or Sorting ?

Which algorithm has the best expected time complexity to find the median element of an unordered sequence S with n elements, assuming that n is odd?

- A. Binary search.
- B. Quick select.
- C. Quick sort followed by retrieval of the  $[n/2]$ th element.
- D. Build a min-heap and extract  $[n/2]$  elements. The median is the  $[n/2]$ th element.

# Selection or Sorting ?

Which algorithm has the best expected time complexity to find the median element of an unordered sequence S with n elements, assuming that n is odd?

- A. Binary search algorithm is **not** applicable! The binary search algorithm searches for a given element in a sorted sequence. We don't know the median, so we can't search for it.
- B. Quick select works and has expected time complexity  $O(n)$ .
- C. Quick sort and retrieval of middle element works with expected time complexity  $O(n \log n)$  for sorting and  $O(1)$  for retrieval. Therefore  $O(n \log n)$  for the complete solution.
- D. Building min-heap takes  $O(n)$  time using heapify. Extracting approx. half of the elements takes  $O(n \log n)$  time, since we call remove  $\lceil n/2 \rceil$  times and each one takes  $O(\log k)$  time (proportional to the height of the heap with  $k$  elements, due to bubbling), with  $k$  between  $\lceil n/2 \rceil$  and  $n$ .

Resit exam 18-19 - MCQ 9  
(1 Point)

9. (1 point) Which of the following statements on sorting algorithms is **false**?

- A. An algorithm is called stable if it maintains the relative order of identical elements (or elements with an identical key).
- B. Four sorting algorithms ordered by **best-case** time complexity: insertion sort  $\leq$  quicksort  $\leq$  heap sort  $\leq$  selection sort.
- C. The minimum possible time complexity of a comparison-based sorting algorithm is  $\Omega(n \log n)$  for an input sequence with  $n$  elements.
- D. Quicksort is the fastest algorithm for sorting 1 million license plate numbers.

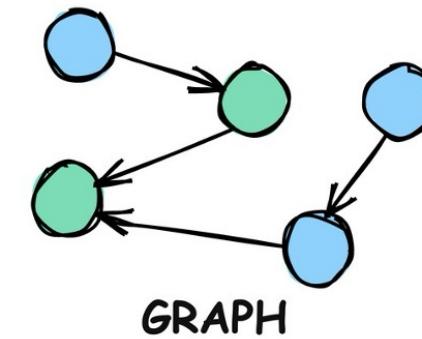
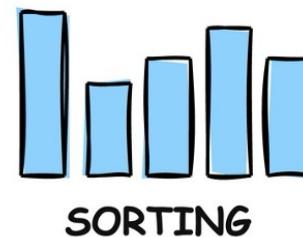
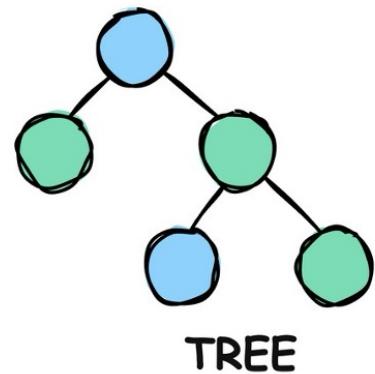
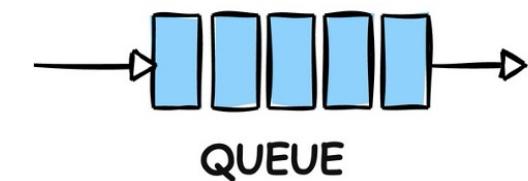
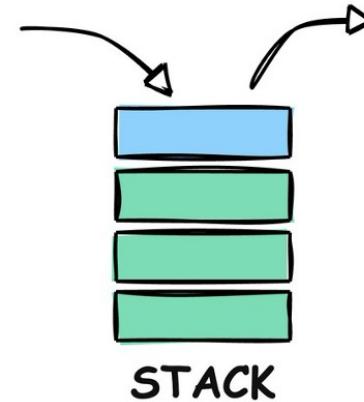
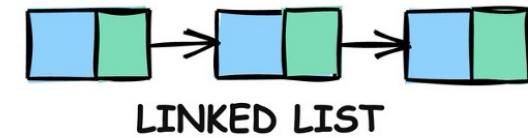
## Resit exam 18-19 - MCQ 9

(1 Point)

9. (1 point) Which of the following statements on sorting algorithms is **false**?

- A. An algorithm is called stable if it maintains the relative order of identical elements (or elements with an identical key).
- B. Four sorting algorithms ordered by **best-case** time complexity: insertion sort  $\leq$  quicksort  $\leq$  heap sort  $\leq$  selection sort.
- C. The minimum possible time complexity of a comparison-based sorting algorithm is  $\Omega(n \log n)$  for an input sequence with  $n$  elements.
- D. Quicksort is the fastest algorithm for sorting 1 million license plate numbers.

- A. **True.** This is the definition of stability of sorting.
- B. **True.** Best case: insertion sort is  $O(n)$ , then quick sort is  $O(n \log n)$  but is faster than heap/merge sorts in best and average cases, then heap sort is  $O(n \log n)$  but slower than quick sort, then selection sort is  $O(n^2)$ . Why is quick sort usually faster than heap/merge?
  - Avoids unnecessary swaps: it doesn't necessarily do less comparisons, but it does much fewer swaps (swaps are costly) – it avoids a lot of swaps/moves for what is already ordered, while heap/merge do not.
  - Cache-efficient: it's in-place, all the work happens in this one array, so it can make use of locality to access contiguous positions that have been cached as a block (merge sort is not in-place, there's overhead to create auxiliary data structures, copying elements back and forth, and accessing RAM).
- C. **True.** We have done this proof in the lecture videos. Note that here we are talking about what we can achieve if we use the **best algorithm in the worst case**, i.e., the **worst case time complexity of the best algorithm**. For instance, insertion sort might require  $O(n)$  in the best case but it requires  $O(n^2)$  for the worst case.
- D. **False.** Radix sort is faster, since these numbers are of small length ( $< 10$ ) and composed of letters (26-size alphabet possible) and digits (10-size alphabet), each of which can be used individually as a key for bucket sort. In this case, radix sort has linear time complexity and is therefore faster than comparison-based algorithms.

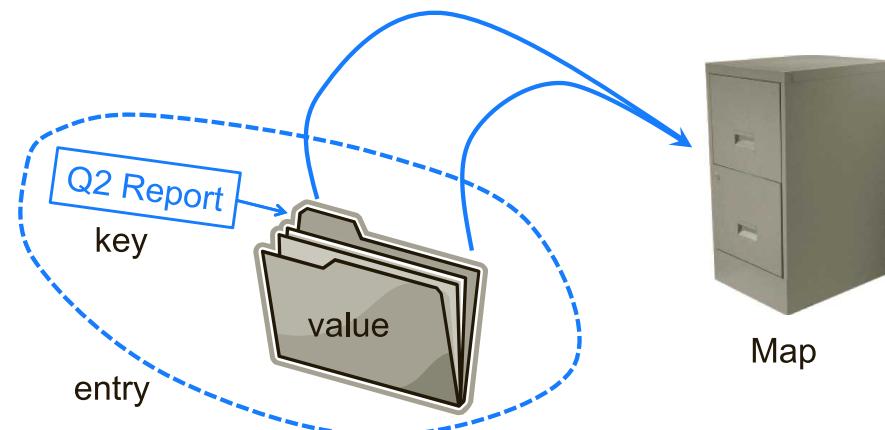


# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Map ADT

```
1 public interface Map<K,V> {  
2     int size();  
3     boolean isEmpty();  
4     V get(K key);  
5     V put(K key, V value);  
6     V remove(K key);  
7     /* more operations in the book */  
8 }
```



# Implementing Maps using **unordered** ArrayList

**Example** (strings as keys, integers as values):

(a, 10)	(f, 9)	(y, 1)	(c, 3)	(de, 6)	(dd, 3)	(i, 20)	(d, 1)	(z, 17)
---------	--------	--------	--------	---------	---------	---------	--------	---------

$\mathcal{O}(n)$

**V get ( K key )** : traverse the ArrayList to find the key and return the corresponding value

**V put ( K key )** : traverse the ArrayList to find the key

$\mathcal{O}(n)$

- if such a key exists update the value
- Otherwise add a new entry at the end

**V remove ( K key )** : traverse the ArrayList to find the key

$\mathcal{O}(n)$

- if such a key exists remove the corresponding entry
- Otherwise we are done

# Implementing Maps using **ordered** ArrayList

**The entries are stored in a sorted order based on keys**

**Example** (strings as keys, integers as values):

(a, 10)	(c, 3)	(d, 1)	(dd, 3)	(de, 6)	(f, 9)	(i, 20)	(y, 1)	(z, 17)
---------	--------	--------	---------	---------	--------	---------	--------	---------

Which of the three operations **get**, **put** and **remove**  
now can be done more efficiently?

# Implementing Maps using **ordered** ArrayList

Represent maps as **ordered arraylists** of **key/value pairs** (called “entries”)

- ▶ `V get(K key)`: find the matching entry using **binary search**  $\mathcal{O}(\log_2 n)$
- ▶ `V put(K key, V value)`: find the matching entry using **binary search**  $\mathcal{O}(n)$ 
  - ▶ If such an entry exists, update the value
  - ▶ Otherwise, add a new entry at the right place, and shift everything
- ▶ `V remove(K key)`: find the matching entry using **binary search**  $\mathcal{O}(n)$ 
  - ▶ If such an entry exists, remove it, and shift everything
  - ▶ Otherwise, we are done already

**Example** (strings as keys, integers as values):

(a, 10)	(c, 3)	(d, 1)	(dd, 3)	(de, 6)	(f, 9)	(i, 20)	(y, 1)	(z, 17)
---------	--------	--------	---------	---------	--------	---------	--------	---------

# Efficient access

Which data structure to use to access an item efficiently by an integer key?

The value stored with key  $k$  is stored at index  $k$  of the array:

0	1	2	3	4	5	6	7	8	9	10
	d		z			c	q			

What if the **key** is not an integer ?

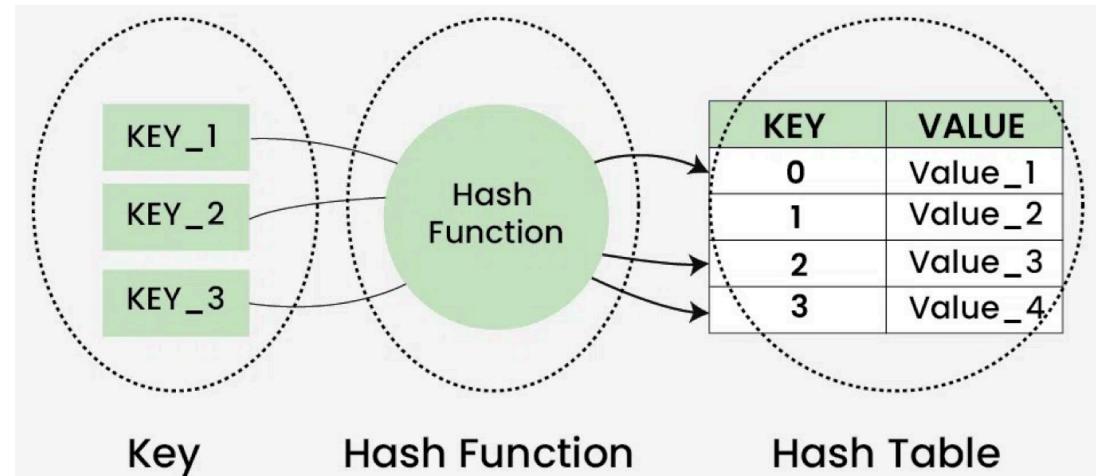
What if the number of keys are much smaller than the largest possible key value ?

# Hash Table

A **hash function**  $h$  maps keys of some type to integers in a fixed interval  $[0, N - 1]$

A **hash table** puts entries with key  $k$  at index  $h(k)$  of an array of size  $N$

The elements of a hash table, called the **buckets**, may contain multiple entries



# Load Balancing

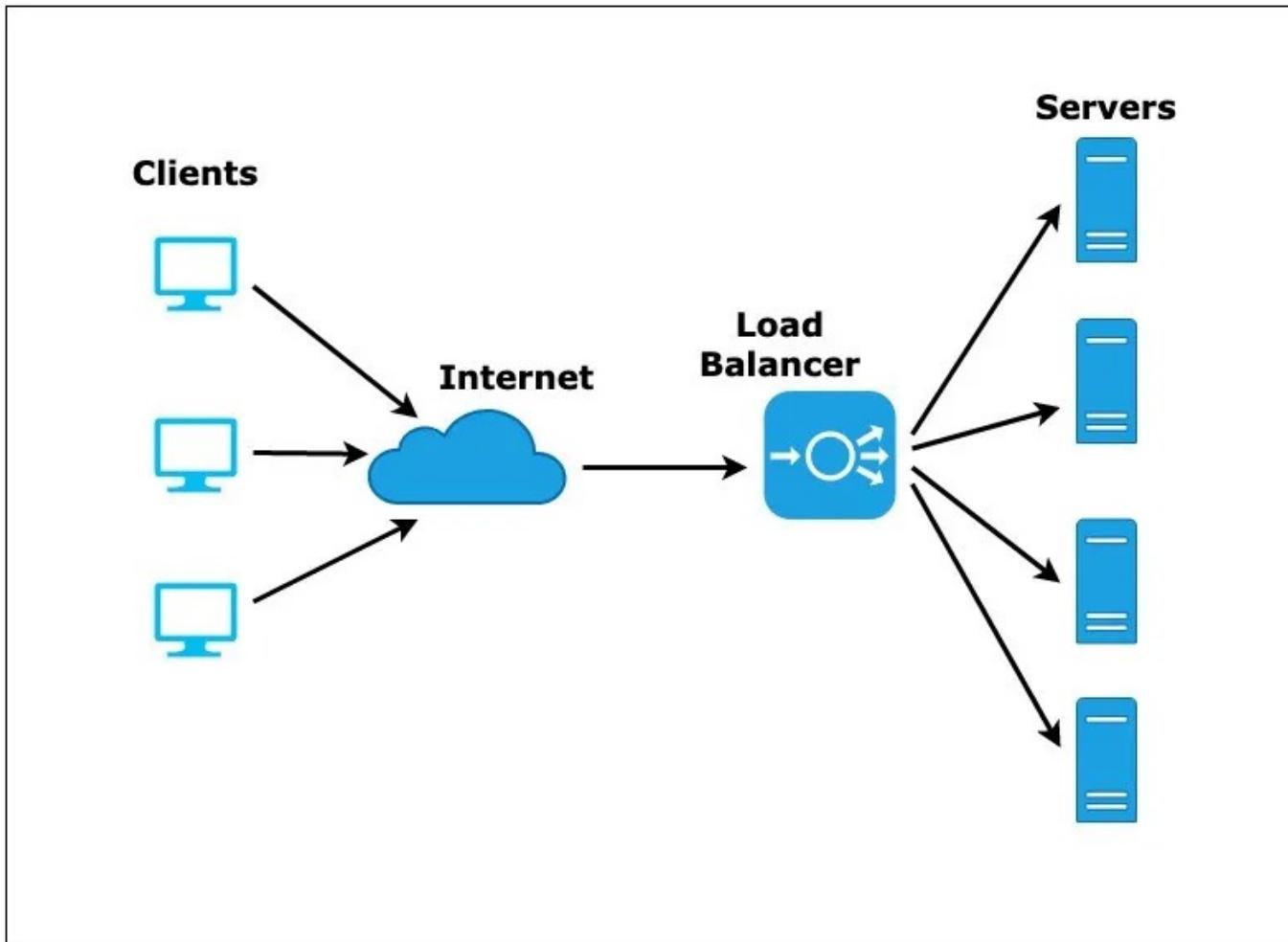


Image source : <https://medium.com/@avinashsoni9829/design-concepts-5586976174f4>

Request Ids

0

1

2

$m-1$

$m$

Assignment

$(0, n)$

$(1, 3)$

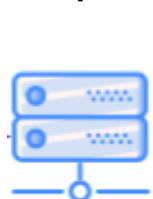
Servers



Server 1



Server 2

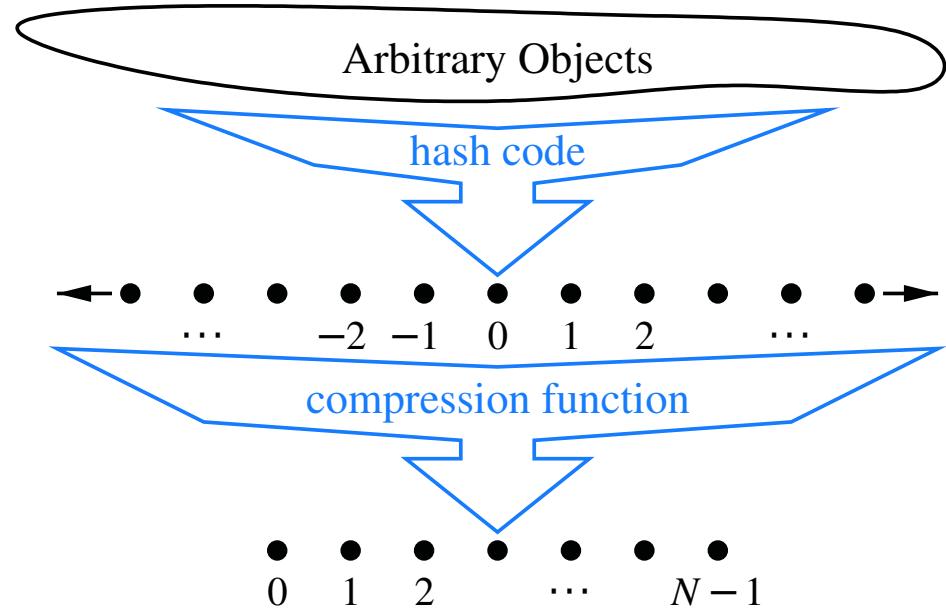


Server  $n$

# Hash functions

Built from two key components

- hashCode : Key  $\rightarrow$  int
- compress : int  $\rightarrow$  [0, N-1]



What is the advantage of this separation?



What can go wrong without this separation ?

# Hash Code

What is true about hash codes?

- (A) If `x.equals(y)`, then `x.hashCode() == y.hashCode()`
- (B) If `x.hashCode() == y.hashCode()` then `x` is equal to `y`
- (C) Both A & B are true
- (D) Neither A or B is true

# Hash Code

You are developing a **dictionary application** that stores **words as keys in a hash table**.

You are considering two different techniques for generating hash codes:

- (i) simple addition of integers corresponding to ASCII codes of characters in the string
- (ii) using polynomial hash codes over the ASCII integer codes of characters in the string

Which of the techniques should you prefer and why?

# Compression Function



Is the result of `hashCode` directly usable for implementing a hashtable?

No, a hashtable has a fixed size  $N$ , whereas `hashCode` yields an `int`

Two common compression methods:

- ▶ The **division method**

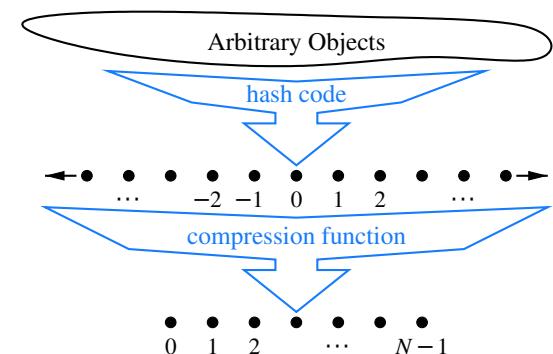
$$\text{compress}(i) = i \mod N$$

Works best when  $N$  is a prime

- ▶ The **Multiply-Add-and-Divide (MAD)** method:

$$\text{compress}(i) = ((ai + b) \mod p) \mod N$$

Where  $p \geq N$  is a prime number, and  $a$  and  $b$  are integers between 0 and  $p - 1$  with  $a > 0$

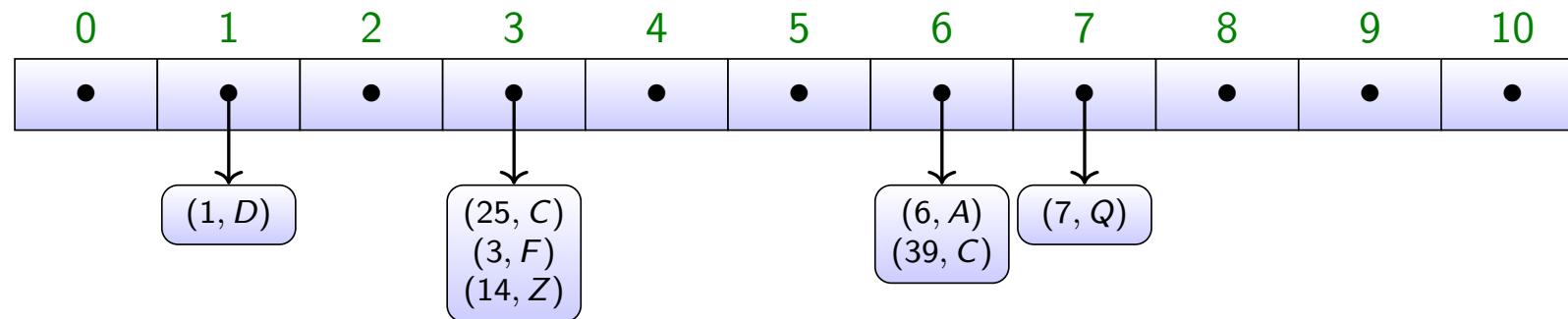


# Compression Functions

If we have a hash table of size  $N=100$  can you think of a situation where the Division Method would not be an ideal compression function?

