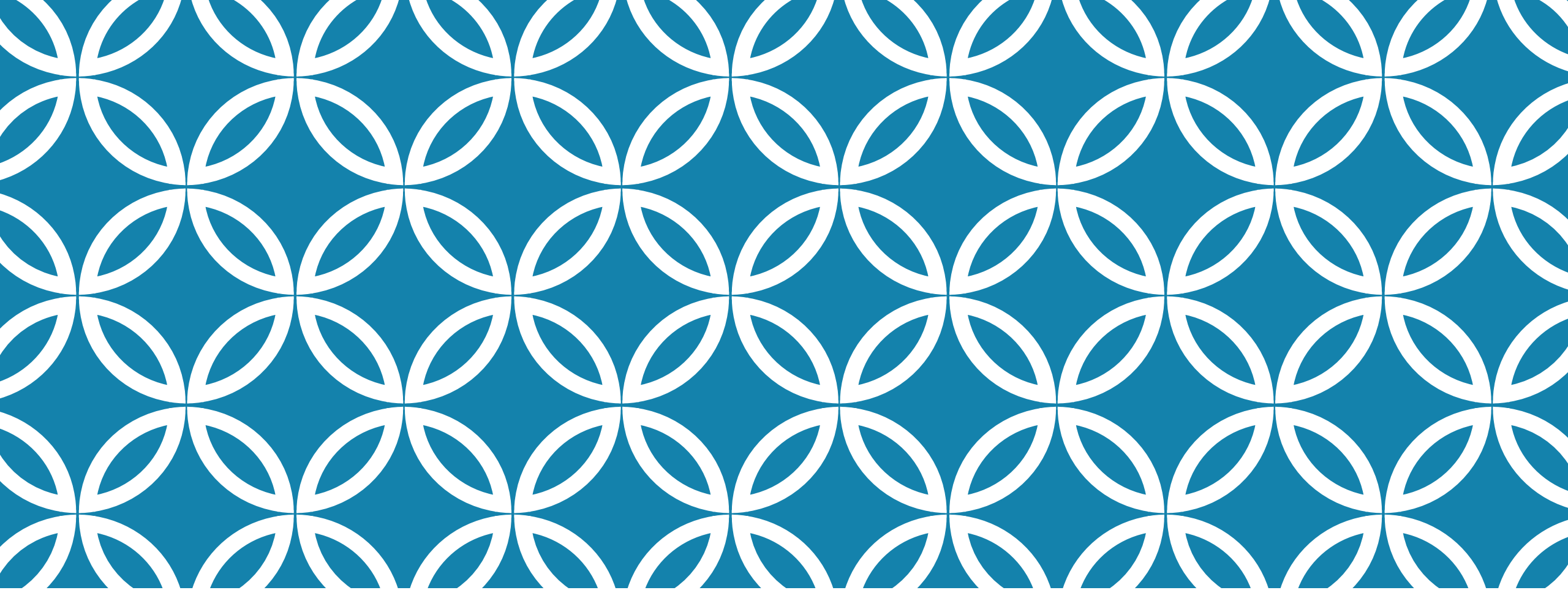


LECTURE 20: SUMMARY AND CLOSING LECTURE

Harold Soh
harold@comp.nus.edu.sg

ADMINISTRATIVE ISSUES

TAs are grading your assignments

- All scores (except late submissions) will be online by next Monday (18th).
- **Please verify** (1 participation point!)

Problem Set 5 was due yesterday!

- **All** Problem Sets should be submitted by midnight Friday (Nov 15th).
- **Email me and your DG TA** if you make a late submission!
- Late Submissions
  - 24 hours: 20% penalty
  - 2 weeks: 40% penalty
  - Last Second of Last Day of class: 60% penalty

No lecture tomorrow. DGs and Tutorials are still on.

FINAL EXAM

4 DEC 2018 (Morning, 9-11am)

2 hours

44 Questions (120 points)

All multiple choice

- Bring pencils to shade in the form

Open-book exam

QUESTIONS?