# Collision

$$h(i) = i \bmod 11$$

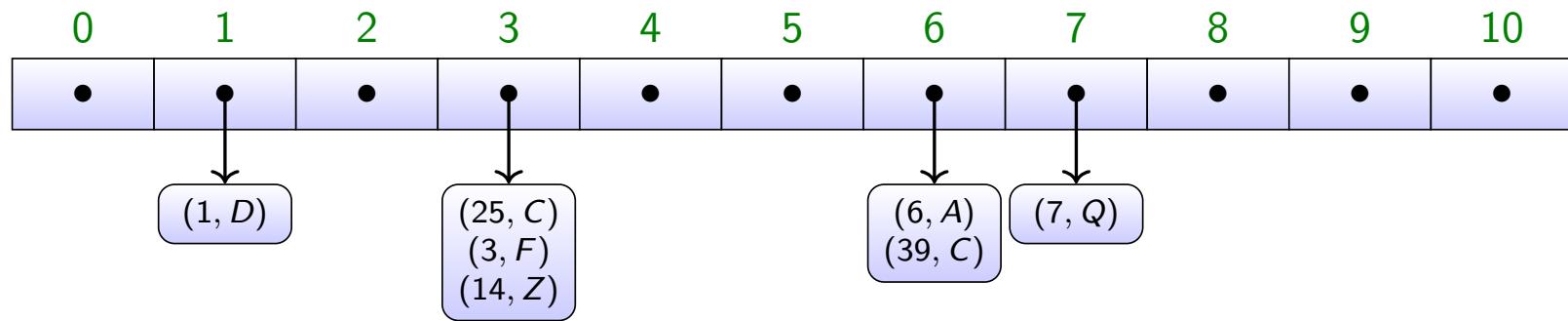


A **collision** occurs when we have keys  $k_1, k_2$  such that

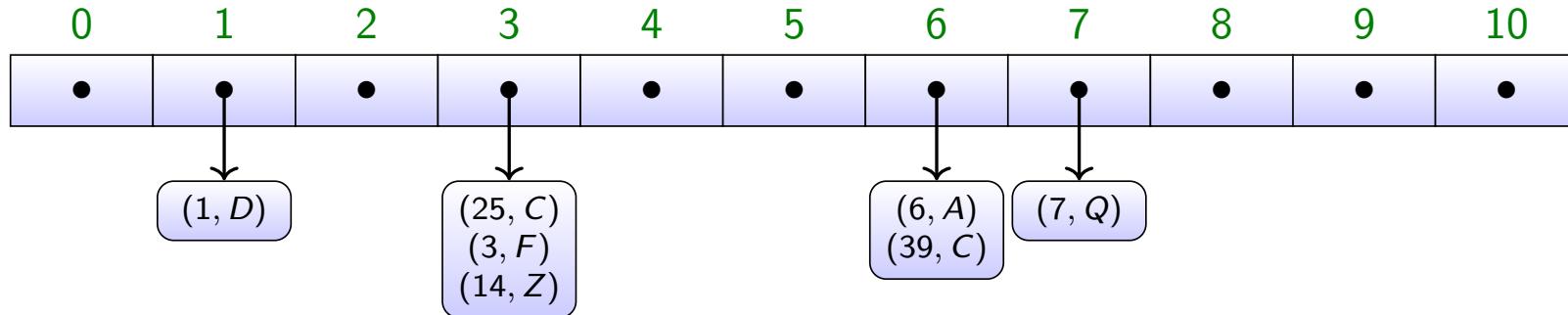
$$k_1 \neq k_2 \text{ but } h(k_1) = h(k_2)$$

# Collision Resolution: Separate Chaining

**Employ a secondary container/map in each bucket**



# Separate Chaining

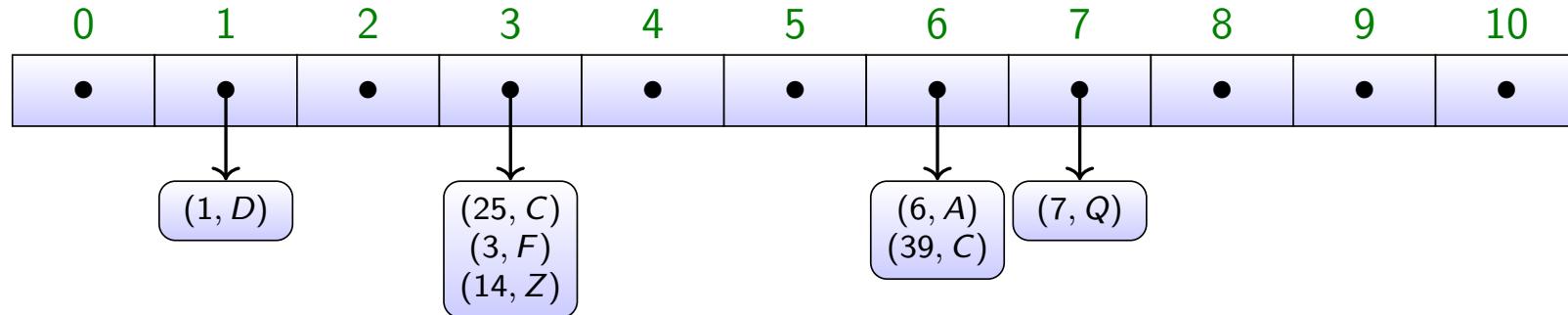


**Which data type would you use to implement the secondary container?**



Number of entries in a bucket is usually small but varies

# Separate Chaining



**Which data type would you use to implement  
the secondary container?**

An unordered list, the numbers of entries in a bucket is small and varies (so, an array is not a good choice). Empty buckets can be represented by null pointers.

# Separate Chaining

Given an **array-based map** that uses **hashing with separate chaining** as its collision-handling scheme, what should the size of the **hash table  $N$  be relative to the number of stored keys  $n$** , in order to ensure that the core operations (insertion, deletion, and search) run in  $O(1)$  average-case time complexity?

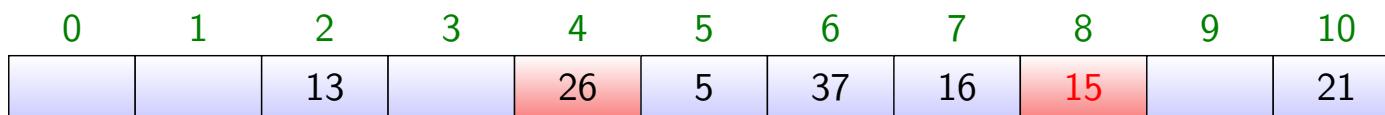
- A)  $N$  should be a constant value independent of  $n$  as long as a good hash function is used
- B.  $N$  is  $\Theta(n)$  ensuring a constant load factor.
- C.  $N < n$  to optimize space efficiency while keeping operations efficient.
- D.  $N$  is  $O(n^2)$  to completely avoid collisions

# Open Addressing

**Use other buckets in case of a collision**

## Linear Probing

- When inserting  $(k, v)$  and  $A[h(k)]$  is already occupied try  $A[(h(k)+1) \bmod N]$
- If that one is occupied too, try  $A[(h(k) + 2) \bmod N], \dots$



# Linear Probing

Consider the following fixed size hash table using linear probing (associated values are omitted):

0	1	2	3	4	5	6	7	8	9
9		11					36	16	18

The hash function is  $h(k) = (k + 1) \bmod 10$ . Assume we first insert an entry with key 27, then delete the entry with key 16, and then insert an entry with key 6. In which bucket will the inserted entry appear ?

# Linear Probing

Which core function(s) of a map implementation using Linear Probing as a collision resolution scheme do(es) not work anymore if we do not replace a deleted entry with a "defunct" object?

# Comparing collision resolution schemes

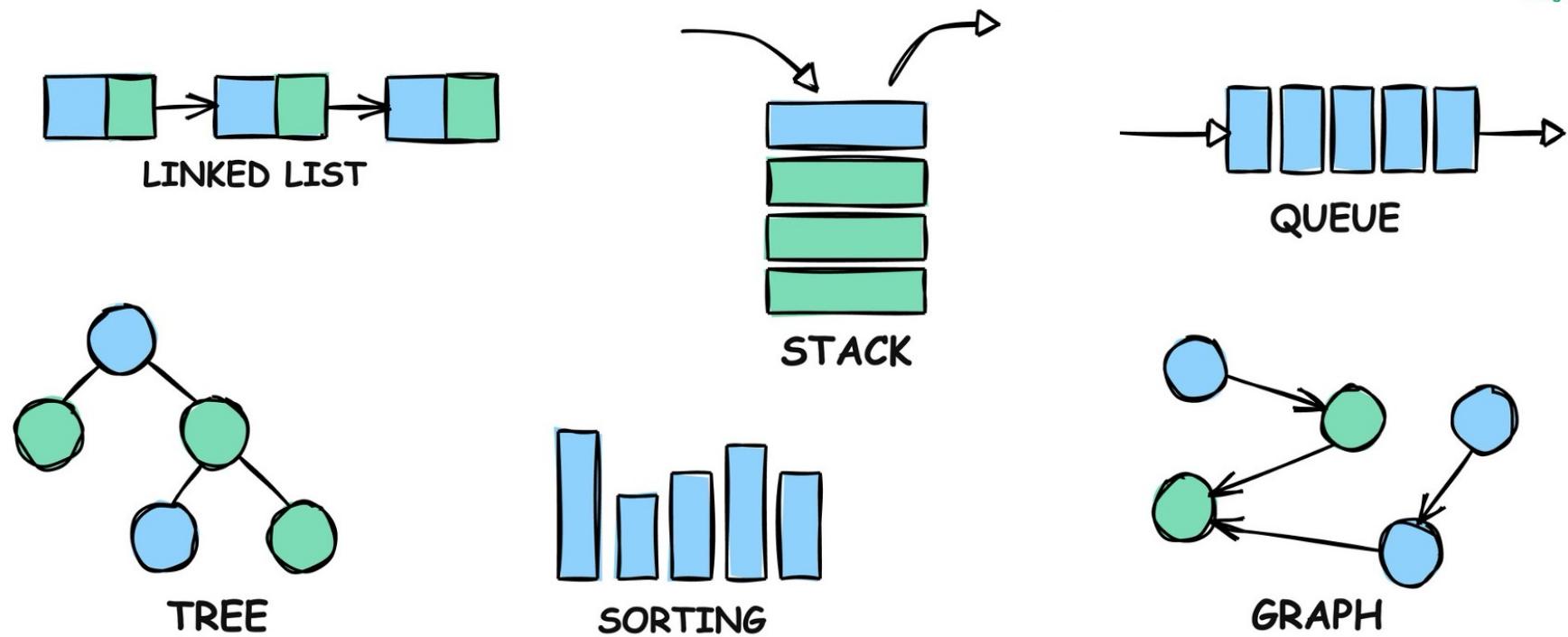
**Which of the following statements correctly describe the differences between linear probing and separate chaining collision resolution schemes?**

- A) Linear probing is susceptible to primary clustering, where consecutive occupied slots increase the likelihood of future collisions, whereas separate chaining distributes collisions across secondary containers, preventing clustering issues.
- B) Separate chaining may require more memory than linear probing because it uses secondary container for each bucket for collision handling.
- C) Linear probing performs better than separate chaining when the load factor is high because it avoids additional memory allocation.
- D) Separate chaining has an average case insertion time complexity of  $O(1)$ , whereas linear probing has an average case insertion time of  $O(n)$ .



Say that for some reason, we want to implement a hashmap where we rehash once our load factor  $\lambda$  reaches 1.1. Which collision-handling scheme should we choose?

- (a) Separate Chaining
- (b) Linear probing
- (c) It doesn't matter

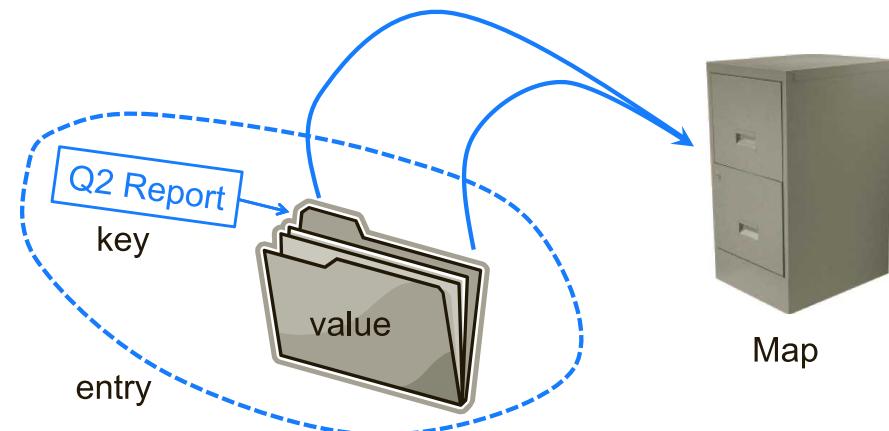


# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

# Sorted Maps

```
1 public interface Map<K,V> {  
2     int size();  
3     boolean isEmpty();  
4     V get(K key);  
5     V put(K key, V value);  
6     V remove(K key);  
7     /* more operations in the book */  
8 }
```



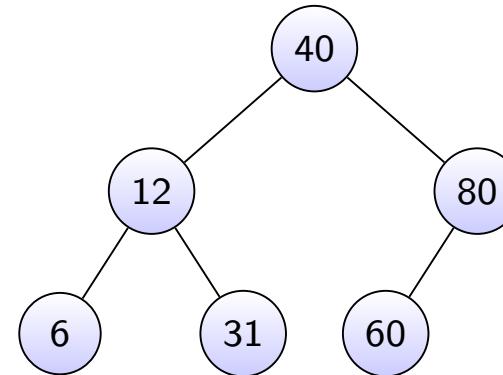
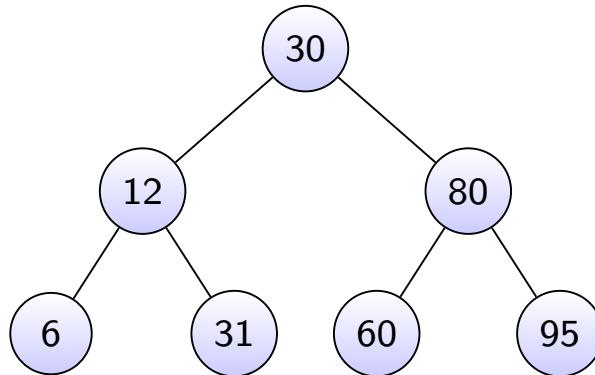
**Goal:** An efficient implementation of **sorted map** with time complexity of  $O(\log n)$  for get, put and remove operations

# Binary Search Trees



Are the following trees binary search trees?

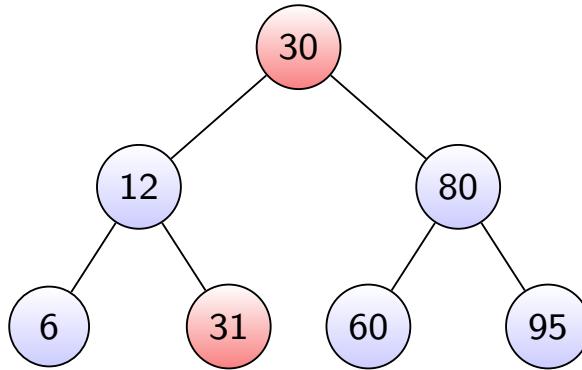
- (a) Yes, both of them are binary search trees
- (b) The left tree is a binary search tree, the right one isn't
- (c) The right tree is a binary search tree, the left one isn't
- (d) No, both of them are not binary search trees



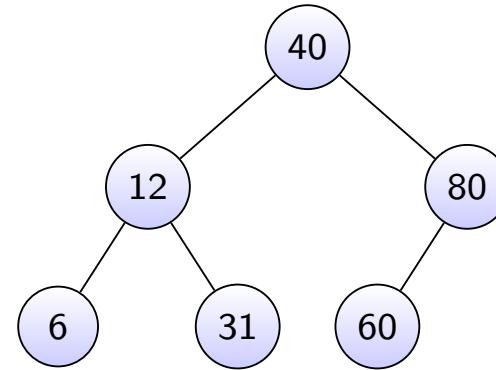
# Binary Search Trees



Are the following trees binary search trees?



No, 31 is not smaller than 30

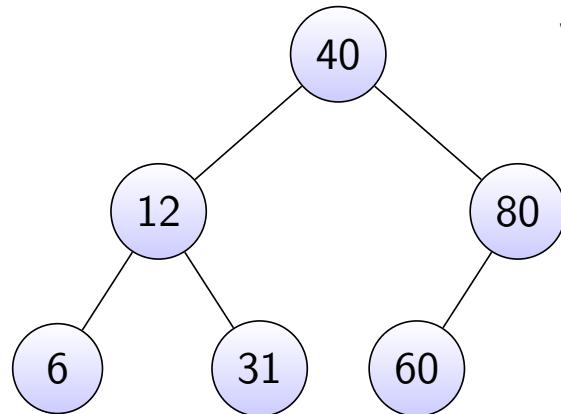


Yes

# Insertion in a BST

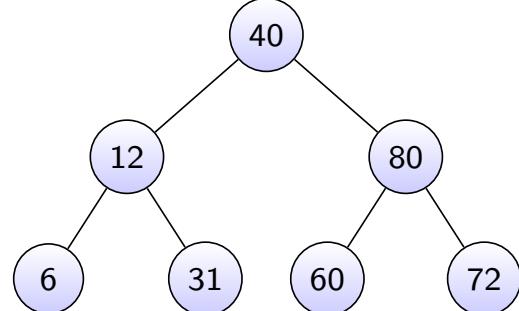


If we add 72 to the tree below, what tree do we get?

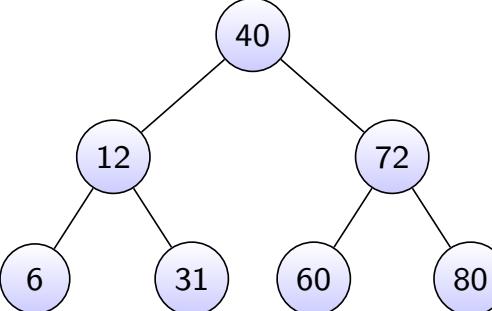


We are only interested in insertion without any restructurings.

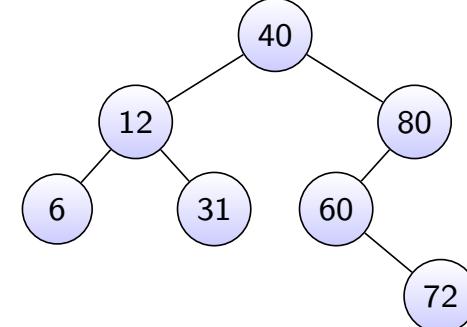
A



B



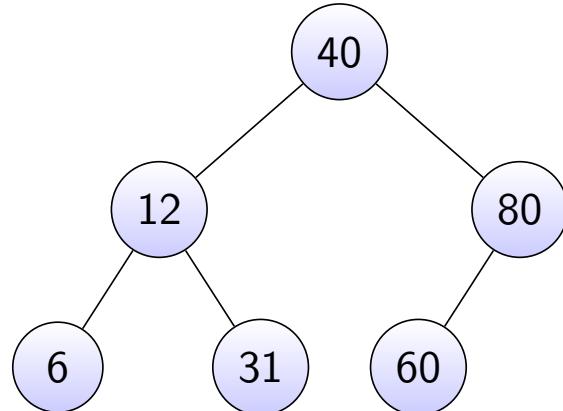
C



# Insertion in a BST

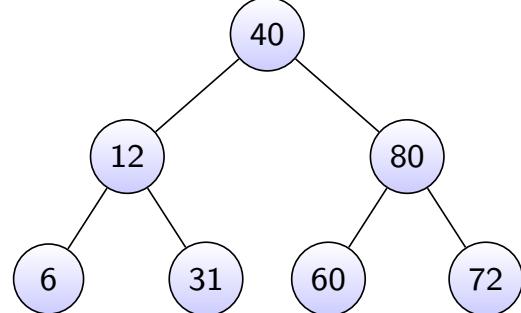


If we add 72 to the tree below, what tree do we get?

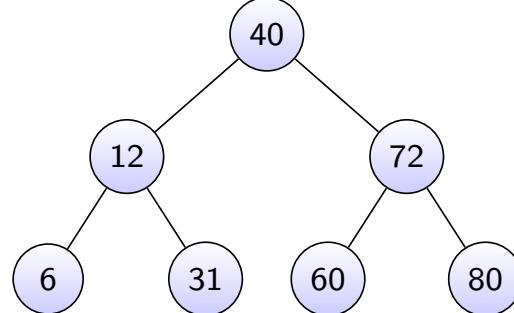


We are only interested in insertion without any restructurings.

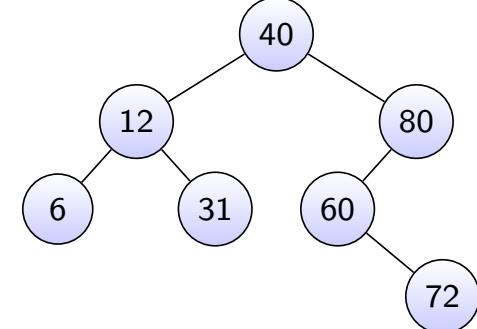
A



B

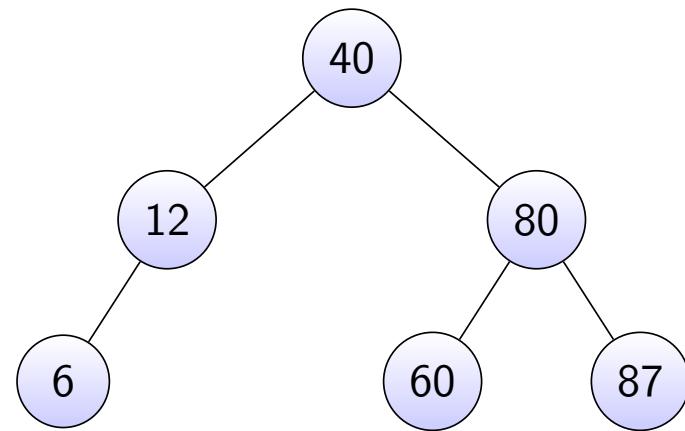


C

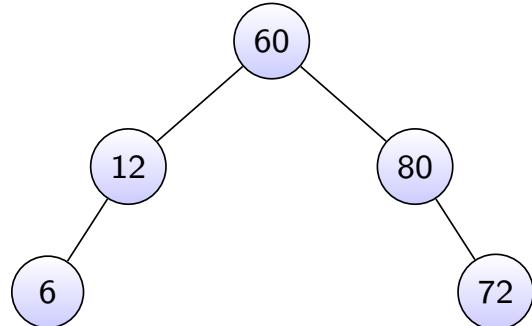


# Deletion

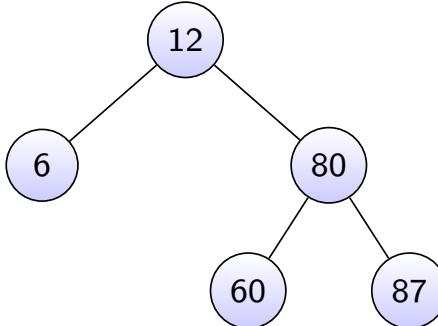
If we remove 40 from the tree below which BST tree (without any restructurings) do we obtain?



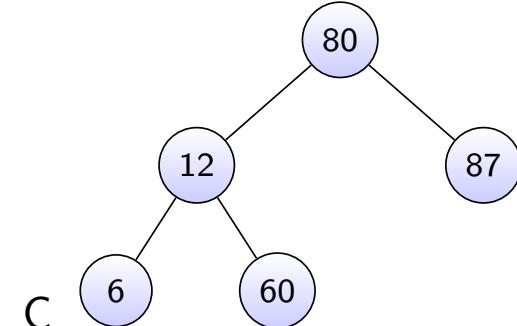
A



B

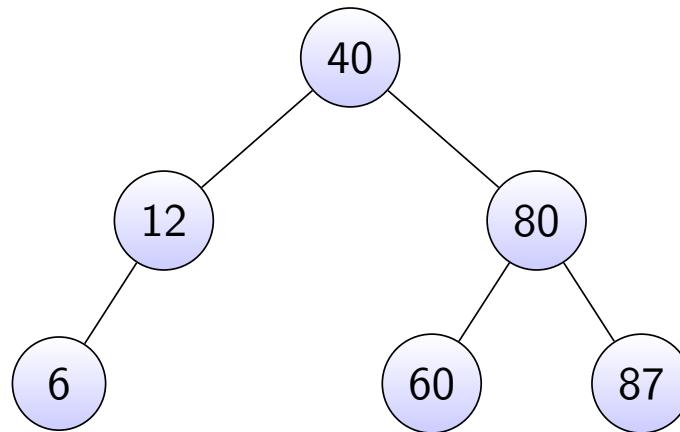


C

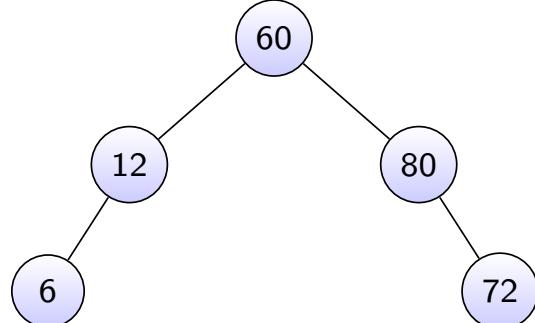


# Deletion

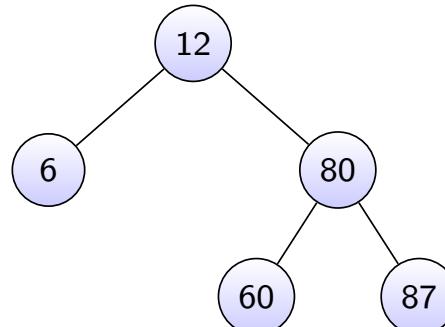
If we remove 40 from the tree below which BST tree (without any restructurings) do we obtain?



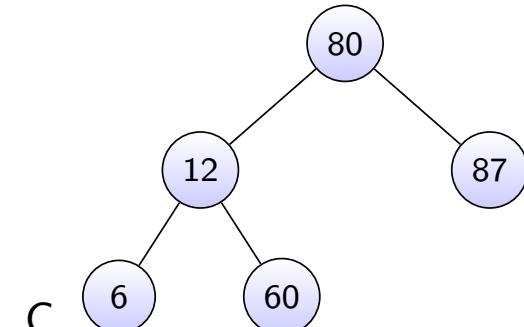
A



B



C



# Time complexity for BST

What is the time complexity of the search operation in the BST?

# Time complexity for BST

What is the time complexity of the search operation in the BST?  $\mathcal{O}(h)$

```
1 public V get(Position<Entry<K,V>> p, K key) {  
2     if (p == null)      return null;  
3  
4     int comp = compare(key, p.getElement());  
5     if (comp == 0)      return p.getElement();  
6     else if (comp < 0)  return get(left(p), key);  
7     else                 return get(right(p), key);  
8 }
```

Recurrence equations (where  $h$  is the height of the tree):

$$T(0) = c_1 \quad T(h) = T(h - 1) + c_2$$

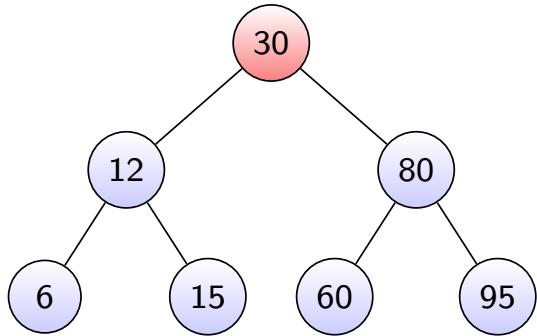
Closed form:

$$T(h) = h \cdot c_2 + c_1$$

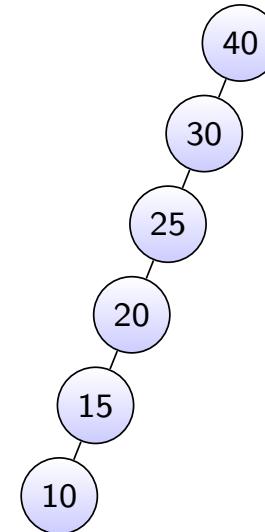
The time-complexity of binary search is  $\mathcal{O}(h)$

# Best and worst case for BST

**Best case:** height is  $\mathcal{O}(\log n)$



**Worst case:** height is  $\mathcal{O}(n)$



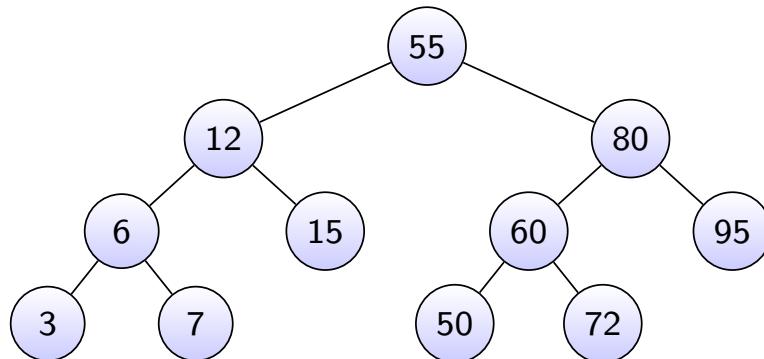
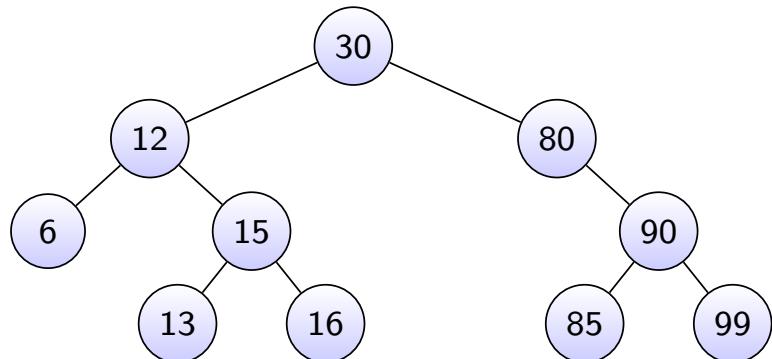
Insert a sequence whose keys is sorted (or inversely sorted), for example: 40, 30, 25, 20, 15, 10

# AVL Tree



Are the following trees AVL trees?

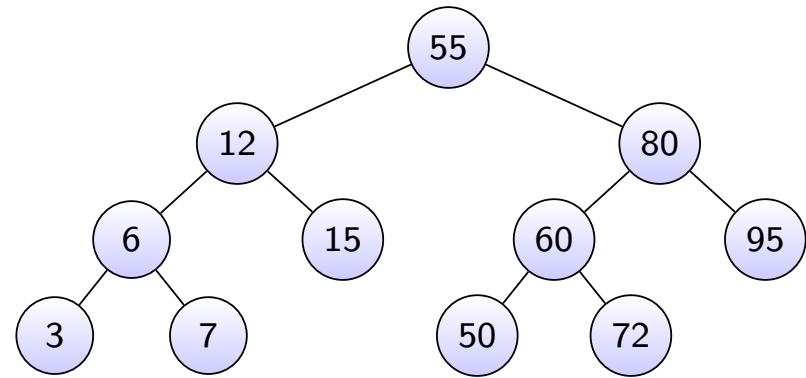
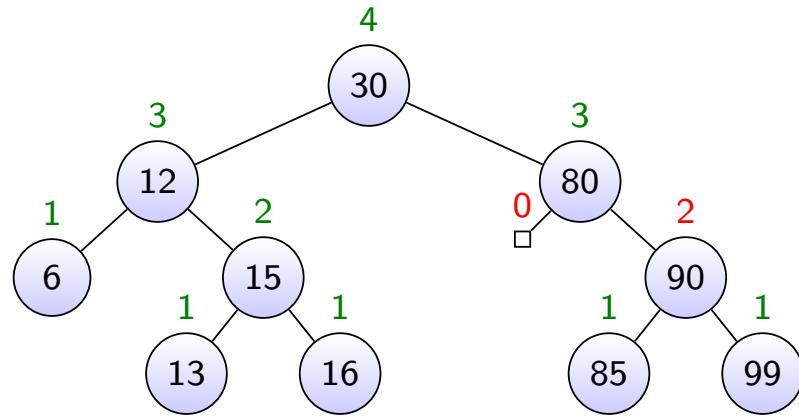
- (a) Yes, both of them are AVL trees
- (b) The left tree is an AVL tree, the right one isn't
- (c) The right tree is an AVL tree, the left one isn't
- (d) No, both of them are not AVL trees



# AVL Tree



Are the following trees AVL trees?

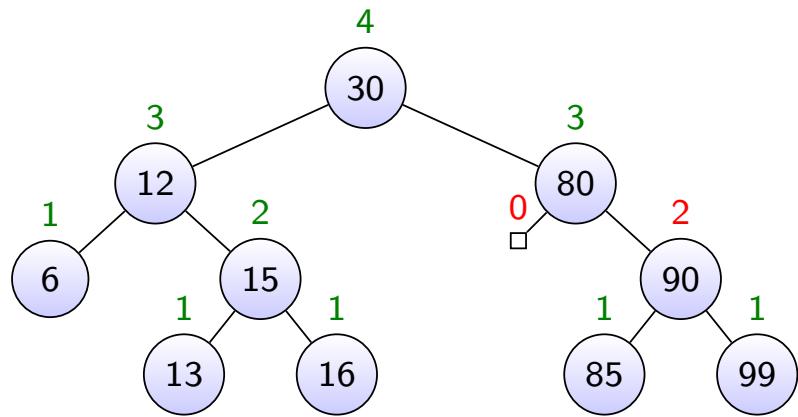


No, be aware of the **nulls**

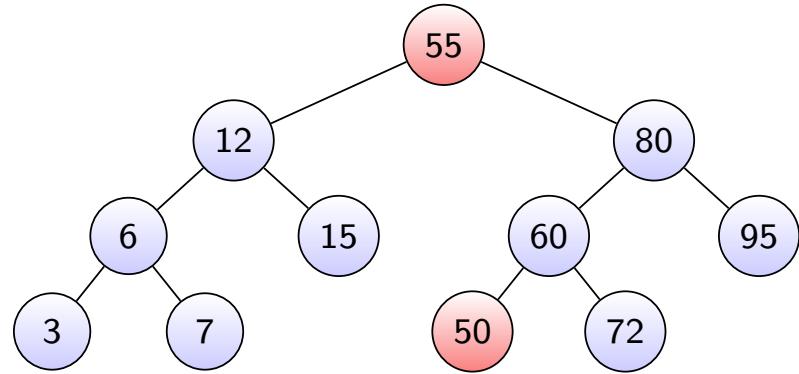
# AVL Tree



Are the following trees AVL trees?



No, be aware of the **nulls**

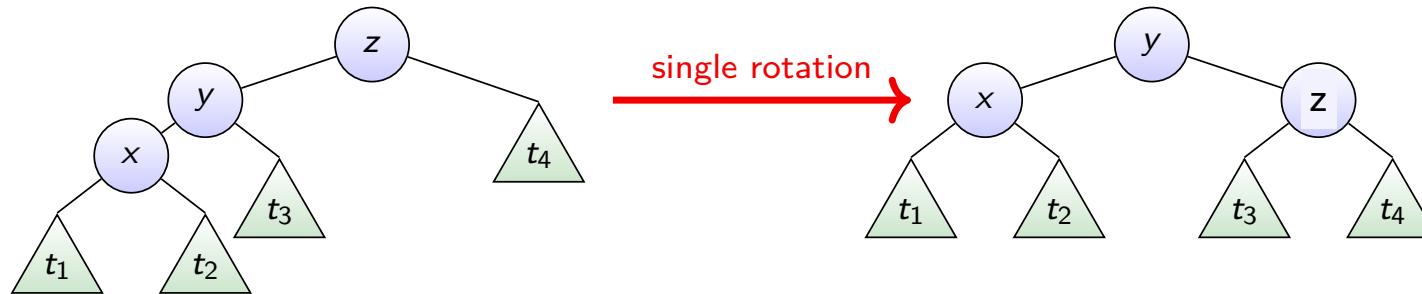


No, AVL trees should be search trees

# Single and Double Rotations

Insert [1,2,3,5,4,6] in that order into an empty AVL Tree

- For which insertions do we need a single rotation?

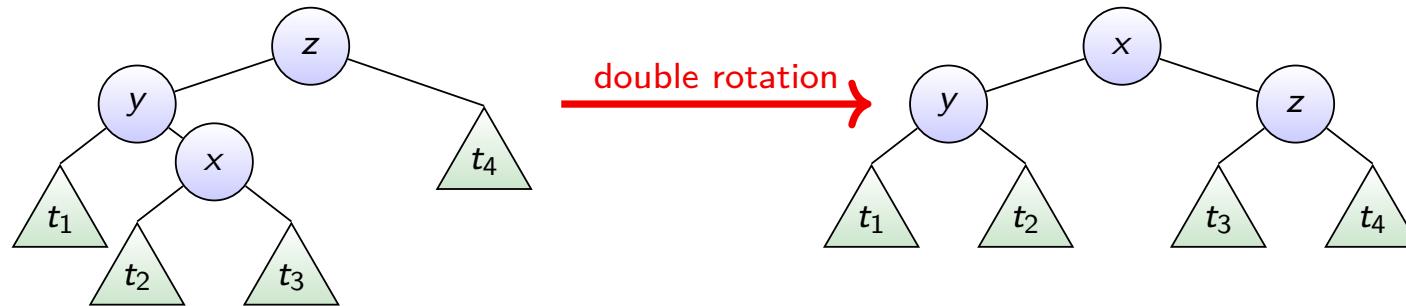


- For a new insertion at position  $p$ , let  $\textcolor{red}{z}$  be the first unbalanced position encountered going up from  $p$  towards the root.
- Let  $\textcolor{green}{y}$  be the child of  $z$  with greater height. Let  $\textcolor{blue}{x}$  be the child of  $y$  with greater height.

# Single and Double Rotations

Insert [1,2,3,5,4,6] in that order into an empty AVL Tree

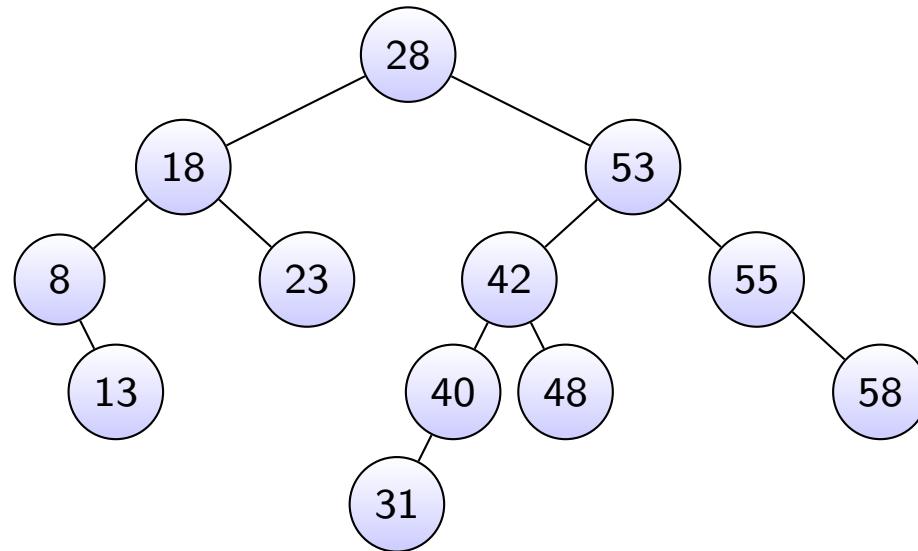
- For which insertions do we need a double rotation?



- For a new insertion at position **p**, let **z** be the first unbalanced position encountered going up from p towards the root.
- Let **y** be the child of z with greater height. Let **x** be the child of y with greater height.

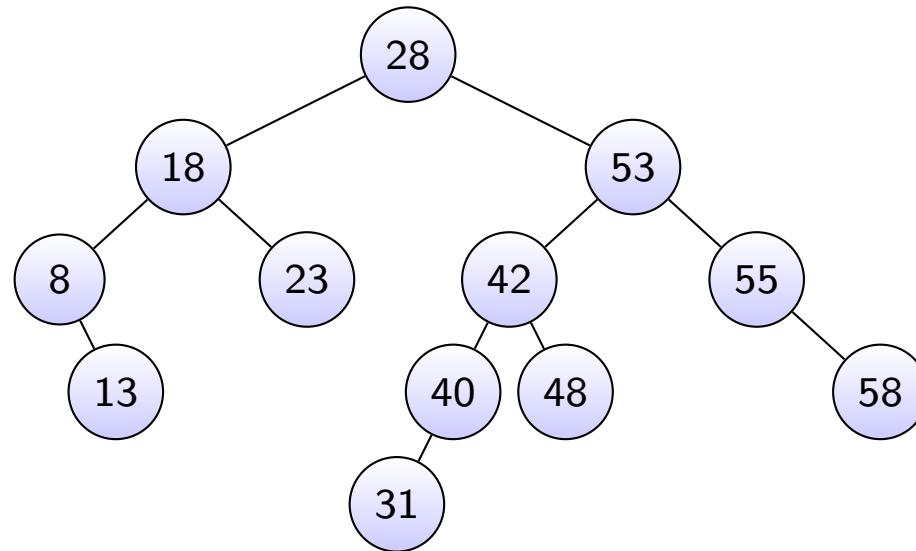
# Number of restructurings

When deleting 18 from the following AVL Tree, how many tri-node restructurings are needed?

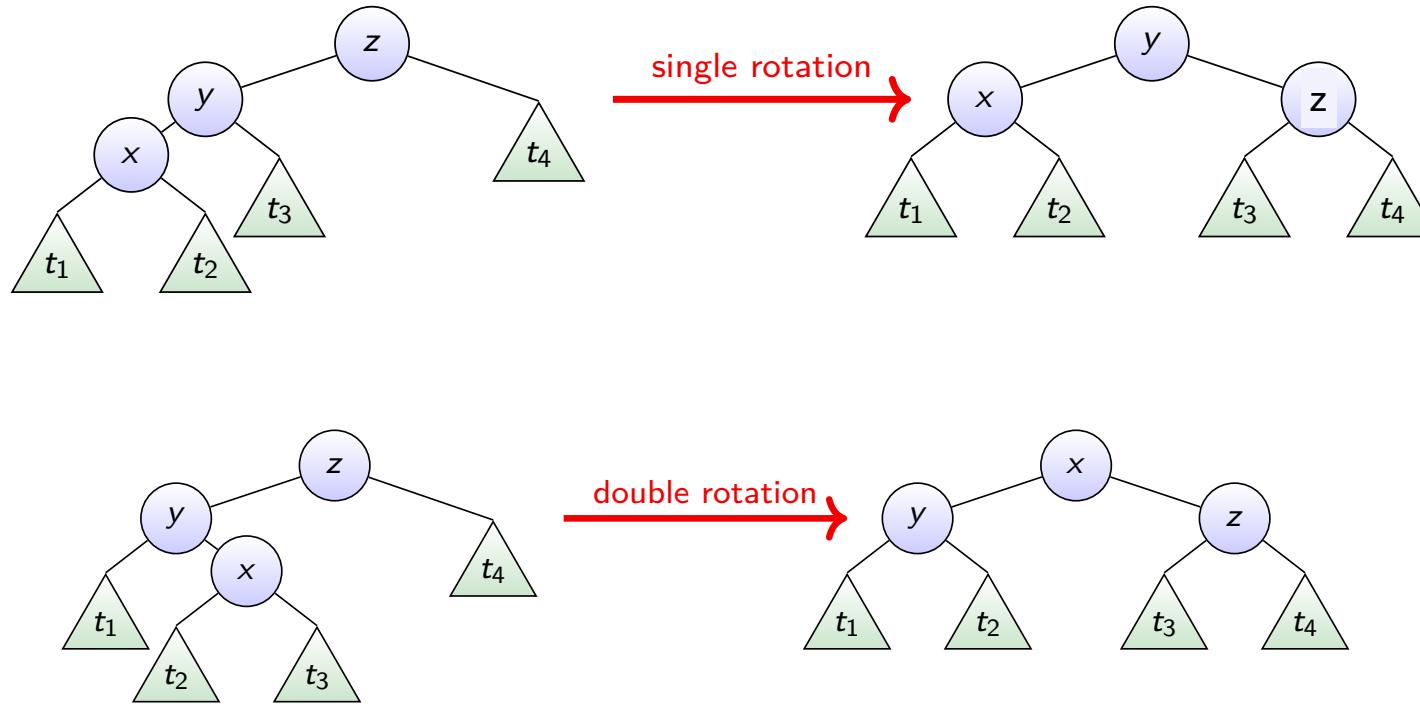


# Single or double rotations

When deleting 18 from the following AVL Tree, would you need a single or a double rotation?



# Single and Double Rotations



- For a **deletion** at position **p**, let **z** be the first unbalanced position encountered going up from **p** towards the root.
- Let **y** be the child of **z** with greater height. Let **x** be the taller child of **y** if the heights of children of **y** differ. Otherwise let **x** be the child of **y** that is on the same side as **y**

# Time complexity

What is the time complexity of the core operations (get, put, remove) in AVL Tree?

# Time complexity

What is the time complexity of the core operations (get, put, remove) in AVL Tree?  $O(\log n)$

- Height of AVL tree with  $n$  keys is  $O(\log n)$
- A tri-node restructuring takes  $O(1)$  time
- Number of required tri-node restructurings are bounded by the height of the AVL tree

# Height of AVL Tree

Argue why the height of the AVL tree is  $O(\log n)$ ?

# Choosing the best data structure

Consider that you are grading the final exam of Algorithms and Data Structures, and would like to use a **data structure to store pairs ( $s, g$ )**, where  $s$  is the student number and  $g$  is the corresponding grade. While grading you take exams from the pile in any order, and you would like to **access and eventually update or replace the grade of each student multiple times**. The **number of students is known**, but the **range of student numbers is not fixed**. What data structure would provide the **best expected time complexity for a large number of accesses and updates?**

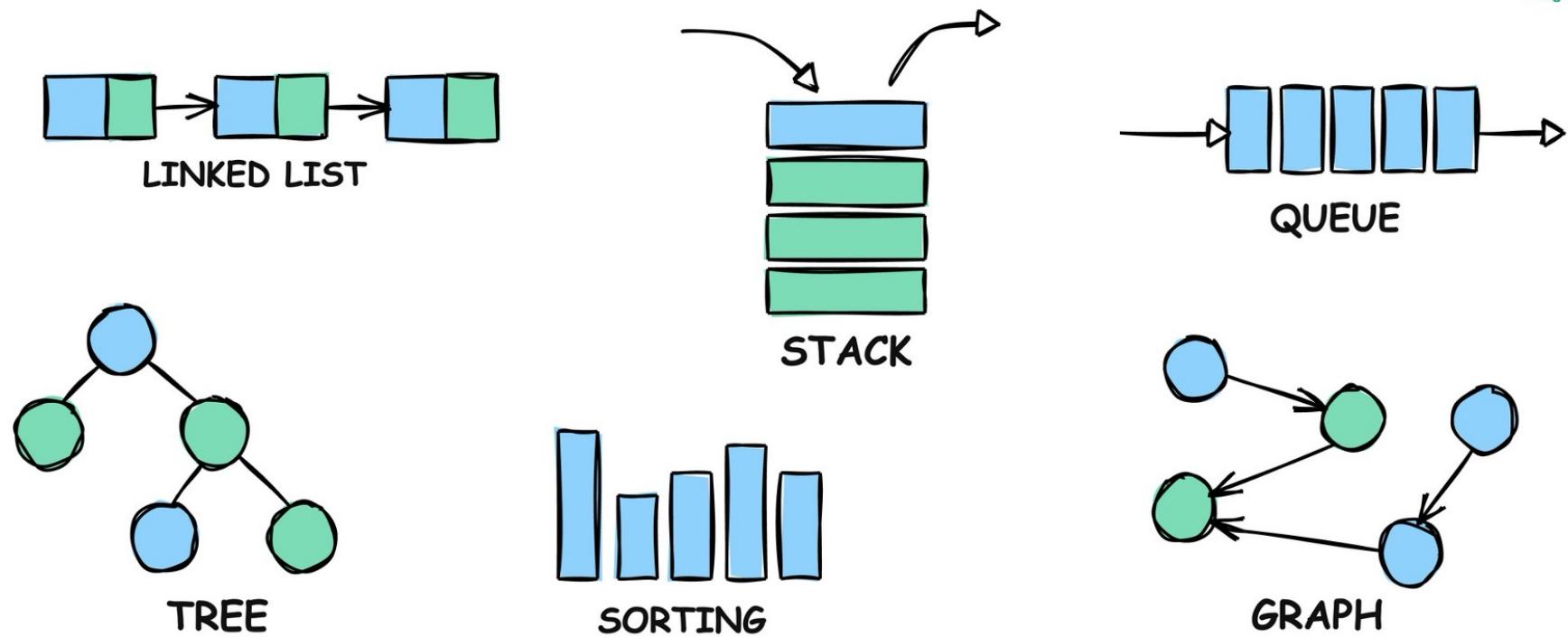
- A. AVL Tree
- B. Hash Table
- C. Array List
- D. Linked List

# Choosing the best data structure

Consider that you are grading the final exam of Algorithms and Data Structures, and would like to use a **data structure to store pairs ( $s, g$ )**, where  $s$  is the student number and  $g$  is the corresponding grade. While grading you take exams from the pile in any order, and you would like to **access and eventually update or replace the grade of each student multiple times**. The **number of students is known**, but the **range of student numbers is not fixed**. What data structure would provide the **best expected time complexity for a large number of accesses and updates?**

- A. AVL Tree
- B. Hash Table**
- C. Array List
- D. Linked List

The usage scenario does not require that the student numbers are sorted, and an AVL tree has larger time complexity for methods get and put than a properly behaved hash table. An array list is not a good solution, since the student numbers may occur in a large range. A linked list is always a bad solution for this scenario, since it does not enable efficient searching.