Poll Everywhere

https://bit.ly/2LvG9bq

PATH TO MASTERY / COURSE STRUCTURE

Intro to Graphs

Searching on Graphs! - Depth/Breadth-First

Topological Sorting

Single-Source Shortest Paths Dijkstra, Bellman-Ford, All-Pairs Shortest Paths

Minimum Spanning Trees (MST)

Bloom Filters, Cuckoo Hashing

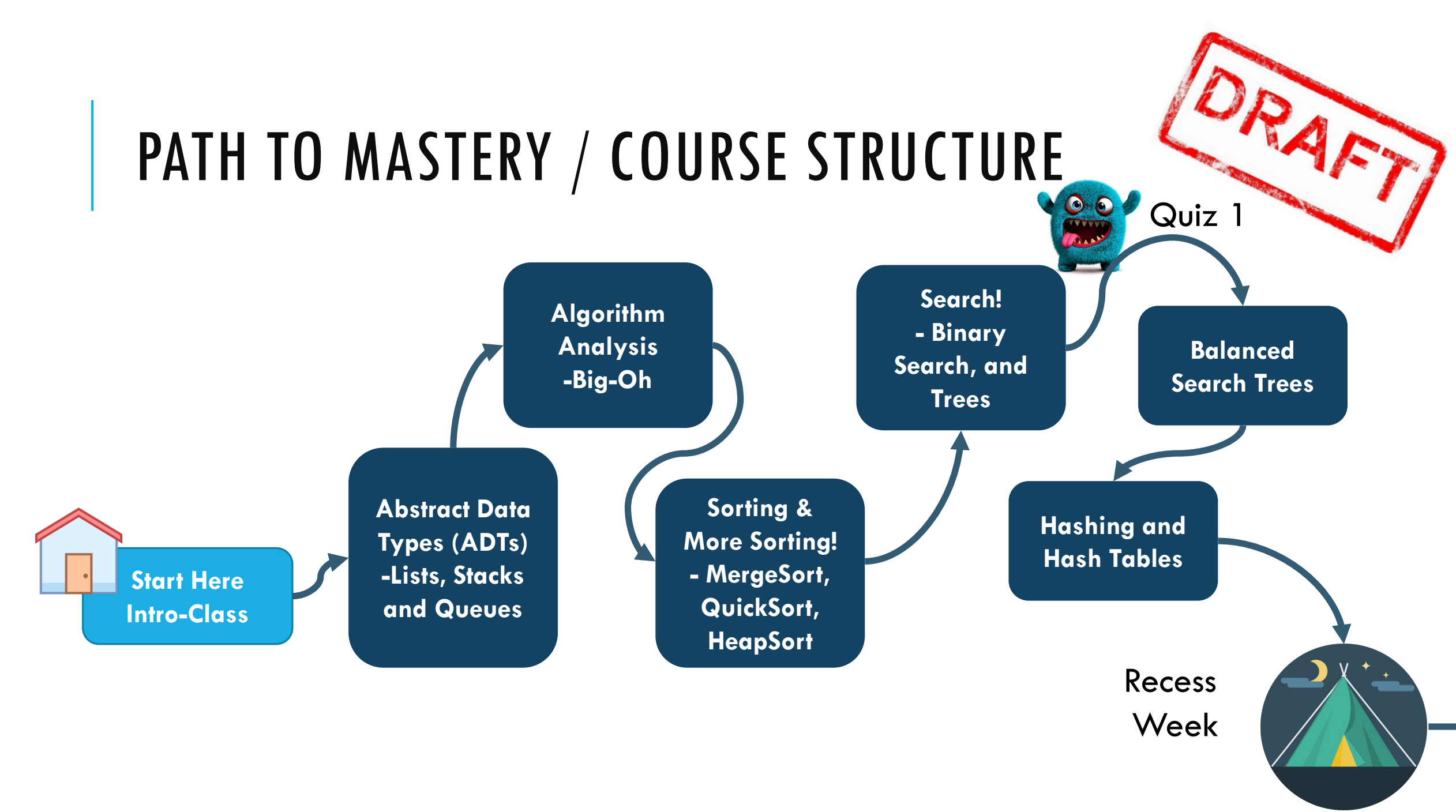
YOU ARE HERE

Quiz 2

Quiz 3

Final Boss!

PATH TO MASTERY / COURSE STRUCTURE



CS2040S GAME SHOW!!

4 teams

Answer on polleverywhere

Let's go there now!

QUESTION

CS2040 is designed to develop which of these skills?

A. Monkeying around.
B. Problem solving with algorithmic thinking.
C. Ultra-Competitive programming.
D. Eating cake.

QUESTION

CS2040 is designed to develop which of these skills?

A. Monkeying around.

B. Problem solving with algorithmic thinking.

C. Ultra-Competitive programming.

D. Eating cake.

COMPUTER SCIENTISTS HAVE CHANGED THE WORLD…

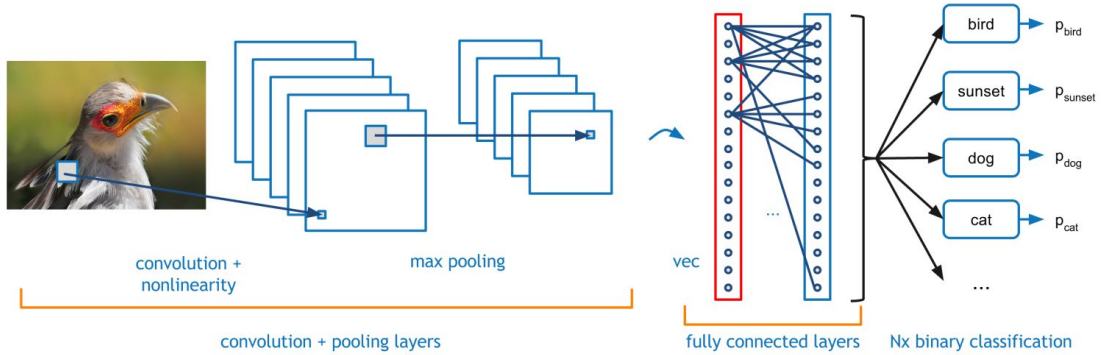
. . . BY SOLVING PROBLEMS

CS2040S is about **solving problems**.

It prepares you for what lies ahead…

Search

Image Recognition



QUESTION

People are not perfect. Code is complex.

How can we prevent them from making stupid mistakes when working with your code?

A. Use ADTs.
B. Specify the invariants.
C. Hire smarter people (or monkeys).
D. Develop AI that does all the work for us.

QUESTION

People are not perfect. Code is complex.

How can we prevent them from making stupid mistakes when working with your code?

A. **Use ADTs.**
B. Specify the invariants.
C. Hire smarter people (or monkeys).
D. Develop AI that does all the work for us.

ABSTRACT DATA TYPES (ADTs)

Define **behavior**, *not* internal operation.

Interface `SimpleStore`
`add(x)`

- **Description:** adds x to the stored items.
- **Pre:** x is the item to be added
- **Post:** x is added to the store
- **Computational cost:**
  - takes constant time
  - adds constant space (memory).

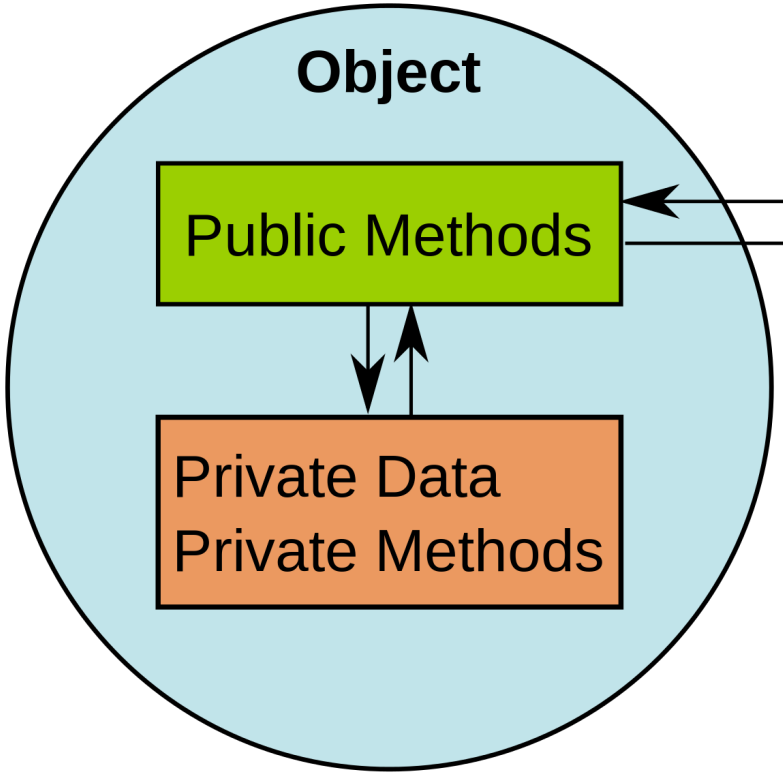
V.S.

Class `SimpleStore`
`add(x)`

- adds x to the stored items.
- if arr has already reached its current maximum size,
  - create a new temporary array
  - copy the original array over
  - add the x
  - update the end marker
- and so on ....

ENCAPSULATION AND INFORMATION HIDING

Implementer



User

[Image from Wikipedia commons]

QUESTION

Which ADT can I use to transform a recursive function into an iterative function?