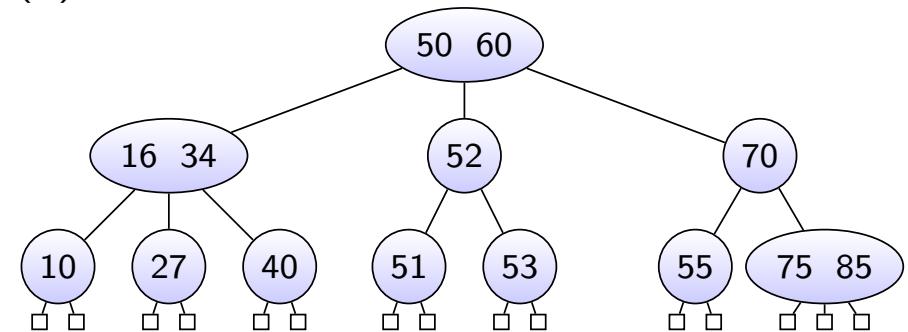
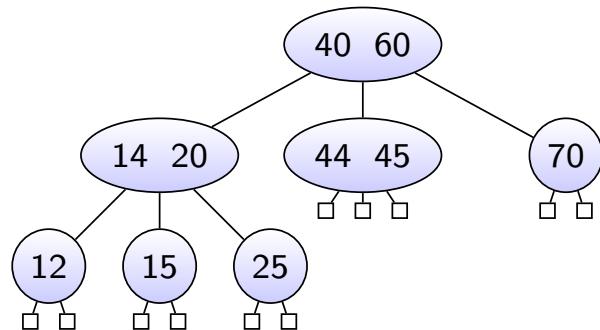


# Algorithms & Data Structures

Megha Khosla, Ivo van Kreveld

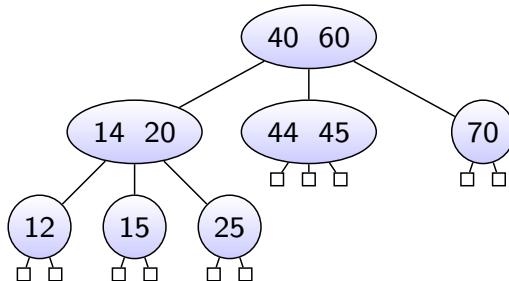
# (2,4) Trees

Which of the following are (2,4) trees?

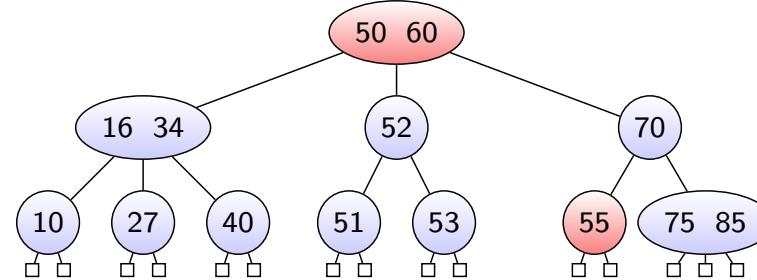


# (2,4) Trees

Which of the following are (2,4) trees?



No, the depth property is violated



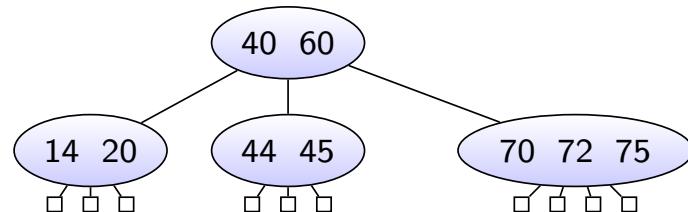
No,  $55 \not> 60$

# Insertion in (2,4) Trees

Perform multiway search, and put the new entry at the *parent* of the leaf (i.e. parent of the **null**) where the search ended

- ▶ This preserves the *depth property*
- ▶ This may break the *size property* by causing an **overflow**:  
a node may become a 5-node

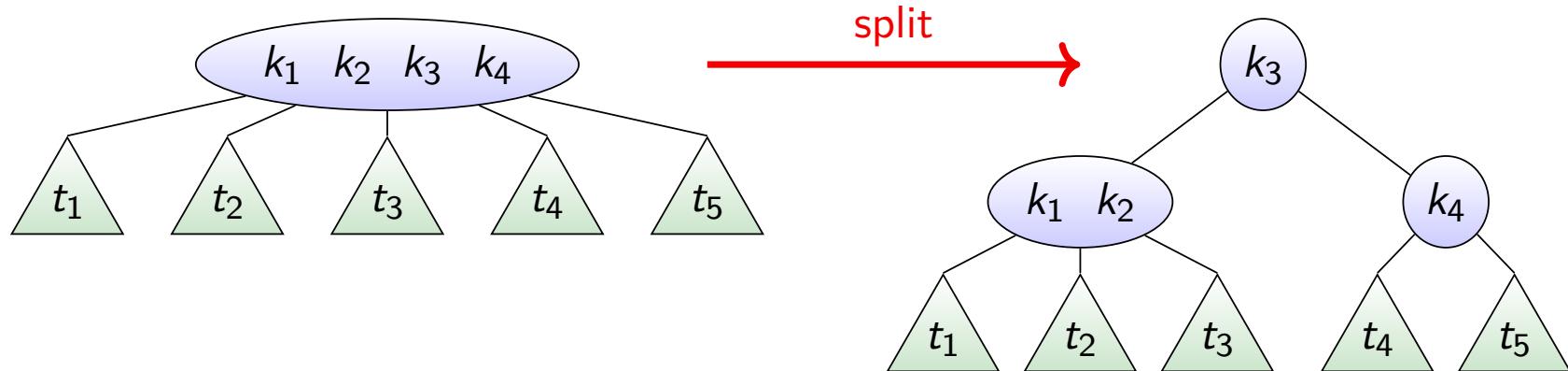
**Example:** insert 74



# Fixing overflow in (2,4) Trees

Repairing an overflow:

- ▶ We perform a **split** of the node that became a 5-node:



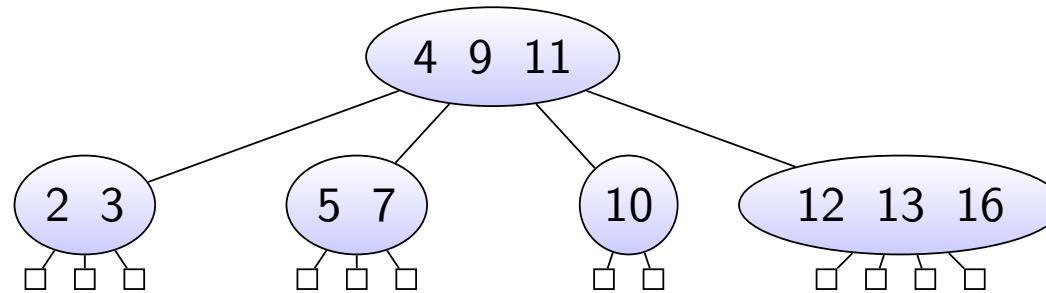
- ▶ If the node was the root, the resulting top root is the new root
- ▶ Otherwise, we merge the resulting top node with its parent
- ▶ If the parent overflows, we proceed recursively

# Insertion in (2,4) Trees



When inserting 18 into the following tree, how many overflows are caused?

- (a) none (b) one (c) two (d) three

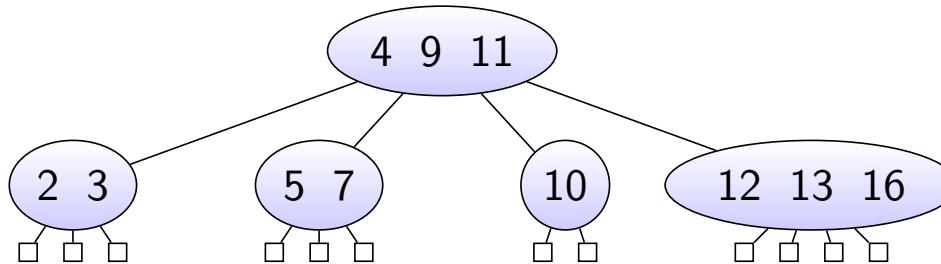


# Insertion in (2,4) Trees



When inserting 18 into the following tree, how many overflows are caused?

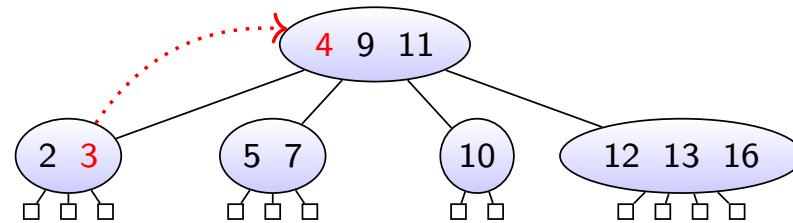
- (a) none (b) one (c) **two** (d) three



- ▶ We get an overflow in **12 13 16 18**, so 16 propagates up
- ▶ We then get another overflow in **4 9 11 16**, so we get 11 as the new root

# Deletion in (2,4) Trees

**Example:** deleting 4 can be reduced to deleting 3



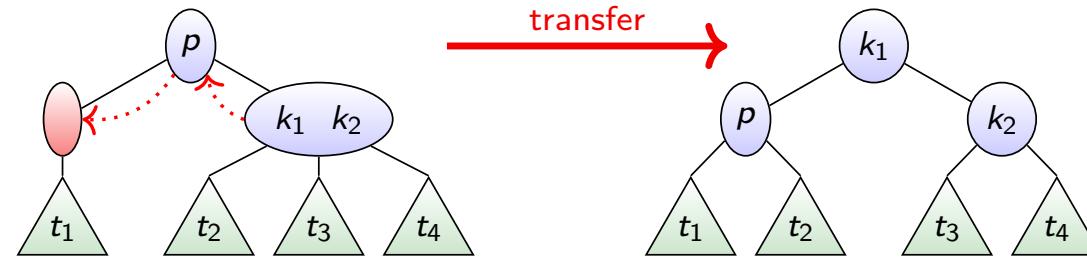
Recipe for removing an entry:

- ▶ Use multiway search to find the entry that needs to be removed
- ▶ If the entry is **not a node of level 1** (i.e. its children are not **null**) then:
  - ▶ Find the rightmost (=maximal) entry in the left subtree
  - ▶ Swap that entry with the key that has to be removed
  - ▶ Proceed with the rightmost entry
- ▶ Remove the entry, and ‘repair’ the tree while traversing upwards

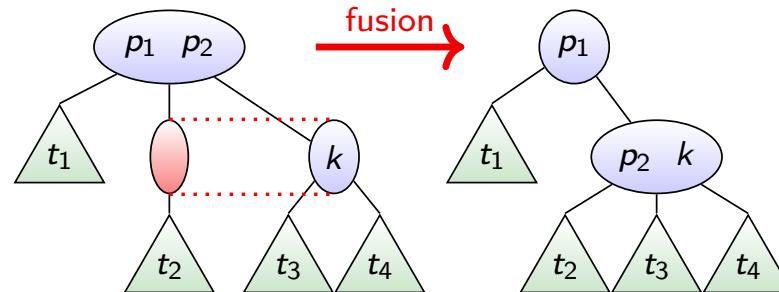
# Deletion in (2,4) Trees

Removal may break the *size property* by causing an **underflow**: a node may become a 1-node. To handle an underflow, we consider two cases (in given order):

- (a) An adjacent sibling is a 3-node or a 4-node, we then perform a **transfer**:



- (b) An adjacent sibling is a 2-node, we then perform a **fusion**:

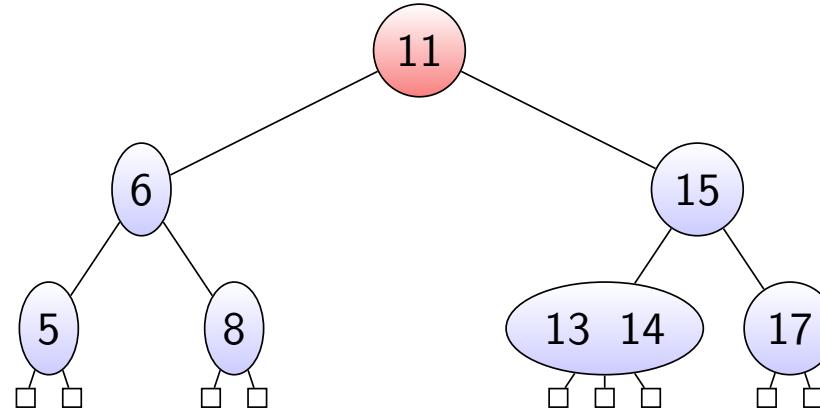


# Deletion in (2,4) Trees



Which rebalancing operations need to be performed when removing 11?

- (a) none
- (b) one transfer
- (c) one fusion
- (d) two fusions
- (e) one fusion and one transfer

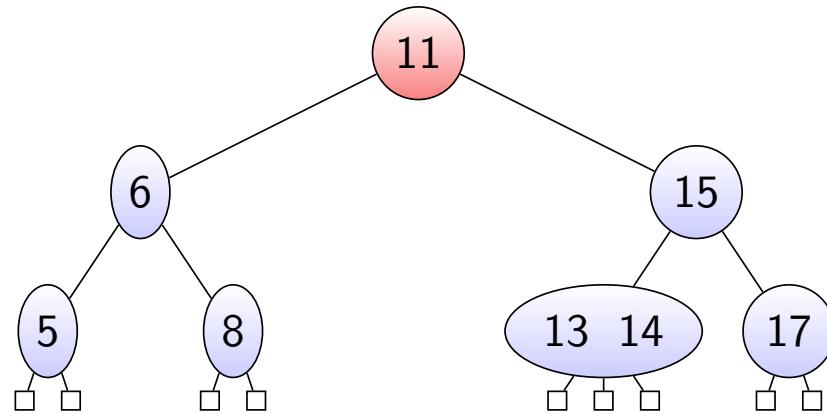


# Deletion in (2,4) Trees



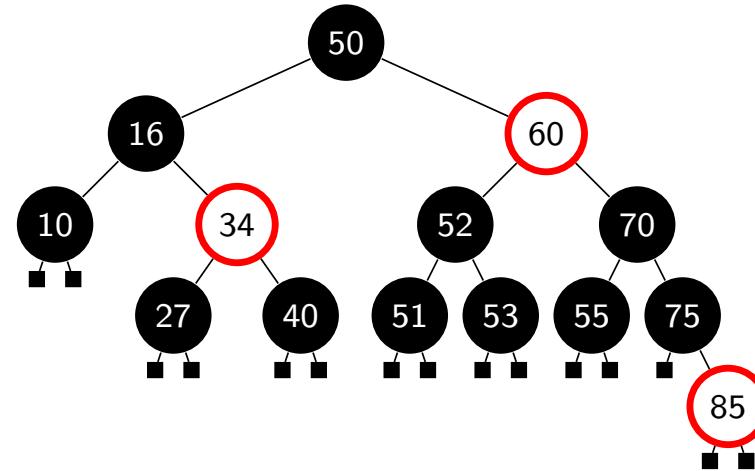
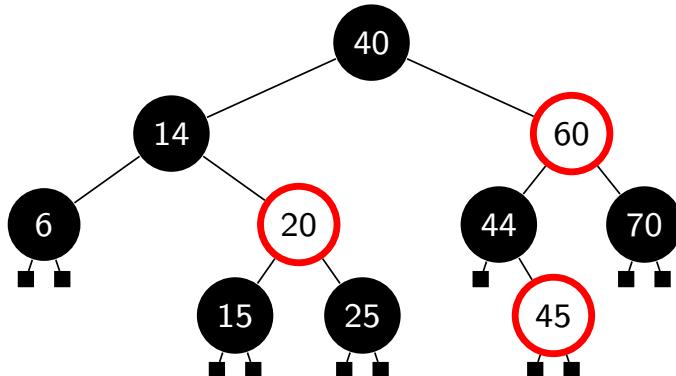
Which rebalancing operations need to be performed when removing 11?

- (a) none
- (b) one transfer
- (c) one fusion
- (d) **two fusions**
- (e) one fusion and one transfer



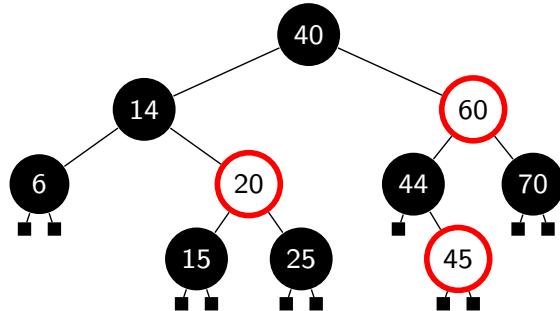
# Red-Black Trees

Which of the following are Red-Black trees?

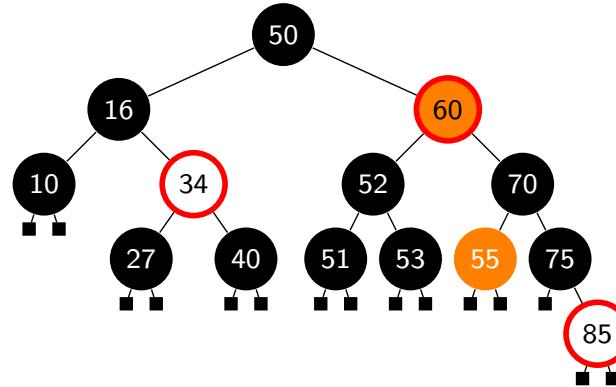


# Red-Black Trees

Which of the following are Red-Black trees?

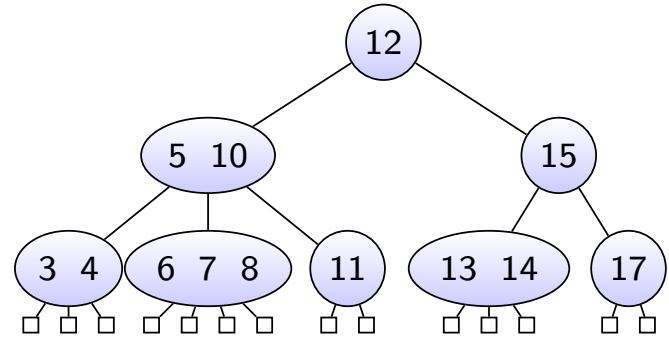
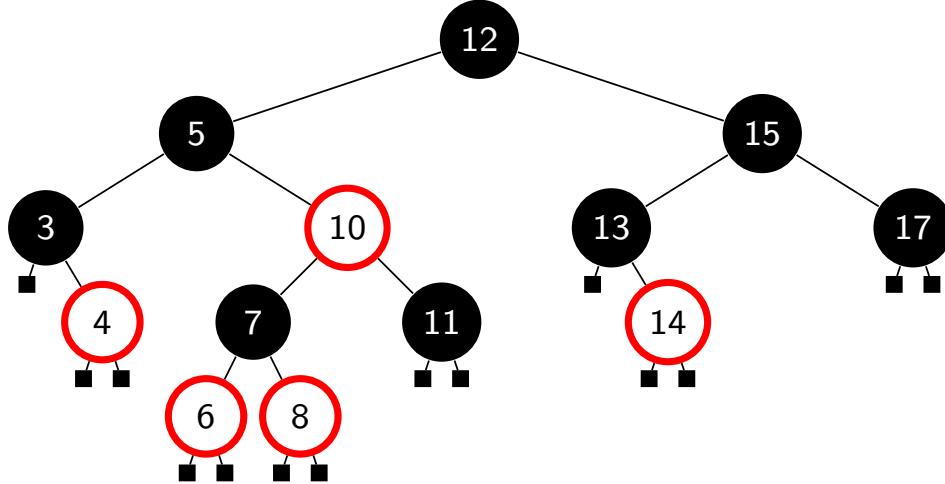


No, the depth property is violated

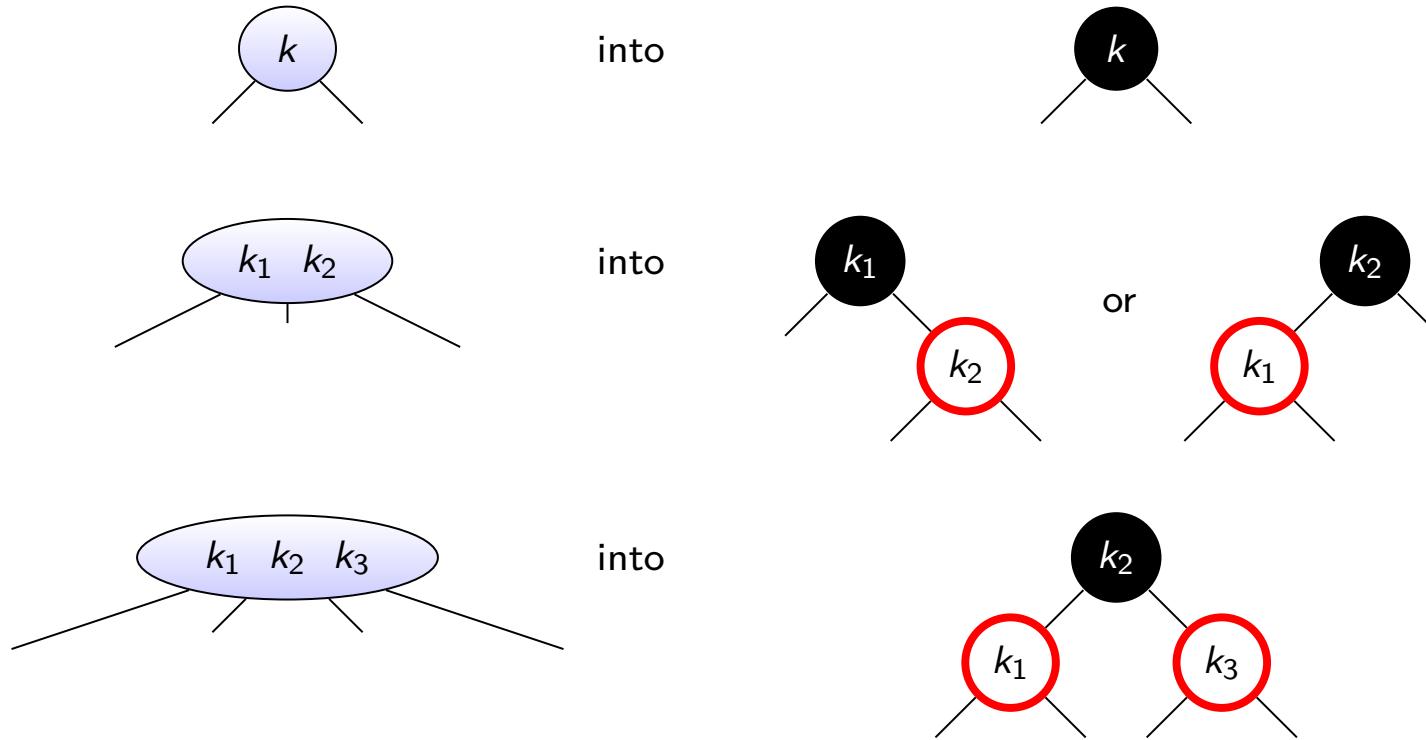


No,  $55 \not> 60$

# Red-Black Trees Vs (2,4) Trees

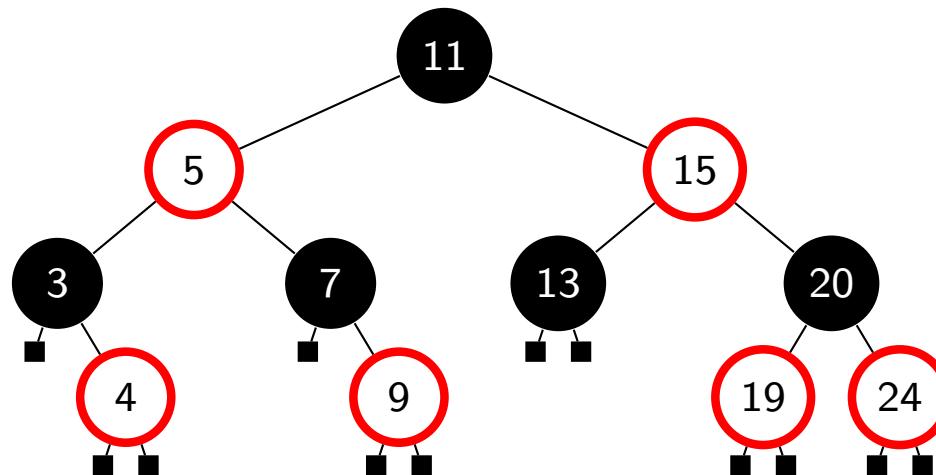


# Red-Black Trees Vs (2,4) Trees



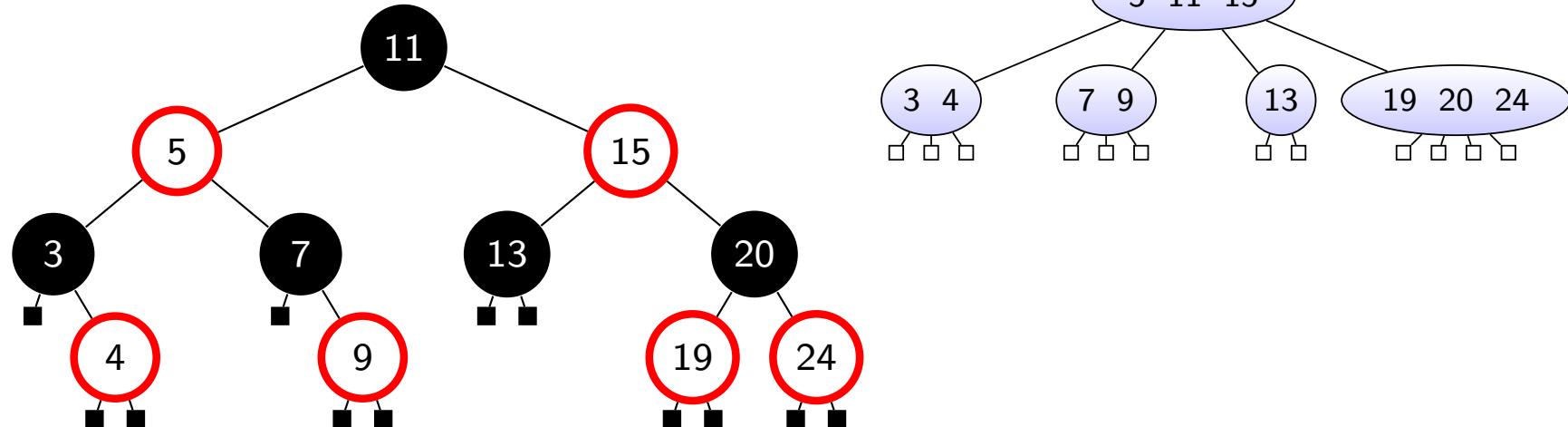
# Red-Black Trees Vs (2,4) Trees

What is the corresponding (2,4) Tree?



# Red-Black Trees Vs (2,4) Trees

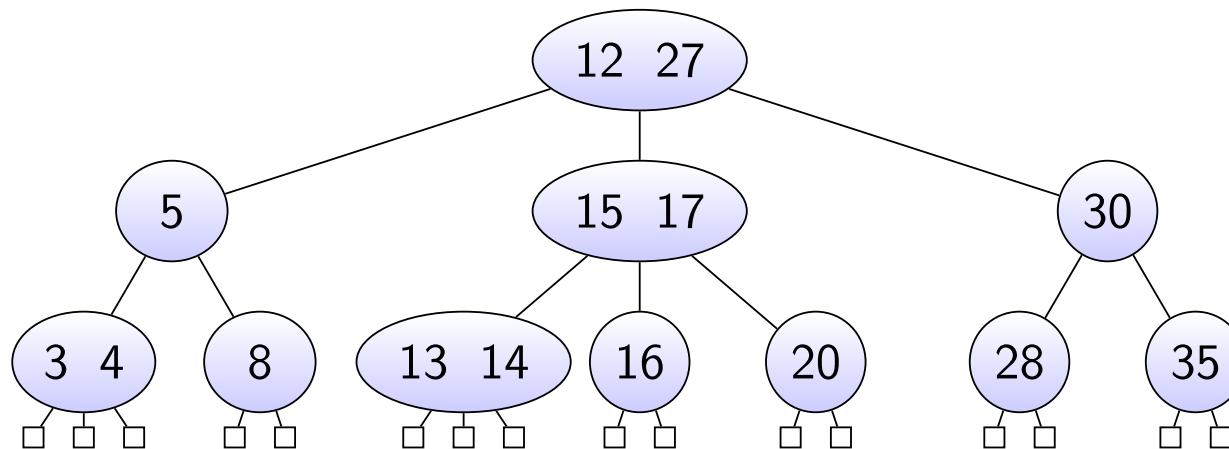
## What is the corresponding (2,4) Tree?



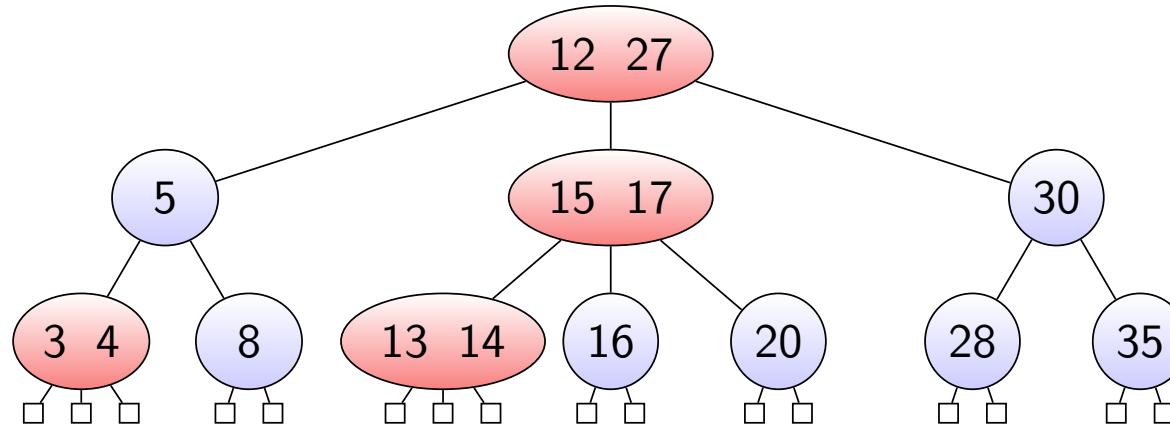
# Red-Black Trees Vs (2,4) Trees

How many red-black trees could correspond to the following (2,4) tree?

- A) 1
- B) 2
- C) 8
- D) 16



# Red-Black Trees Vs (2,4) Trees



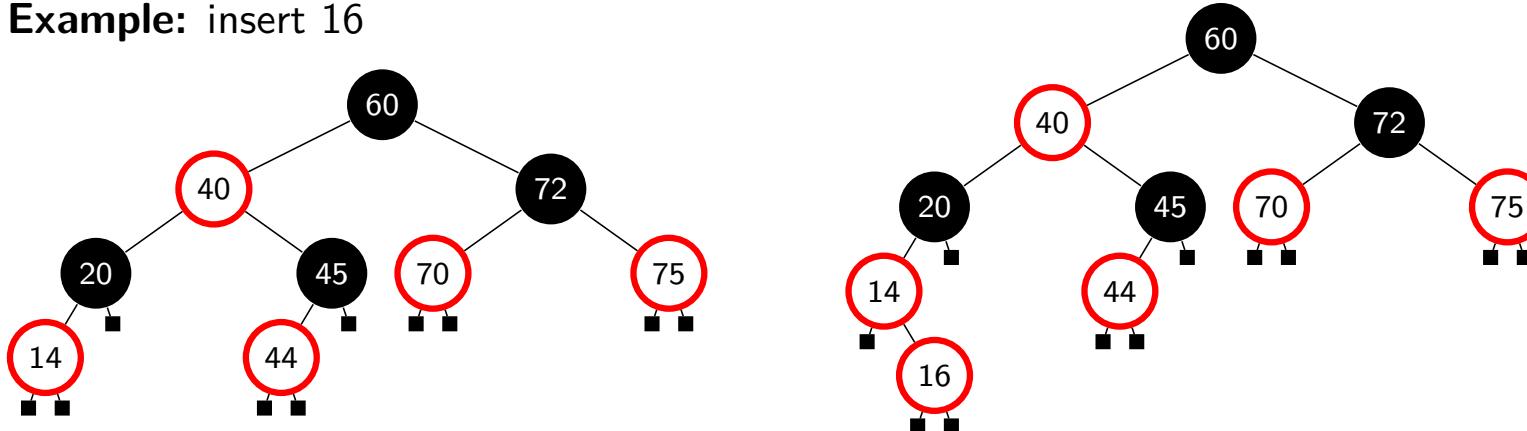
For each 3-node we have two choices. The above tree contains four 3-nodes, so  $2^4 = 16$  red-black trees correspond to this (2,4) tree

# Insertion in Red-Black Trees

Perform binary search, and put the new red entry at the leaf (i.e. the `null`) where the search ended, and add two black leaves to the just created entry

- ▶ This preserves the *depth property*
- ▶ This may break the *red property*:  
two red nodes may be above each other

**Example:** insert 16



# Case 1: Restructuring

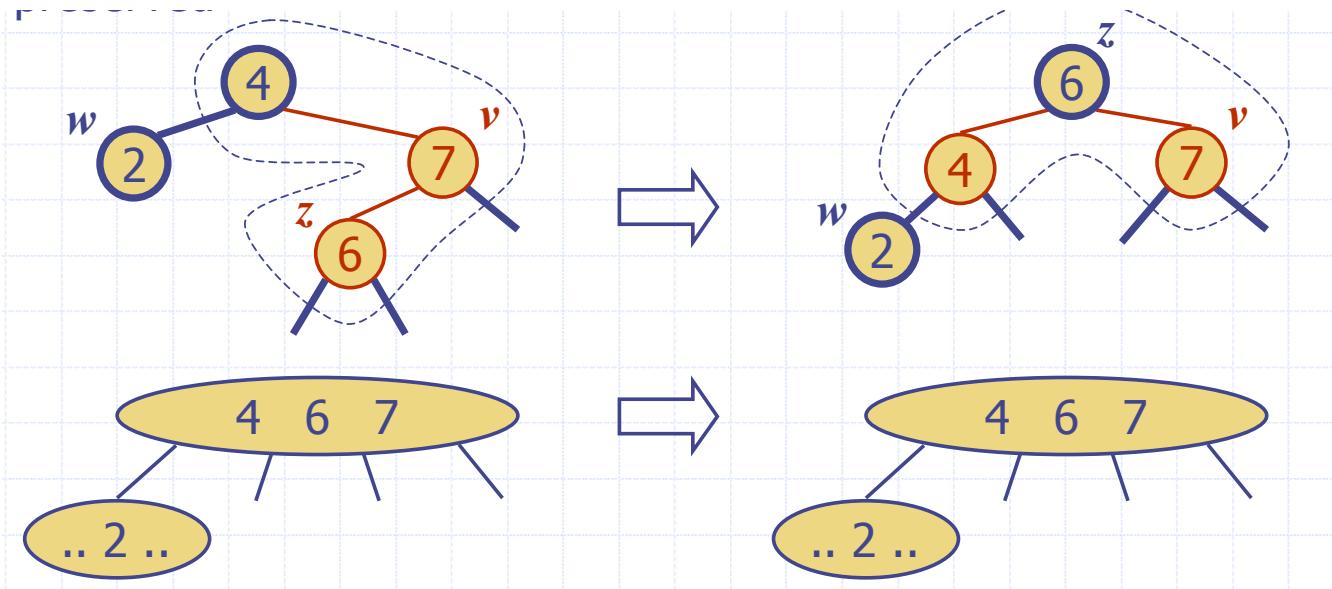
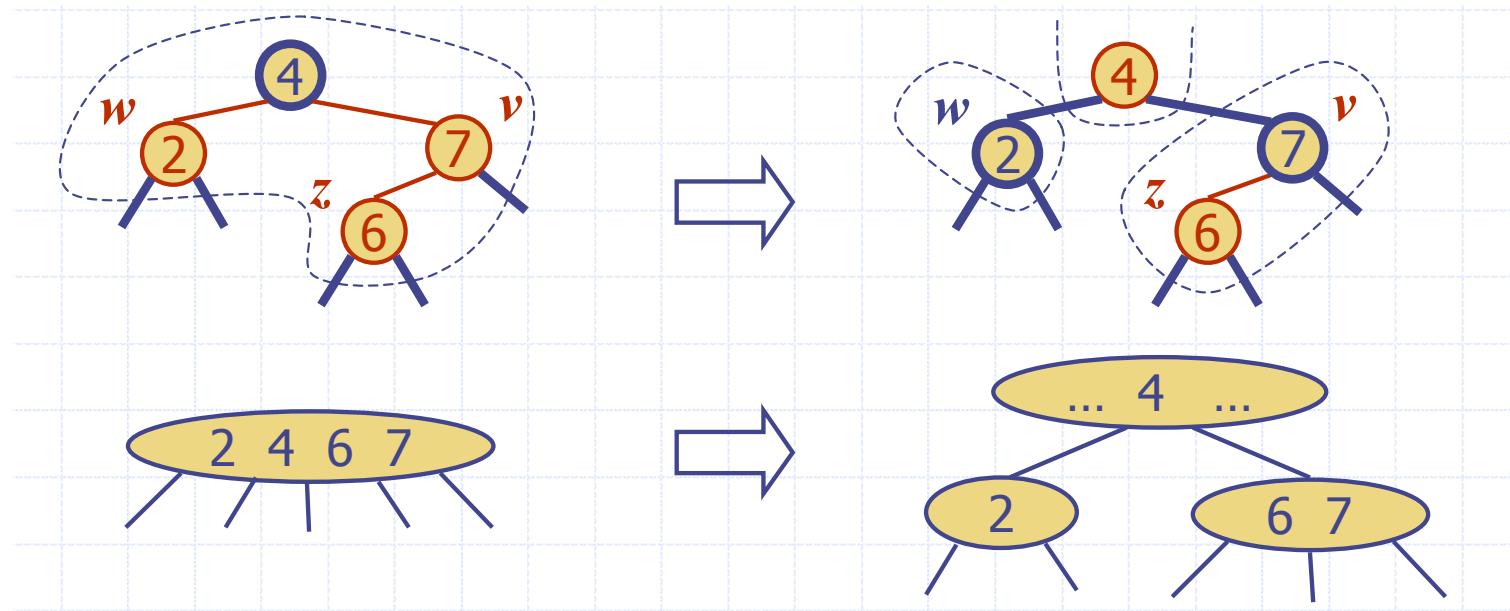


Image Source: <https://cs.brown.edu/cgc/java2.datastructures.net/presentations/RedBlackTrees.pdf>

# Case 2: Recoloring

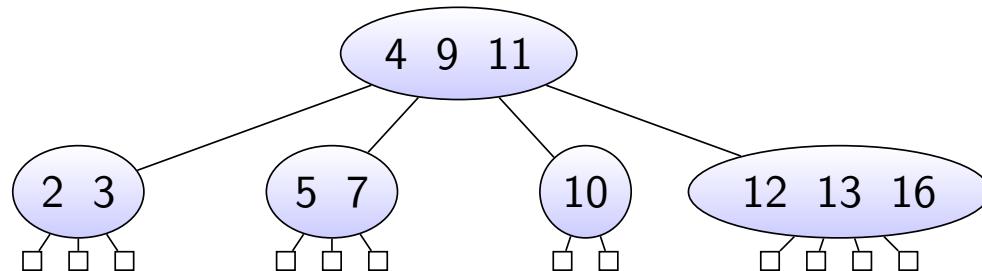


# Insertion in Red-Black Trees



When inserting 18 into the Red-Black tree that corresponds to this (2,4) Tree, what happens?

- (a) Nothing, in the Red-Black tree, inserting 18 wouldn't be a problem
- (b) A single tri-node restructuring
- (c) A single recoloring
- (d) 2 tri-node restructurings
- (e) A tri-node restructuring and a recoloring
- (f) 2 recolorings

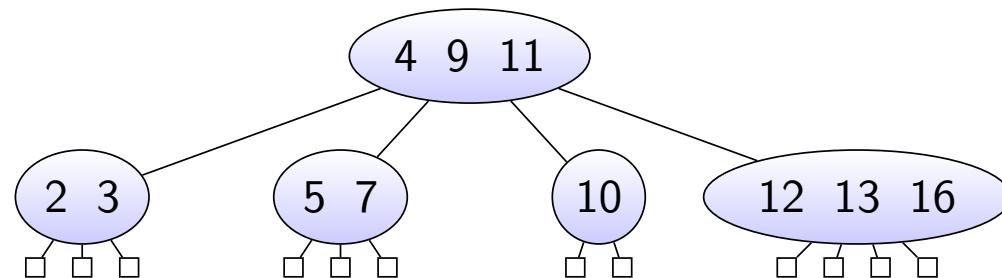


# Insertion in Red-Black Trees



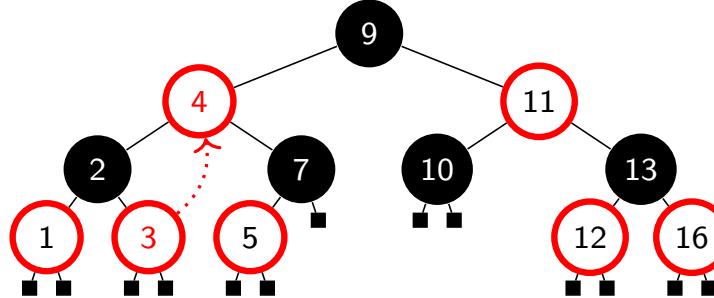
When inserting 18 into the Red-Black tree that corresponds to this (2,4) Tree, what happens?

- (a) Nothing, in the Red-Black tree, inserting 18 wouldn't be a problem
- (b) A single tri-node restructuring
- (c) A single recoloring
- (d) 2 tri-node restructurings
- (e) A tri-node restructuring and a recoloring
- (f) **2 recolorings**



# Deletion in Red-Black Trees

**Example:** deleting 4 can be reduced to deleting 3

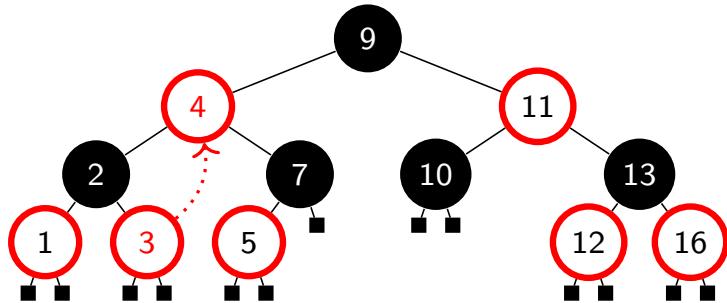


Recipe for removing an entry:

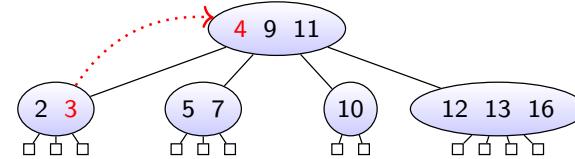
- ▶ Use binary search to find the entry that needs to be removed
- ▶ If the entry is **not a node of level 1** (i.e. its children are not **null**) then:
  - ▶ Find the rightmost (=maximal) entry in the left subtree
  - ▶ Swap that entry with the key that has to be removed
  - ▶ Proceed with the rightmost entry
- ▶ Remove the entry, and ‘repair’ the tree while traversing upwards

# Deletion in Red-Black Trees

**Example:** deleting 4 can be reduced to deleting 3



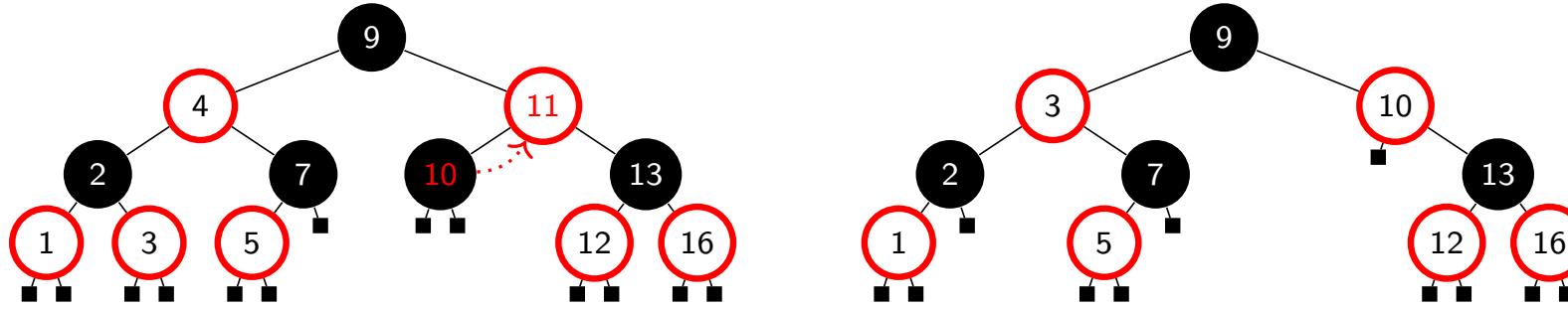
**Example:** deleting 4 can be reduced to deleting 3



Recipe for removing an entry:

- ▶ Use binary search to find the entry that needs to be removed
- ▶ If the entry is **not a node of level 1** (i.e. its children are not **null**) then:
  - ▶ Find the rightmost (=maximal) entry in the left subtree
  - ▶ Swap that entry with the key that has to be removed
  - ▶ Proceed with the rightmost entry
- ▶ Remove the entry, and ‘repair’ the tree while traversing upwards

**Example:** deleting 11 can be reduced to deleting 10

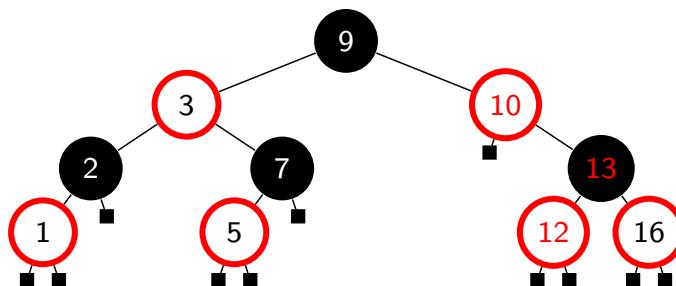


Removal may break the *depth property*

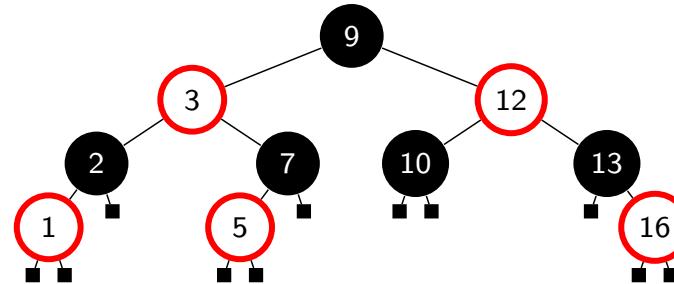
**Corresponds to removing a 2 node in (2,4) Tree**