A. Stack
B. Queue
C. Priority Queue
D. Hash Table

QUESTION

Which ADT can I use to transform a recursive function into an iterative function?

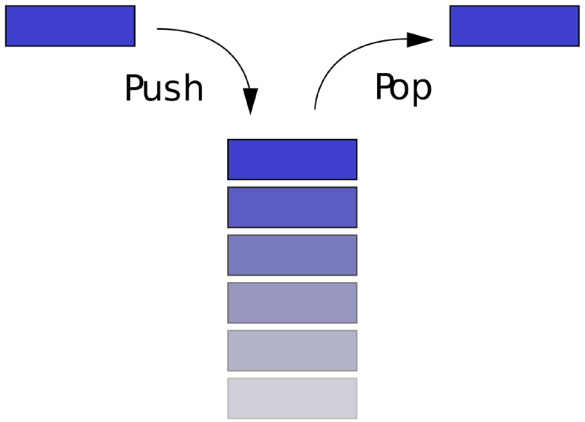
A. **Stack**
B. Queue
C. Priority Queue
D. Hash Table

STACK

Stack Operations:

- push(x)
- peek()
- pop()

Last-in-First-Out (LIFO)



STACKS ARE VERY USEFUL!

Implement Recursion

Expression Evaluation: (x+2)*(2/45) + 2

Depth-First-Search (later in the semester!)

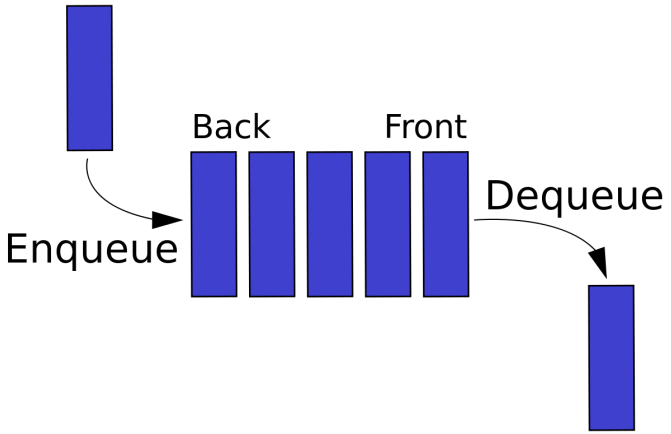
and others…

QUEUE

Queue Operations:

- enqueue(x)
- peek()
- dequeue()

First-in-First-Out (FIFO)



QUESTION

I have a word with n characters stored in a singly linked list. If I can only use O(1) space, detecting if the word is a palindrome requires:

A. $O(1)$ time
B. $O(n)$ time
C. $O(n \log n)$ time
D. $O(n^2)$ time

QUESTION

I have a word with n characters stored in a singly linked list. If I can only use O(1) space, detecting if the word is a palindrome requires:

A. $O(1)$ time

B. $O(n)$ time

C. $O(n \log n)$ time

D. $O(n^2)$ time

DETECTING IF A LINKED LIST IS A PALINDROME

Key idea:

Iterate to middle

Reverse second half of linked list

Iterate from beginning and middle, comparing the characters.

Time Complexity: $O(n)$

REVERSING A SINGLY LINKED LIST PSEUDOCODE

```
Node reversed_list = null
Node current = head
while current is not null
    Node next = current.next
    current.next = reversed_list
    reversed_list = current
    current = next
head = reversed_list
```

QUESTION

What's the time complexity of the following piece of code?

```
int a = 0;
for (int i=0; i<n; i++) {
    a = a + 2;
    A[i] = a;
}
```

RUNNING TIME

Remember when I had to explain this step by step?

Number of primitive instructions / "steps" executed.

Code	Cost	Times
`int a = 0;`	c_1	1
`for (int i=0; i<n; i++) {`	c_2	n
`  a = a + 2;`	c_3	n
`  A[i] = a;`	c_4	n
`}`		

$$T(n) = c_1 + nc_2 + nc_3 + nc_4 = cn + c_1$$

QUESTION

The following functions requires:

```
function testme(n)
  if (n <= 2)
    return n
  return 1 + testme(n/2)
```

A. $O(1)$ time
B. $O(\log n)$ time
C. $O(n)$ time
D. $O(n \log n)$ time

QUESTION

The following functions requires:

```
function testme(n)
  if (n <= 2)
    return n
  return 1 + testme(n/2)
```

A. $O(1)$ time

B. **$O(\log n)$ time**

C. $O(n)$ time

D. $O(n \log n)$ time

QUESTION

True or False: two functions take $O(n^2)$ time to run. I can choose either in practice.

A. True

B. False

QUESTION

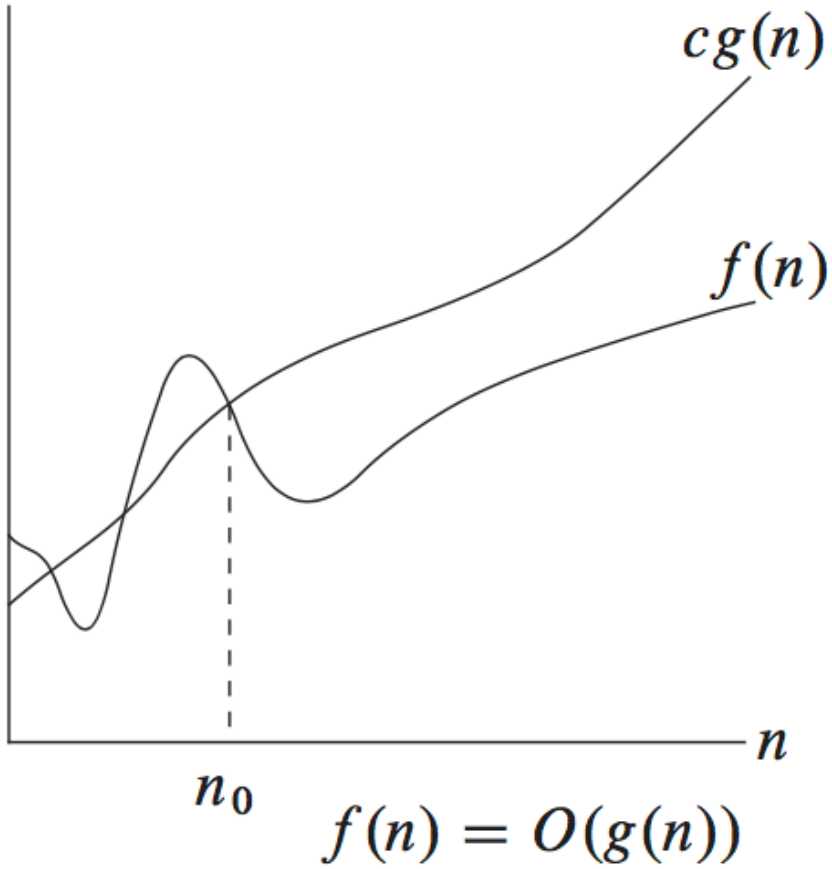
True or False: two functions take $O(n^2)$ time to run. I can choose either in practice.