# Case 1 : Restructure (Transfer in (2,4) Tree)

Example: remove 11



Result after a restructure:

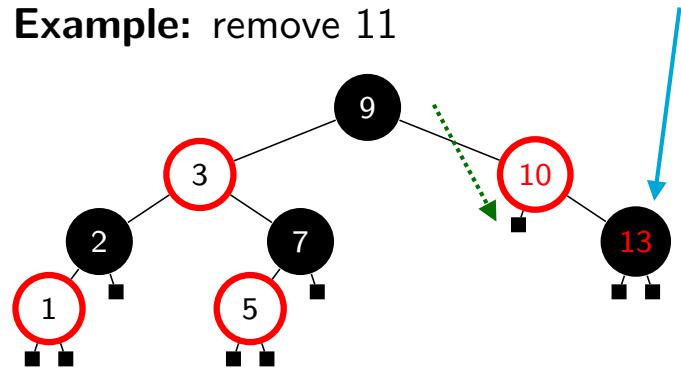


When sibling of removed position has a red child

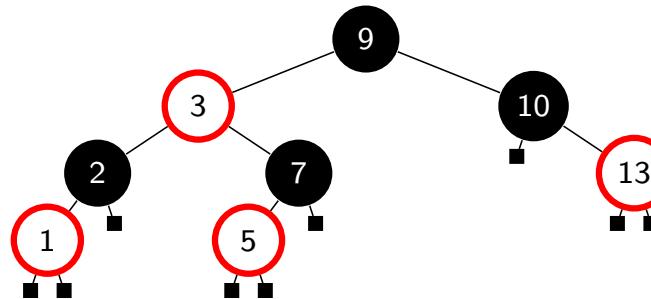
**then Sibling of 1-node is a 3 or 4 node in the equivalent (2,4) Tree**

# Case 2 : Recoloring (Fusion in (2,4) Tree)

Example: remove 11



Result after a restructure:



When sibling of removed position is black and has both children black

**then Sibling of 1-node is a 2-node in the equivalent (2,4) Tree**

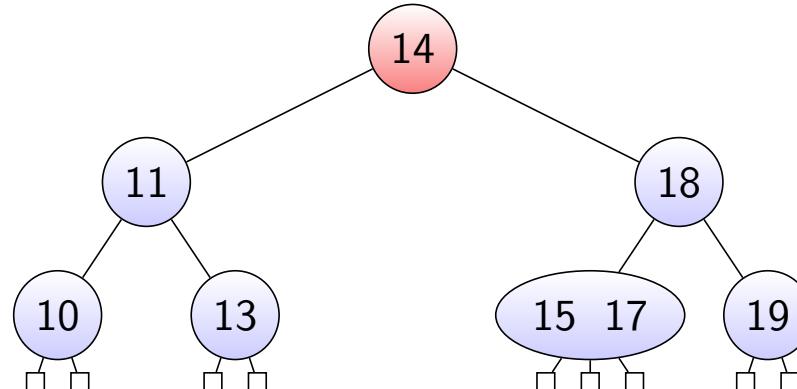
Case 3 : when sibling of the deleted position is red

# Deletion in Red-Black Trees



Which rebalancing operations need to be performed when removing 11 from the Red-Black Tree corresponding to this (2,4) Tree?

- (a) Nothing, in the Red-Black tree, removing 11 wouldn't be a problem
- (b) A single tri-node restructuring
- (c) A single recoloring
- (d) 2 tri-node restructurings
- (e) A tri-node restructuring and a recoloring
- (f) 2 recolorings

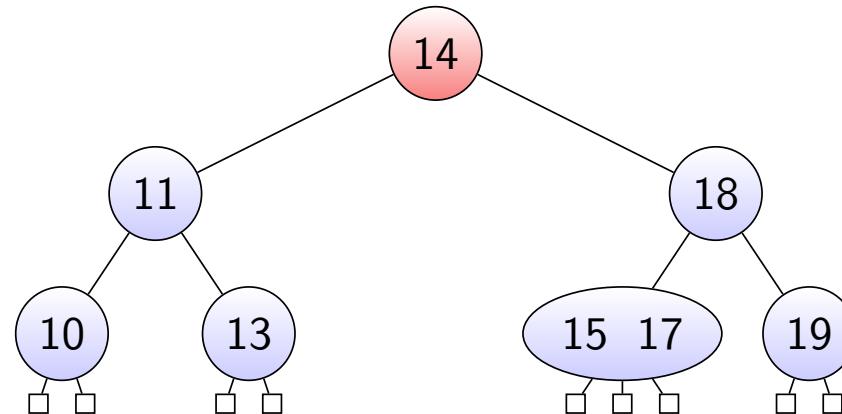


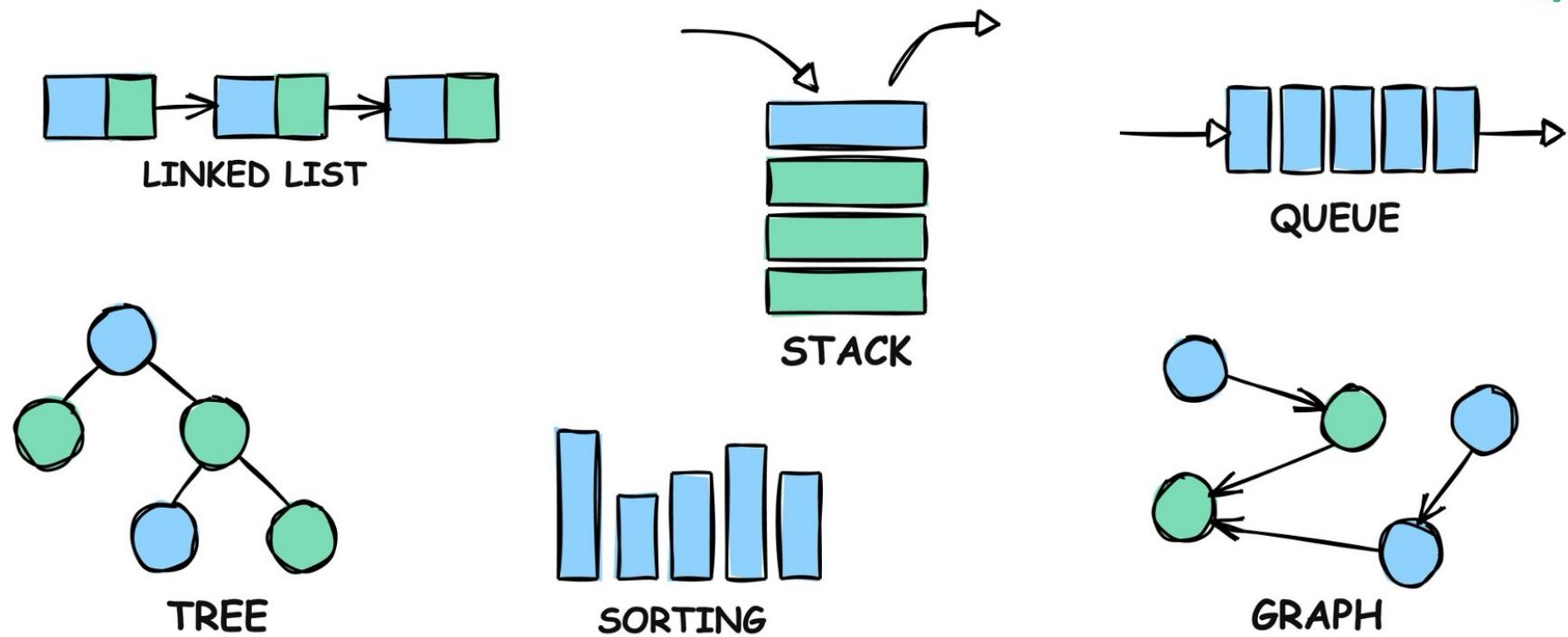
# Deletion in Red-Black Trees



Which rebalancing operations need to be performed when removing 11 from the Red-Black Tree corresponding to this (2,4) Tree?

- (a) Nothing, in the Red-Black tree, removing 11 wouldn't be a problem
- (b) A single tri-node restructuring
- (c) A single recoloring
- (d) 2 tri-node restructurings
- (e) A tri-node restructuring and a recoloring
- (f) **2 recolorings**





# Algorithms & Data Structures

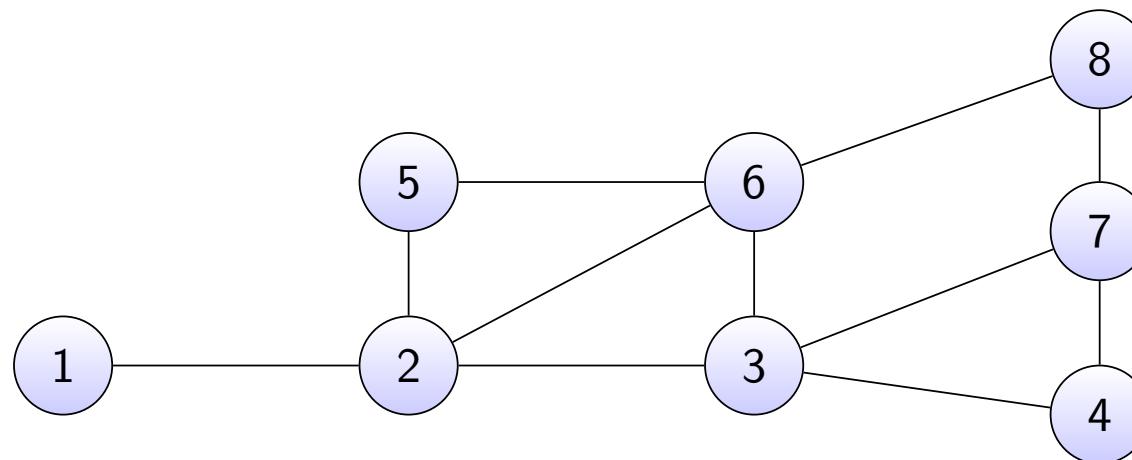
Megha Khosla, Ivo van Kreveld

# Graph

A **graph** is a tuple  $(V, E)$ , where:

- ▶  $V$  is a set of nodes, called **vertices**
- ▶  $E$  is a collection of pairs of vertices, called **edges**

**Example:**



$$V = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

$$E = \{(1, 2), (2, 3), (2, 5), (2, 6), (3, 4), (3, 6), (3, 7), (4, 7), (5, 6), (6, 8), (7, 8)\}$$

# Examples

Think of some examples which you can represent as graphs ?

Are your graphs directed or undirected, weighted or unweighted?

# Examples

- ▶ An object-oriented program  
Vertices: classes, directed edges: inheritance between classes
- ▶ Plumbing in a building  
Vertices: tanks, valves and taps, directed edges: water pipes
- ▶ The computer network in a building  
Vertices: computers, servers and switches, undirected edges: wires
- ▶ Trees  
Vertices: nodes, directed edges: child relationship

# Properties of Graphs

Given an undirected graph  $G = (V, E)$  with  $n$  vertices

and  $m$  edges, show that  $\sum_{v \in V} \deg(v) = 2m$

# Properties of Graphs

For a directed graph  $G = (V, E)$

A.  $\sum_{v \in V} indeg(v) = \sum_{v \in V} outdeg(v)$

B.  $\sum_{v \in V} indeg(v) > \sum_{v \in V} outdeg(v)$

C.  $\sum_{v \in V} indeg(v) < \sum_{v \in V} outdeg(v)$

# Properties of Graphs

For a directed graph  $G = (V, E)$

A.  $\sum_{v \in V} \text{indeg}(v) = \sum_{v \in V} \text{outdeg}(v)$

B.  $\sum_{v \in V} \text{indeg}(v) > \sum_{v \in V} \text{outdeg}(v)$

C.  $\sum_{v \in V} \text{indeg}(v) < \sum_{v \in V} \text{outdeg}(v)$

# Properties of Graphs

What is the maximum number of edges in a simple graph with  $n$  vertices?

# Properties of Graphs

- ▶ A graph is **connected** if there is a path between every pair of vertices
- ▶ A **tree** is a connected undirected graph without cycles
- ▶ A **forest** is an undirected graph without cycles



Given an undirected graph  $G$  with  $n > 0$  vertices and  $m$  edges. (Select the most precise answer).

If  $G$  is a tree then:

- (a)  $m = n$
- (b)  $m > n - 1$
- (c)  $m \geq n - 1$
- (d)  $m = n - 1$
- (e)  $m \leq n - 1$

If  $G$  is a forest then:

- (a)  $m = n$
- (b)  $m > n - 1$
- (c)  $m \geq n - 1$
- (d)  $m = n - 1$
- (e)  $m \leq n - 1$

If  $G$  is connected then:

- (a)  $m = n$
- (b)  $m > n - 1$
- (c)  $m \geq n - 1$
- (d)  $m = n - 1$
- (e)  $m \leq n - 1$

# Properties of Graphs

- ▶ A graph is **connected** if there is a path between every pair of vertices
- ▶ A **tree** is a connected undirected graph without cycles
- ▶ A **forest** is an undirected graph without cycles



Given an undirected graph  $G$  with  $n > 0$  vertices and  $m$  edges. (Select the most precise answer).

If  $G$  is a tree then:

- (a)  $m = n$
- (b)  $m > n - 1$
- (c)  $m \geq n - 1$
- (d)  $\textcolor{red}{m = n - 1}$
- (e)  $m \leq n - 1$

If  $G$  is a forest then:

- (a)  $m = n$
- (b)  $m > n - 1$
- (c)  $m \geq n - 1$
- (d)  $m = n - 1$
- (e)  $\textcolor{red}{m \leq n - 1}$

If  $G$  is connected then:

- (a)  $m = n$
- (b)  $m > n - 1$
- (c)  $\textcolor{red}{m \geq n - 1}$
- (d)  $m = n - 1$
- (e)  $m \leq n - 1$

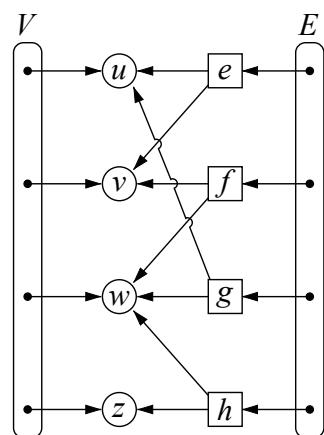
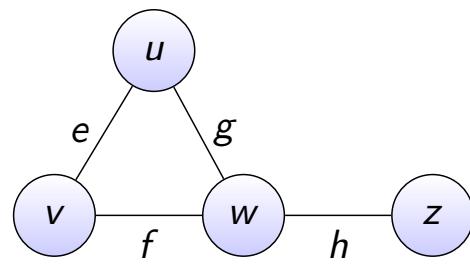
# Data Structures for Graphs

- Four representations
- Each representation maintains a collection to store all the vertices of G
- Different based on how edges are stored

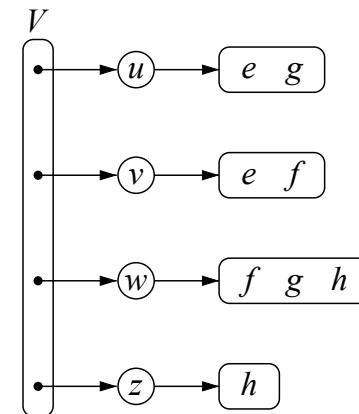
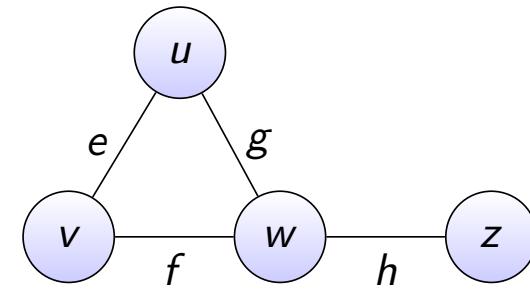
Edge List    Adjacency List    Adjacency Map    Adjacency Matrix

# Data Structures for Graphs

Edge List

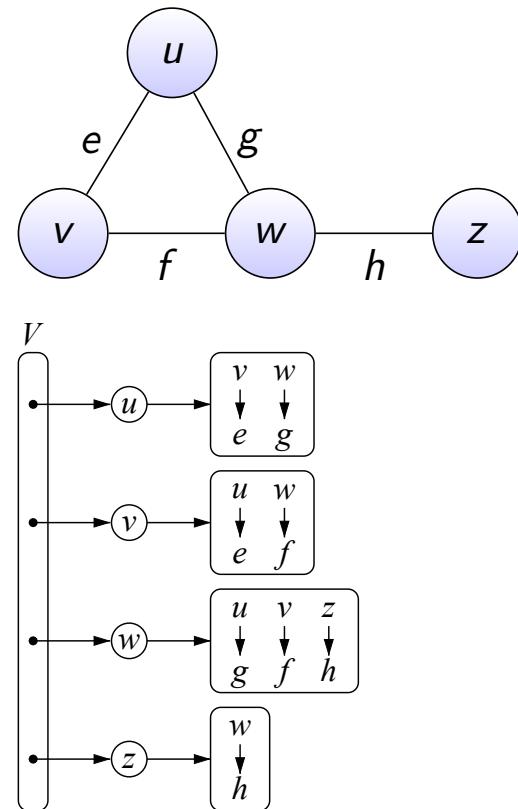


Adjacency List



# Data Structures for Graphs

Adjacency Map



Adjacency Matrix

	0	1	2	3
$u$	→ 0	$e$	$g$	
$v$	→ 1	$e$		$f$
$w$	→ 2	$g$	$f$	$h$
$z$	→ 3		$h$	

# Data Structures for Graphs

- In which of the representations can we access an edge in  $O(1)$  time in the worst case?
  - Which representation has the highest time complexity for retrieving an edge ?

# Data Structures for Graphs

- Which representation incurs the worst space complexity?

# Data Structures for Graphs

For what type of graphs would you prefer adjacency matrix over the other representations?

# Asymptotic Performance

	Edge list	Adj. list	Adj. map	Adj. matrix
Space usage outgoingEdges(v) getEdge(v,w) insertVertex(x) insertEdge(v,w,x) removeVertex(v) removeEdge(e)				

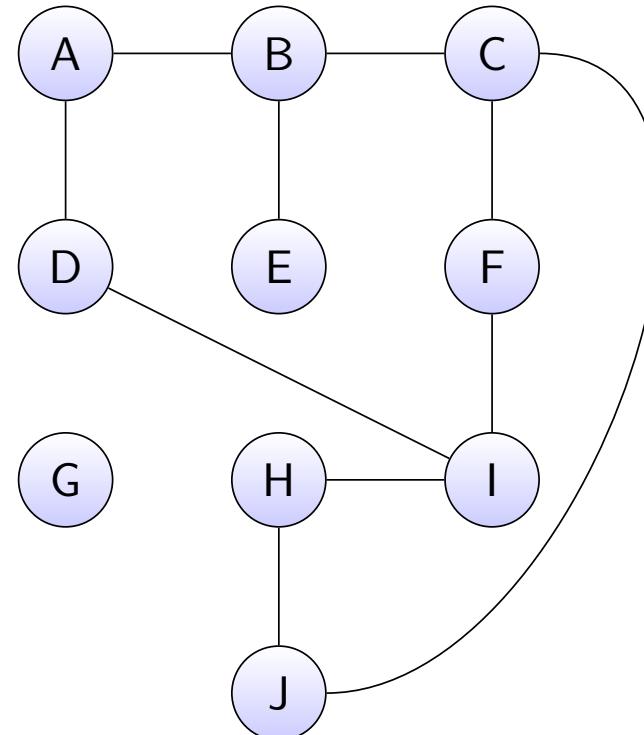
# Asymptotic Performance

	<b>Edge list</b>	<b>Adj. list</b>	<b>Adj. map</b>	<b>Adj. matrix</b>
Space usage	$\mathcal{O}(n + m)$	$\mathcal{O}(n + m)$	$\mathcal{O}(n + m)$	$\mathcal{O}(n^2)$
outgoingEdges(v)	$\mathcal{O}(m)$	$\mathcal{O}(\deg(v))$	$\mathcal{O}(\deg(v))$	$\mathcal{O}(n)$
getEdge(v, w)	$\mathcal{O}(m)$	$\mathcal{O}(\min(\deg(v), \deg(w)))$	$\mathcal{O}(1)$ exp.	$\mathcal{O}(1)$
insertVertex(x)	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n^2)$
insertEdge(v, w, x)	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$ exp.	$\mathcal{O}(1)$
removeVertex(v)	$\mathcal{O}(m)$	$\mathcal{O}(\deg(v))$	$\mathcal{O}(\deg(v))$	$\mathcal{O}(n^2)$
removeEdge(e)	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$ exp.	$\mathcal{O}(1)$

# Graph Traversal



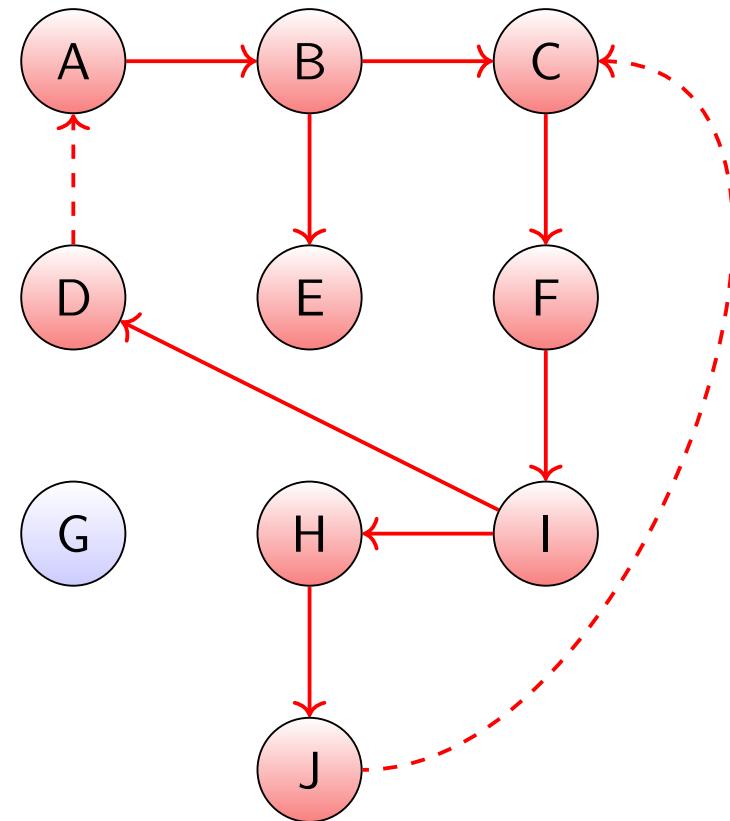
Perform a depth-first search starting in *A*:



# Graph Traversal



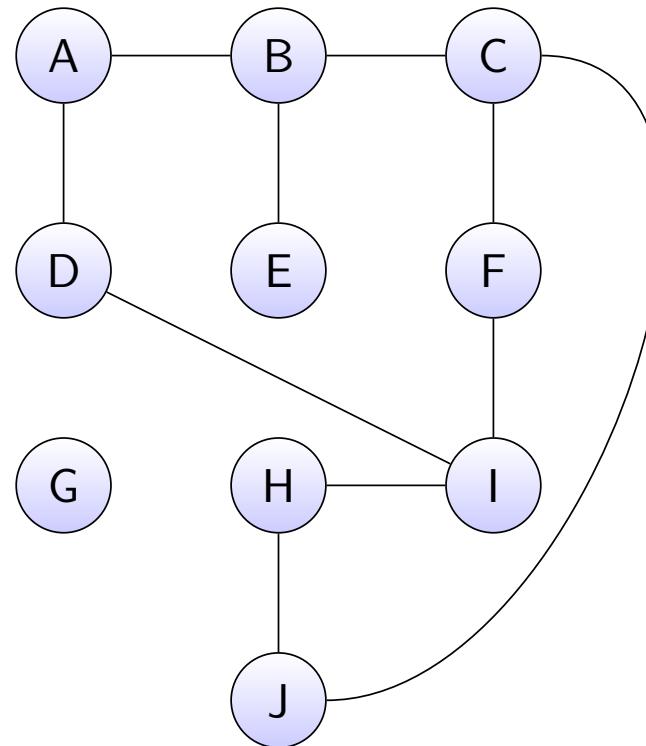
Perform a depth-first search starting in A:



# Graph Traversal



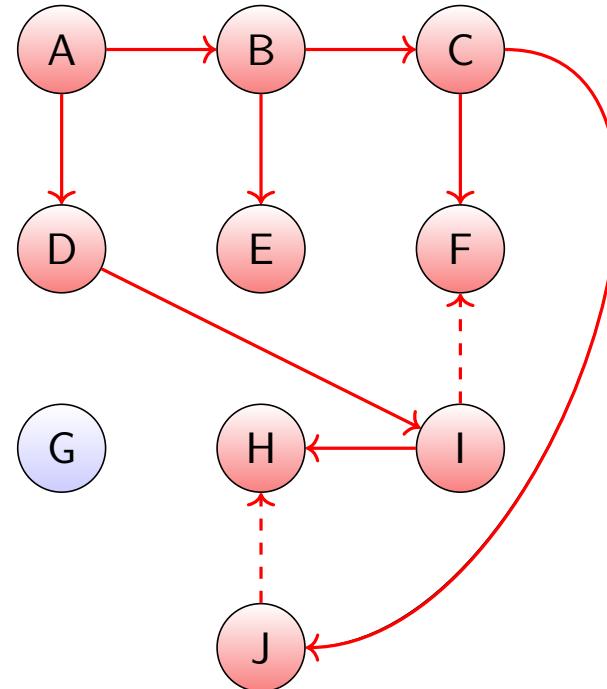
Perform a breadth-first search starting in A:



# Graph Traversal

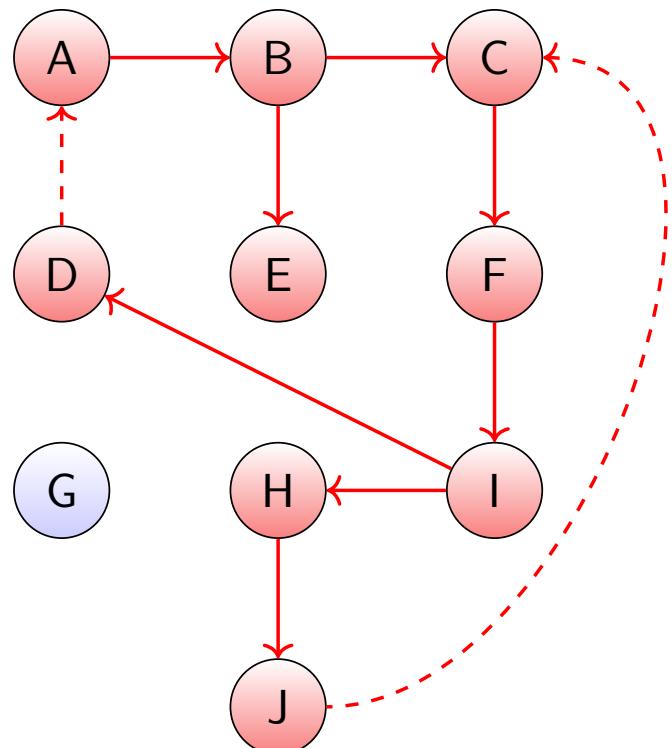


Perform a breadth-first search starting in A:

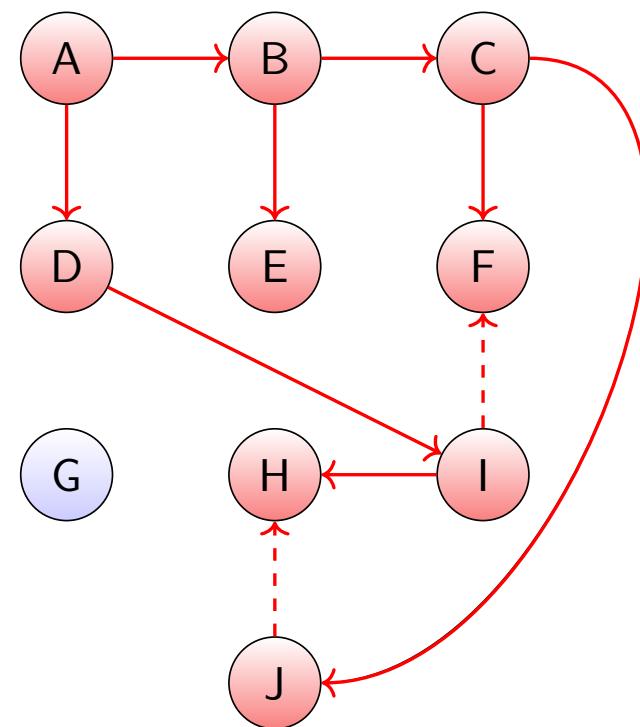


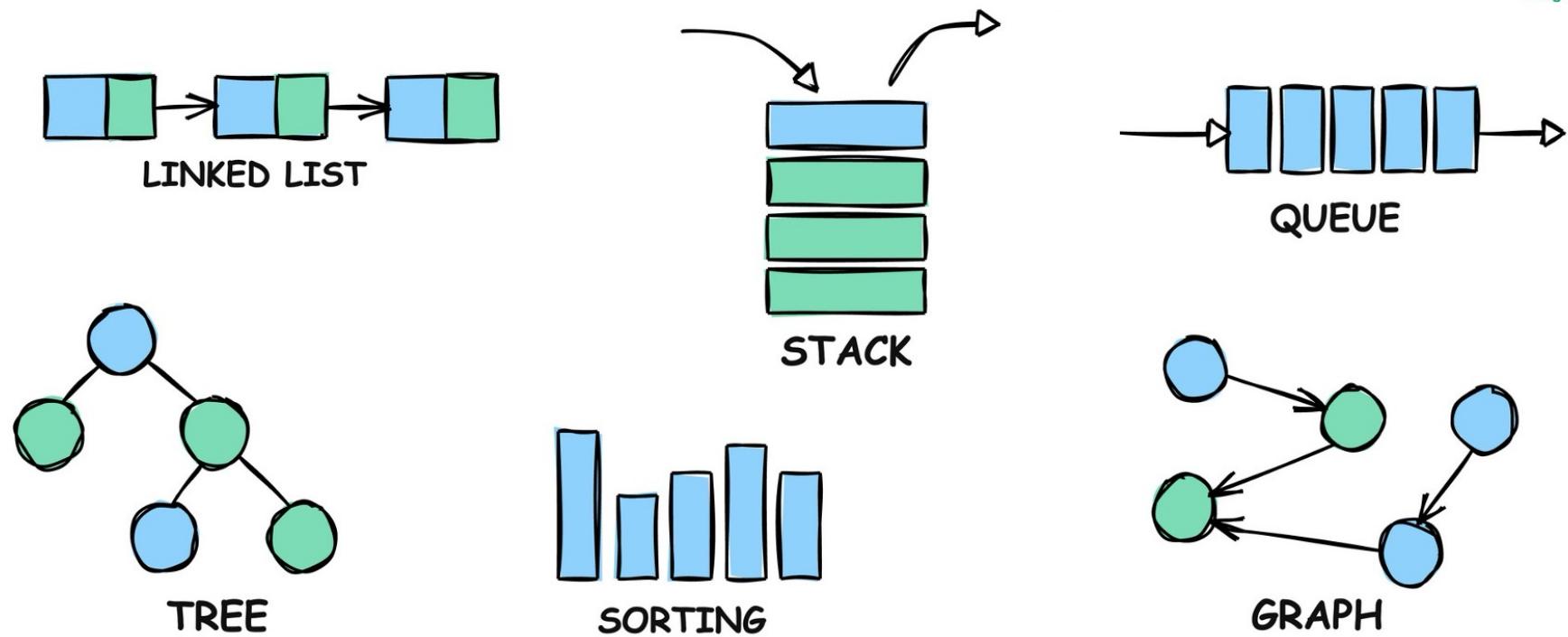
# Comparing DFS and BFS

DFS



BFS





# Algorithms & Data Structures

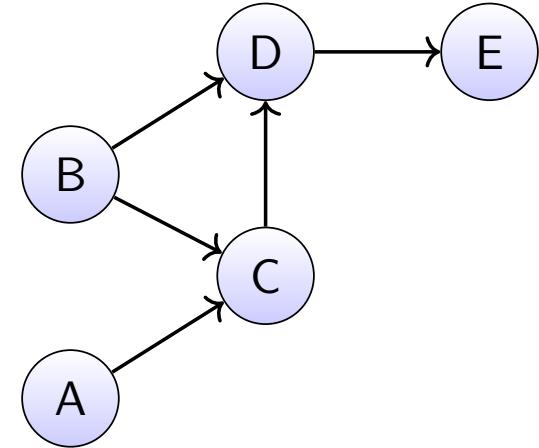
Megha Khosla, Ivo van Kreveld

# Directed Acyclic Graph

A **directed acyclic graph (DAG)** is a directed graph that has no directed cycles

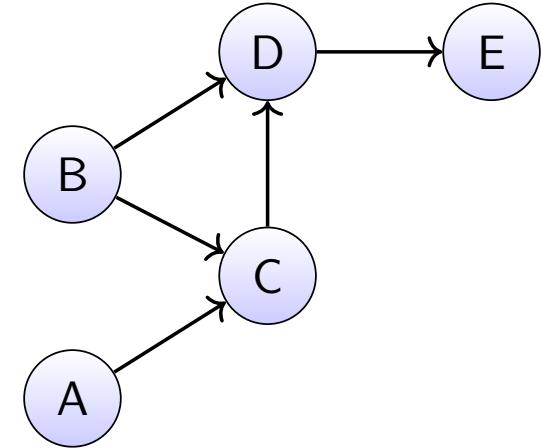


Give some examples of DAGs?



# Directed Acyclic Graph

A **directed acyclic graph (DAG)** is a directed graph that has no directed cycles



Give some examples of DAGs?

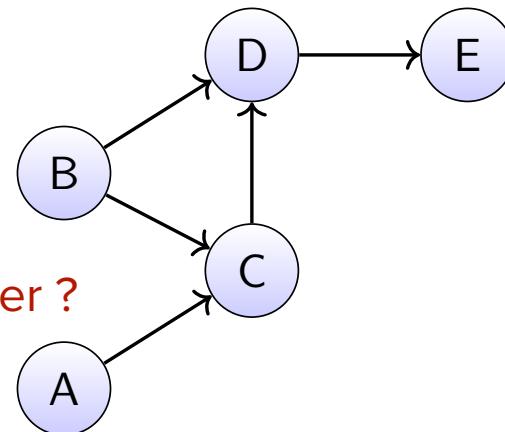
- ▶ Prerequisites between courses of an academic program
- ▶ Inheritance between classes of an object-oriented program
- ▶ Scheduling constraints between the tasks of a project

# Topological Ordering

Given a directed graph  $G = (V, E)$ , a sequence  $v_1 \dots v_n$  consisting of the vertices  $V$  is a **topological ordering** of  $G$  if:

for every edge  $(v_i, v_j) \in E$  we have  $i < j$

**Example:** a possible topological order is  $A, B, C, D, E$



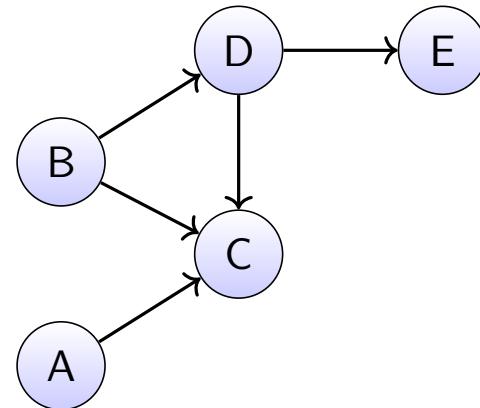
Is there another topological order?

# Topological Ordering



How many of the sequences below are a topological ordering of the graph?

- (a) A, B, D, C, E
- (b) A, B, C, D, E
- (c) B, D, E, A, C
- (d) B, C, E, A, D

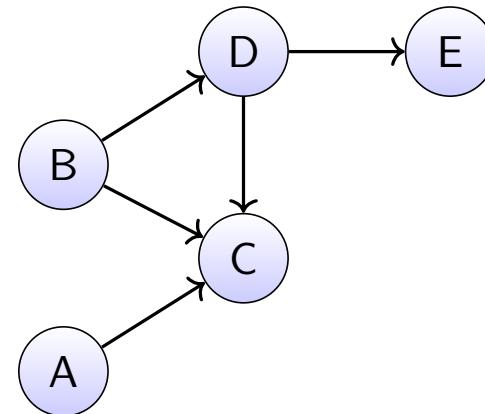


# Topological Ordering



How many of the sequences below are a topological ordering of the graph? 2

- (a) A, B, D, C, E
- (b) A, B, C, D, E
- (c) B, D, E, A, C
- (d) B, C, E, A, D

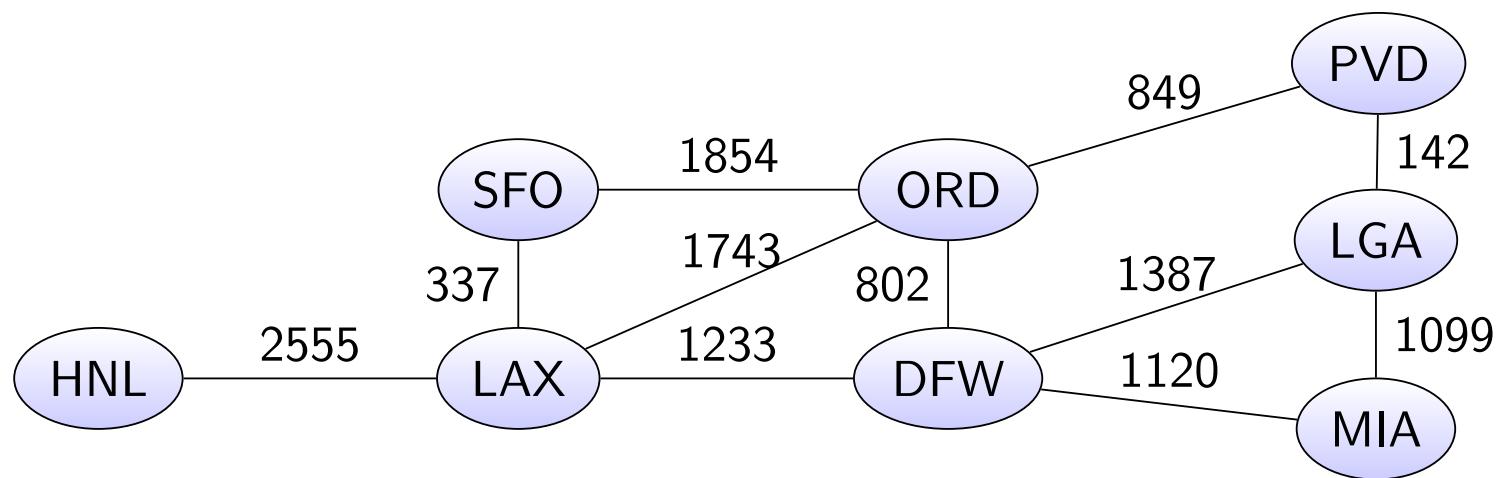


'B, D, C' should appear in that order in the topological order. Then we may the choice to insert 'A' somewhere before 'C' and 'E' somewhere after 'D'.

# Weighted Graph

A **weighted graph** is a graph in which each edge has an associated numerical value, called the **weight of the edge**

**Example:** vertices represent airports, edges represent distances

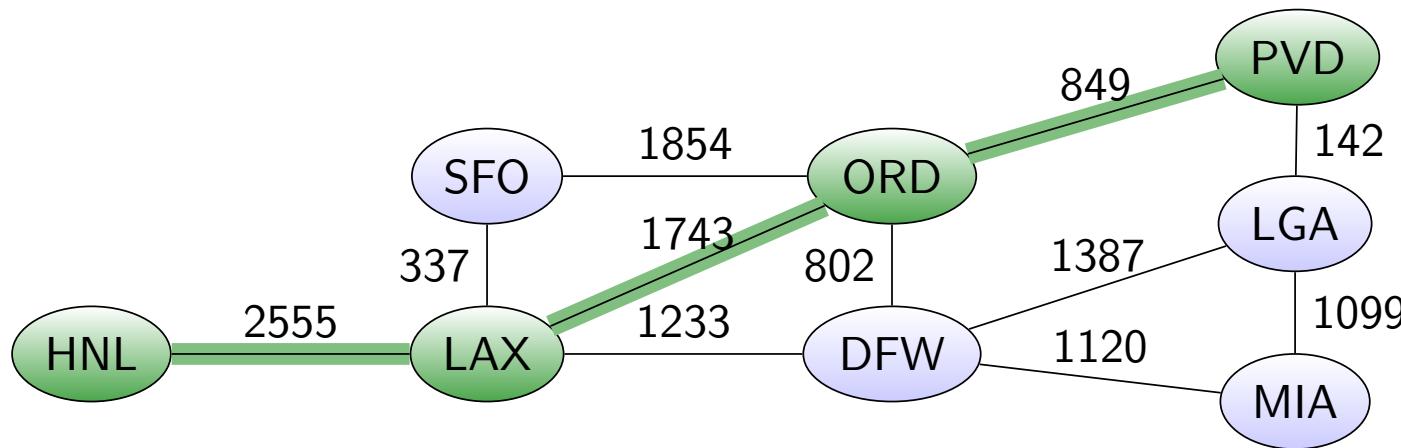


# Shortest Path

**Shortest path problem:** given a weighted graph and vertices  $u$  and  $v$ , we want to find a path of minimum weight between  $u$  and  $v$

The **weight of a path** is the sum of the weights of its edges

**Applications:** internet packet routing, flight reservations, route planners



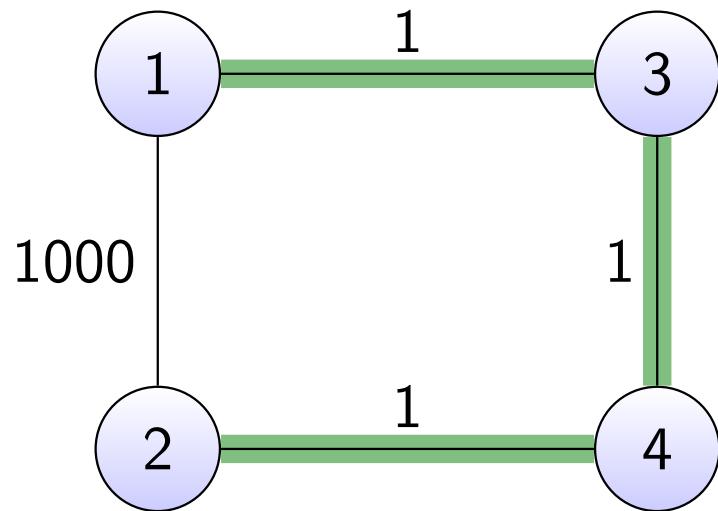
The weight of the path **HNL, LAX, ORD, PVD** is **5147**

# Breadth First Search

Is it always possible to get a shortest path using BFS?

# Breadth First Search

Is it always possible to get a shortest path using BFS? **No**



# Dijkstra's algorithm

**Input:** a weighted graph and a starting vertex  $s$

**Output:** a map  $D[v]$  that gives the weight of the shortest path between  $s$  and any vertex  $v$

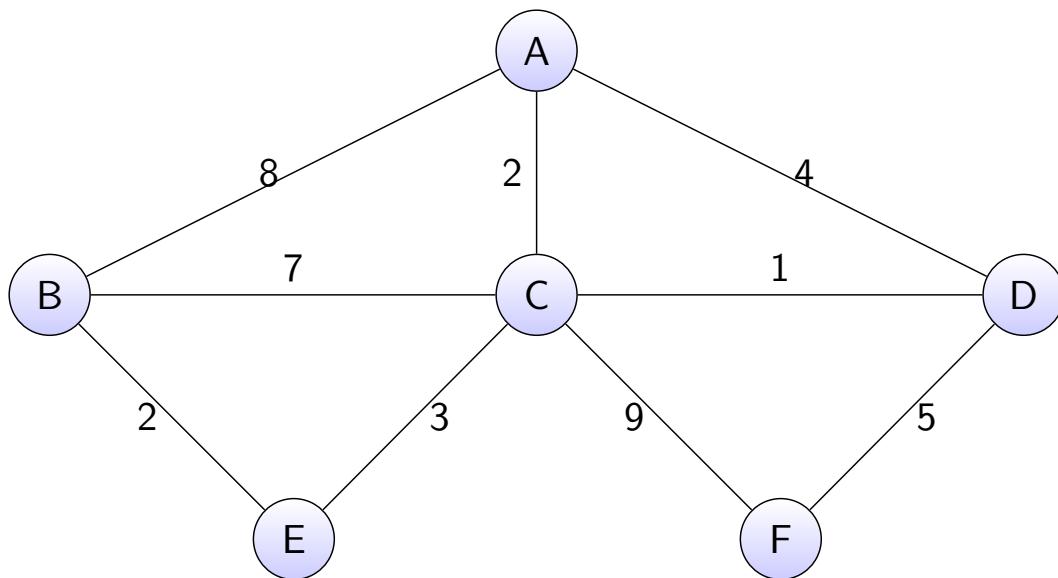
Idea of the algorithm:

- ▶ For each vertex  $v$ , keep track of:
  - ▶ a Boolean that describes whether  $v$  has been visited or not
  - ▶ a label  $D[v]$  that contains the weight of the best path we have found **so far**
- ▶ Initially no node has been visited, and  $D[s] = 0$ ,  $D[v] = \infty$  for each  $v \neq s$
- ▶ Repeat until all vertices have been visited:
  - ▶ pick unvisited vertex with least  $D[u]$ , and mark it visited
  - ▶ ‘relax’  $D[v]$  for each edge  $(u, v)$  with weight  $w(u, v)$  as follows:

$$D[v] = \begin{cases} D[u] + w(u, v) & \text{if } D[u] + w(u, v) < D[v] \\ D[v] & \text{otherwise} \end{cases}$$

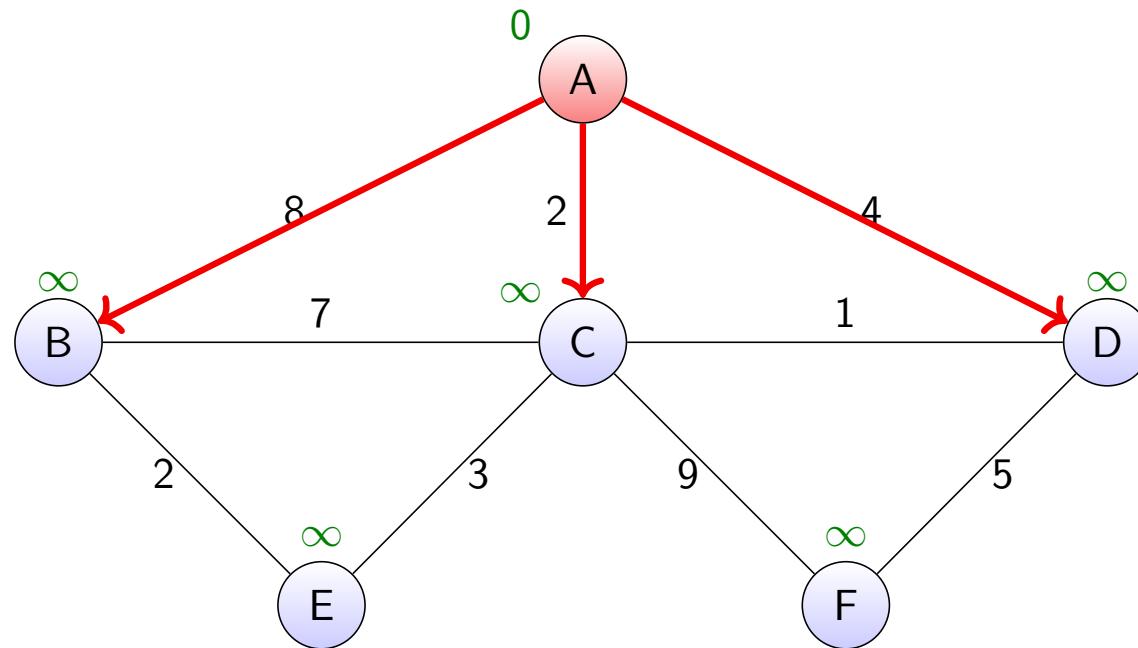
# Dijkstra's algorithm

Perform Dijkstra's algorithm with source vertex as A.



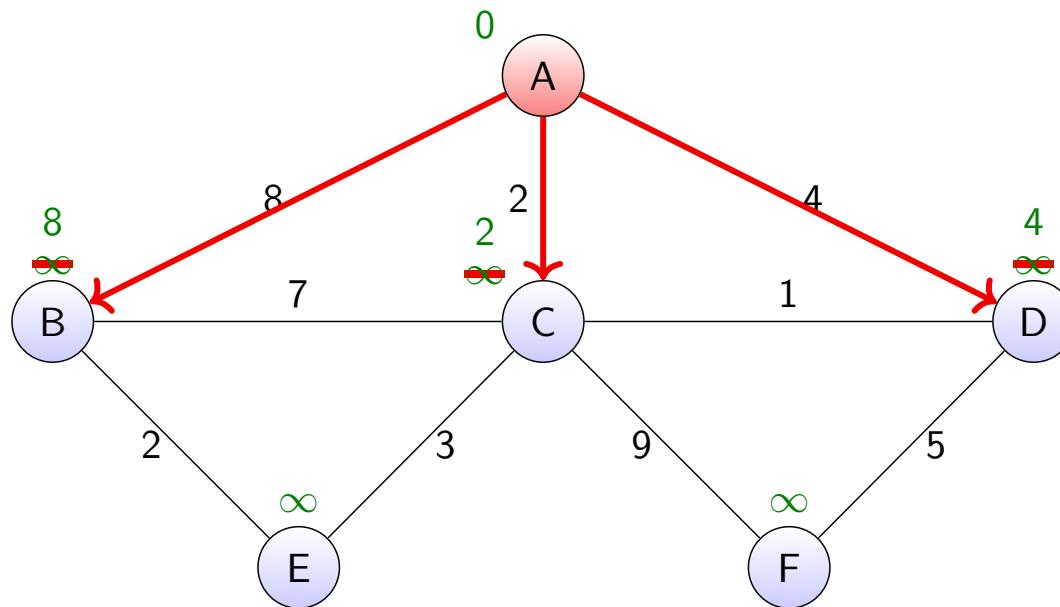
# Dijkstra's algorithm

$$D[v] = \begin{cases} D[u] + w(u, v) & \text{if } D[u] + w(u, v) < D[v] \\ D[v] & \text{otherwise} \end{cases}$$



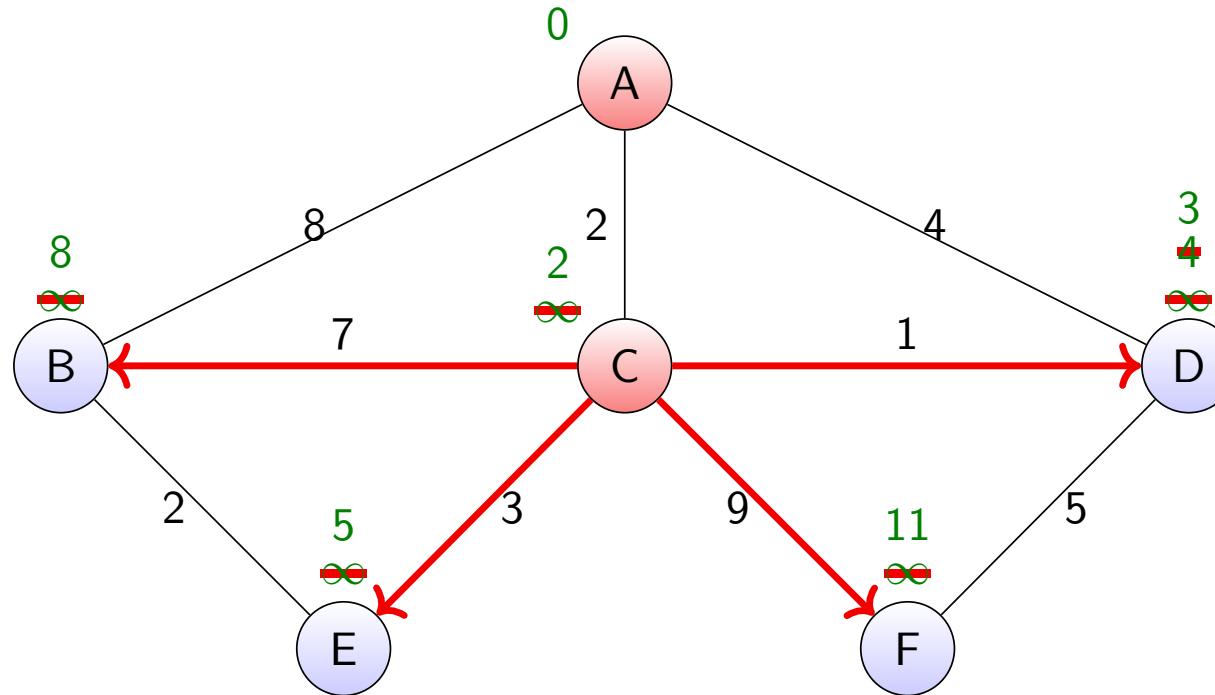
# Dijkstra's algorithm

$$D[v] = \begin{cases} D[u] + w(u, v) & \text{if } D[u] + w(u, v) < D[v] \\ D[v] & \text{otherwise} \end{cases}$$



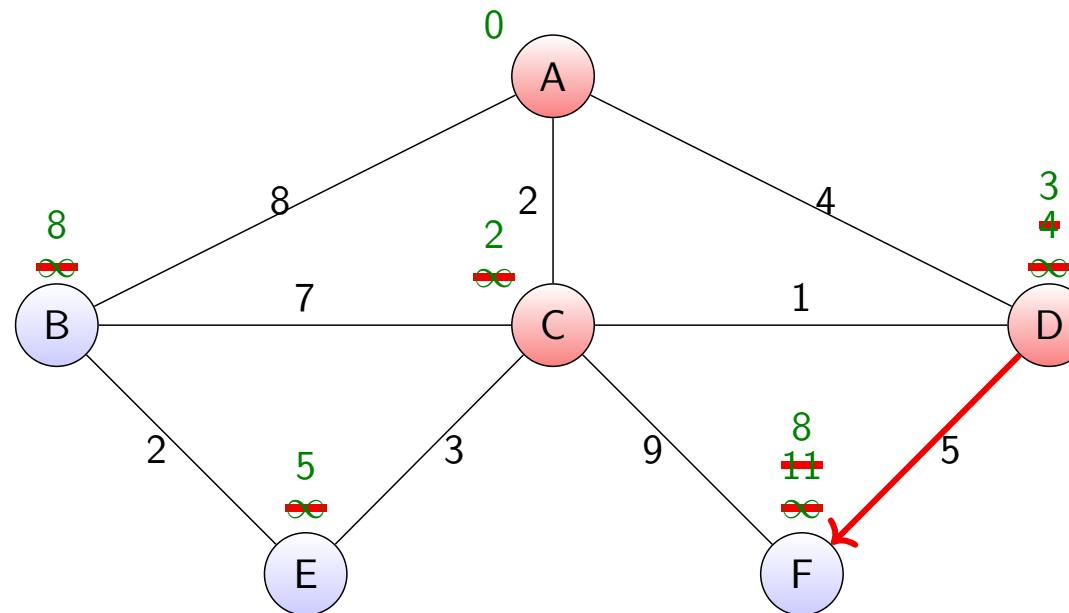
# Dijkstra's algorithm

$$D[v] = \begin{cases} D[u] + w(u, v) & \text{if } D[u] + w(u, v) < D[v] \\ D[v] & \text{otherwise} \end{cases}$$

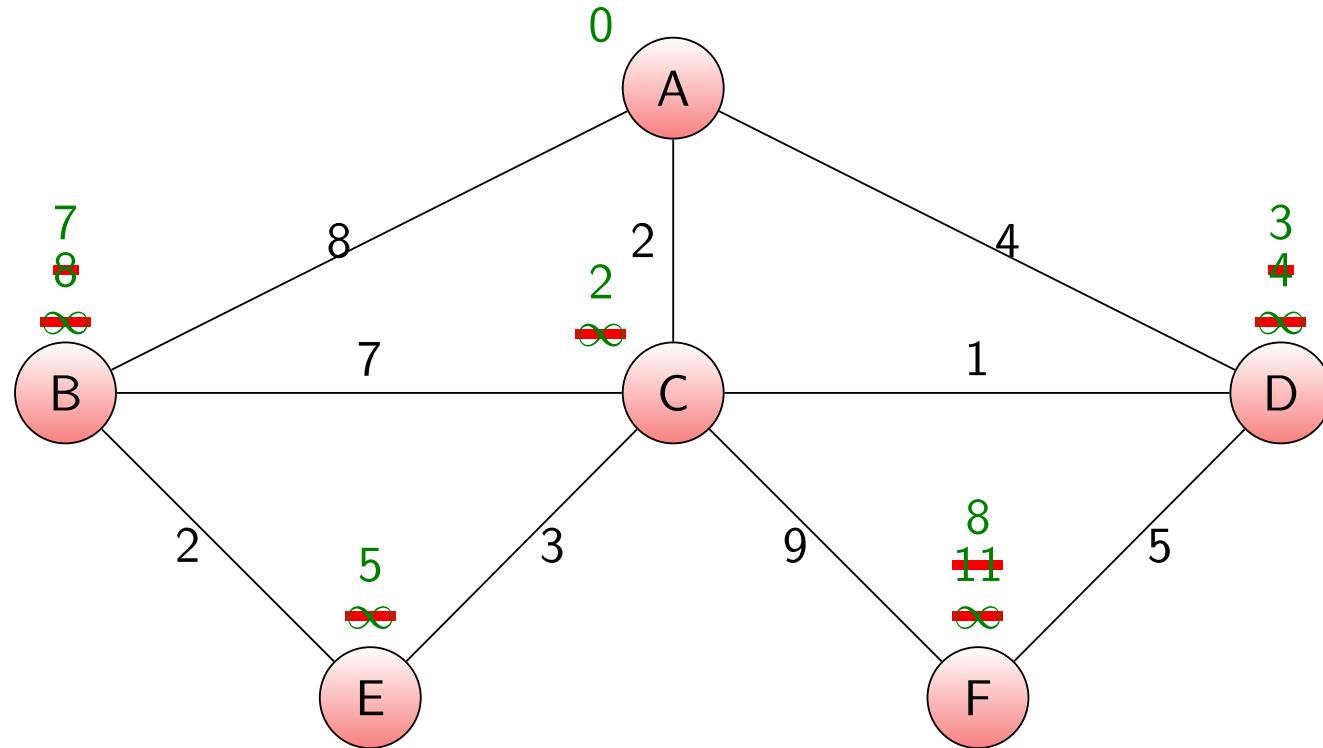


# Dijkstra's algorithm

$$D[v] = \begin{cases} D[u] + w(u, v) & \text{if } D[u] + w(u, v) < D[v] \\ D[v] & \text{otherwise} \end{cases}$$



# Dijkstra's algorithm



# Dijkstra's algorithm

What data structure should we use to pick the vertex with the least weight?

# Dijkstra's algorithm

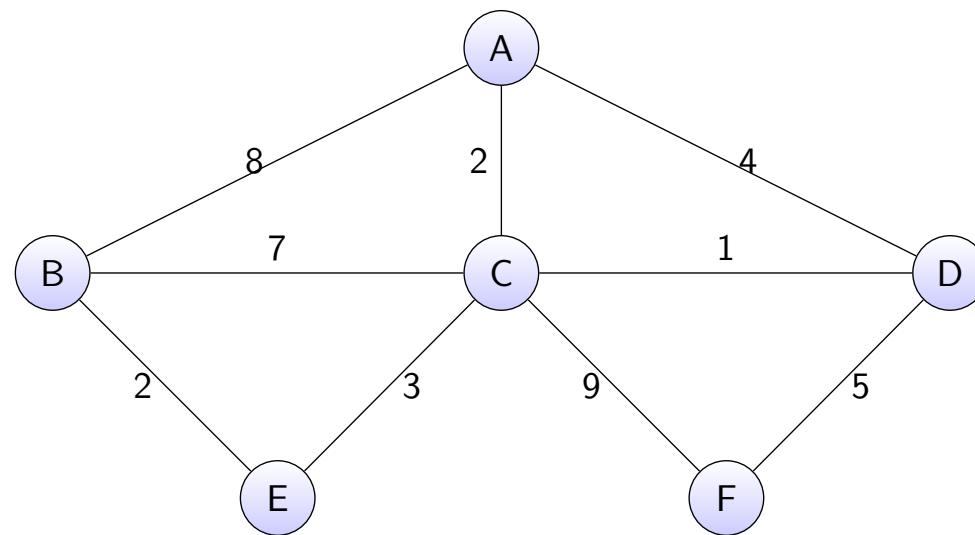
What data structure should we use to pick the vertex with the least weight?

## Adaptable Priority Queue

- We want the vertex with the lowest label
- We need to be able to update the label of a vertex

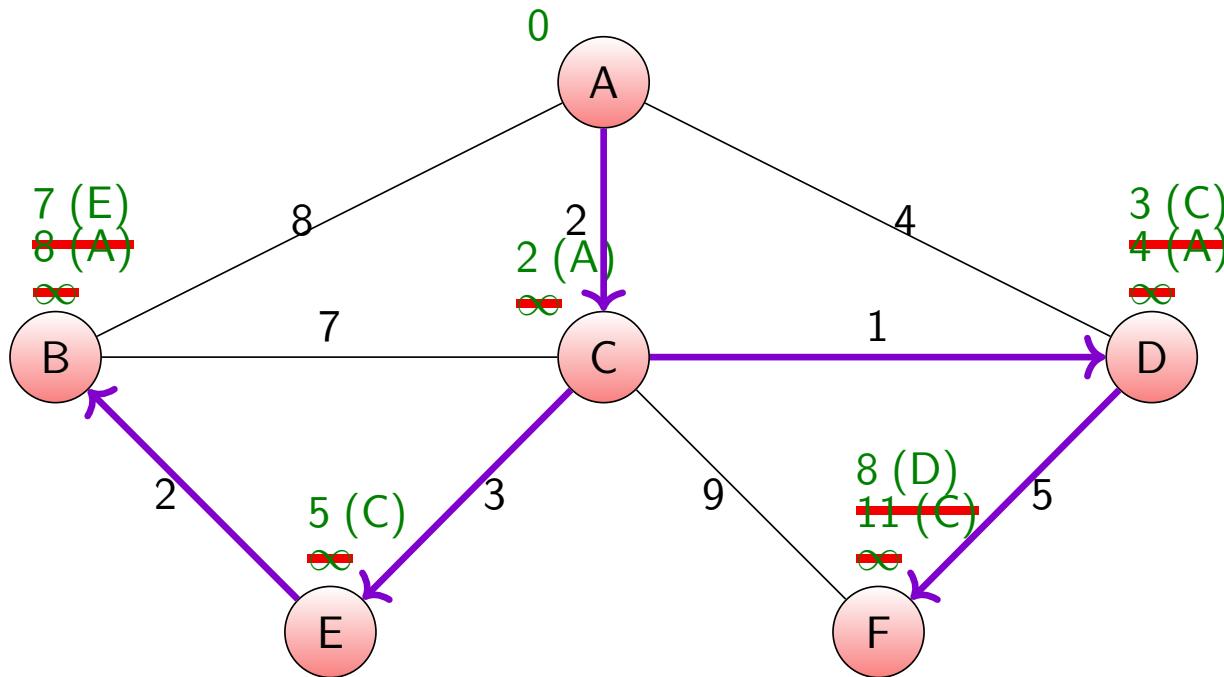
# Dijkstra's algorithm

How do we use Dijkstra's algorithm to find the shortest path between A and B?



# Dijkstra's algorithm

Shortest Path between A and B : A , C, E, B



# Minimum Spanning Tree

**Minimum spanning tree (MST) problem:** given an undirected, weighted graph  $G$ , we are interested in finding a spanning tree  $T$  of  $G$  that minimizes the sum

$$w(T) \triangleq \sum_{(u,v) \in T} w(u, v)$$



Suppose we wish to connect all the computers in a new office building using the least amount of cable. We can represent this as an MST problem. What will be the vertices, edges and weight of the graph?

# Minimum Spanning Tree

**Minimum spanning tree (MST) problem:** given an undirected, weighted graph  $G$ , we are interested in finding a spanning tree  $T$  of  $G$  that minimizes the sum

$$w(T) \triangleq \sum_{(u,v) \in T} w(u, v)$$



Suppose we wish to connect all the computers in a new office building using the least amount of cable. We can represent this as an MST problem. What will be the vertices, edges and weight of the graph?

- ▶ **Vertices:** computers
- ▶ **Edges:** all pairs of computers
- ▶ **Weight:** amount of cable needed to connect two computers

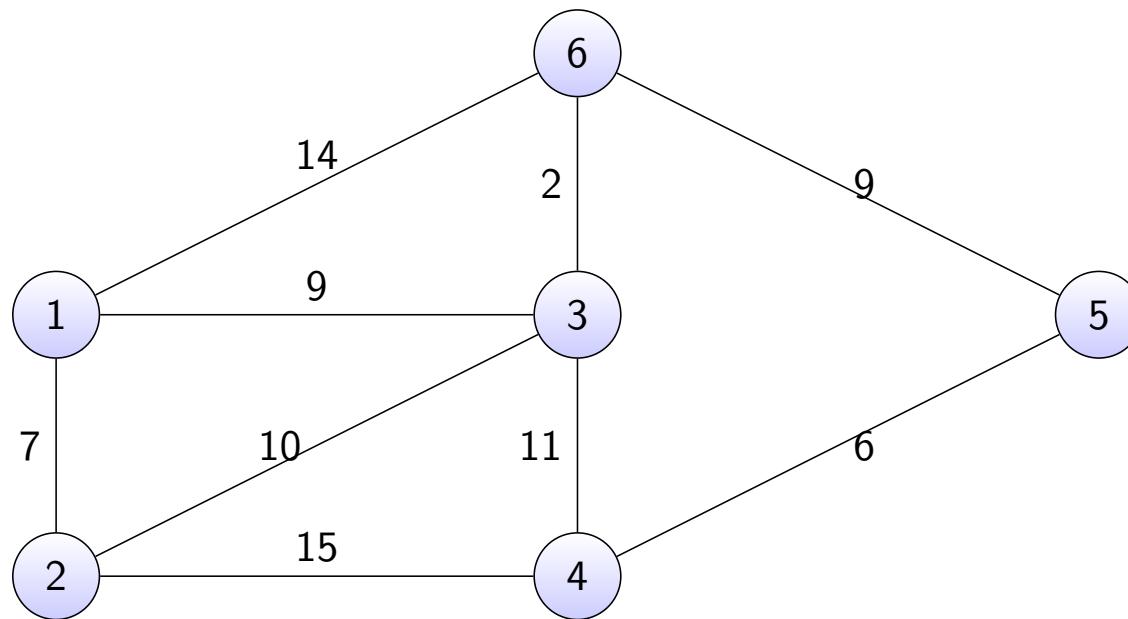
# Kruskal's Algorithm

## Idea:

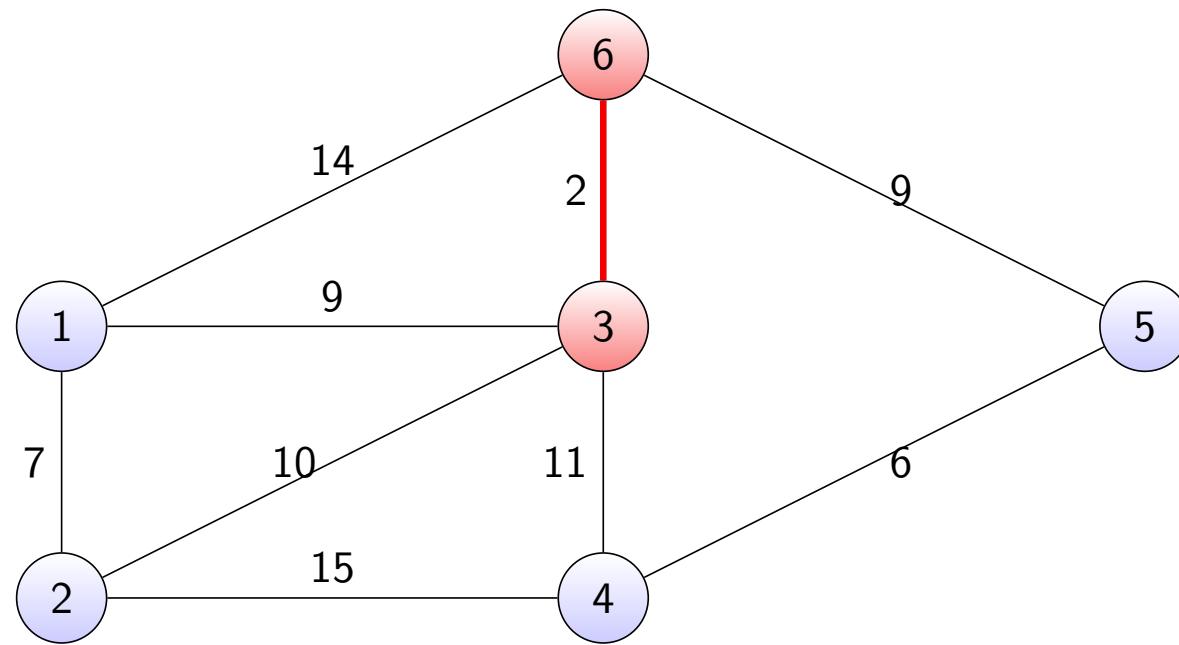
1. Maintain a partition of the vertices into **clusters**  
Initially, **each vertex forms a single cluster**
2. Pick an edge  $(u, v)$  with smallest weight that is not yet part of any cluster
3. If the **clusters of  $u$  and  $v$  are different**: **merge** both using the edge  $(u, v)$
4. Otherwise: remove the edge
5. Repeat until we have obtained a single cluster

# Kruskal's Algorithm

Find the minimum spanning tree using Kruskal's algorithm.



# Kruskal's Algorithm



# Kruskal's Algorithm