A. True

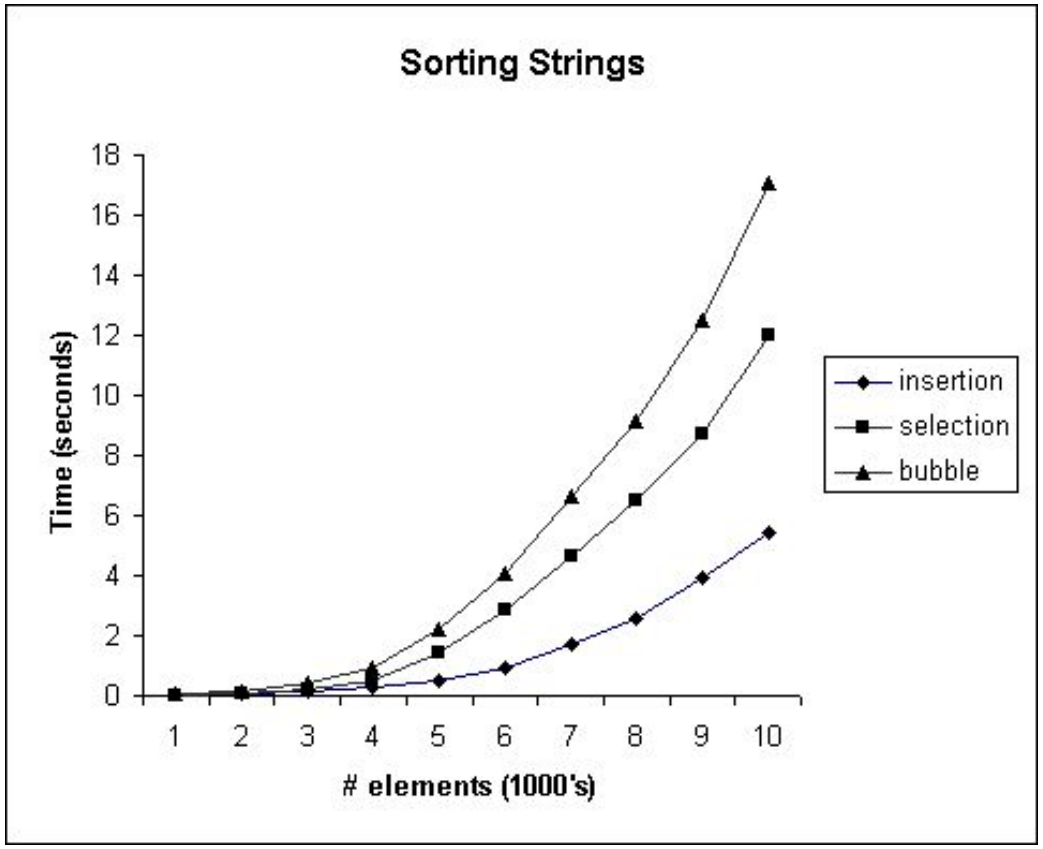
B. **False**

BIG-OH $O(g(n))$

$O(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$



SOME PRACTICE WITH BIG-OH: LIMITATIONS



[Bubble Sort: An Archaeological Algorithmic Analysis, Owen Astrachan, cs.duke.edu]

QUESTION

$T(n) = T\left(n - \frac{n}{4}\right) + T\left(\frac{n}{4}\right) + O(n)$ is bounded asymptotically by:

A. $O(1)$
B. $O(n)$
C. $O(n^2)$
D. $O(n \log n)$

QUESTION

$T(n) = T\left(n - \frac{n}{4}\right) + T\left(\frac{n}{4}\right) + O(n)$ is bounded asymptotically by:

A. $O(1)$

B. $O(n)$

C. $O(n^2)$

D. $\boldsymbol{O(n \log n)}$

QUESTION

Which of these quotes is not by a famous computer scientist?

A. Simplicity is a prerequisite for reliability
B. The use of COBOL cripples the mind
C. There should be no such thing as boring mathematics
D. Make things as simple as possible, but not simpler
E. It is practically impossible to teach good programming to students that have prior experience to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration.

EDSGER W. DIJKSTRA

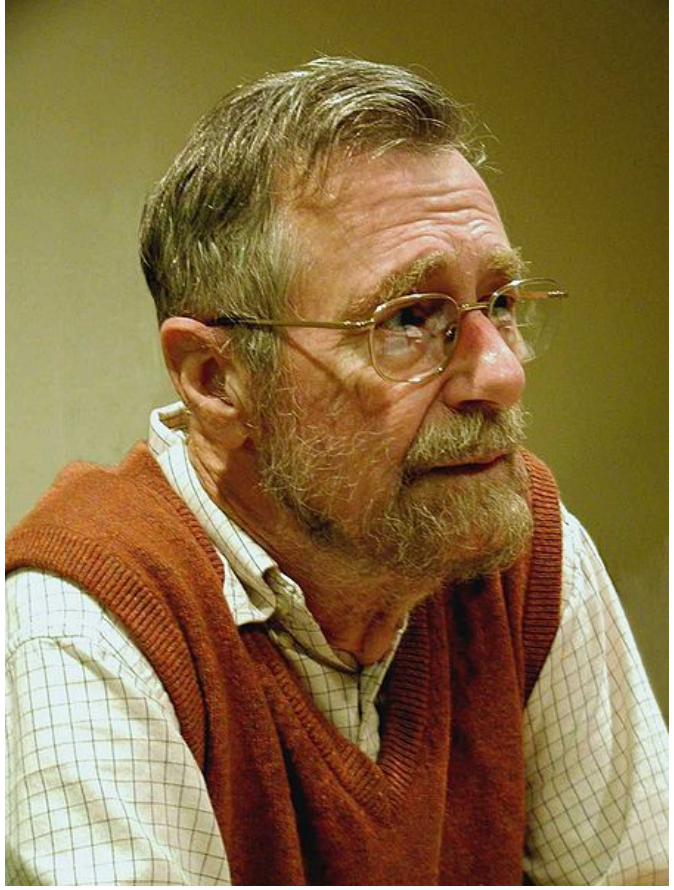
"Computer science is no more about computers than astronomy is about telescopes."

"The question of whether a computer can think is no more interesting than the question of whether a submarine can swim."

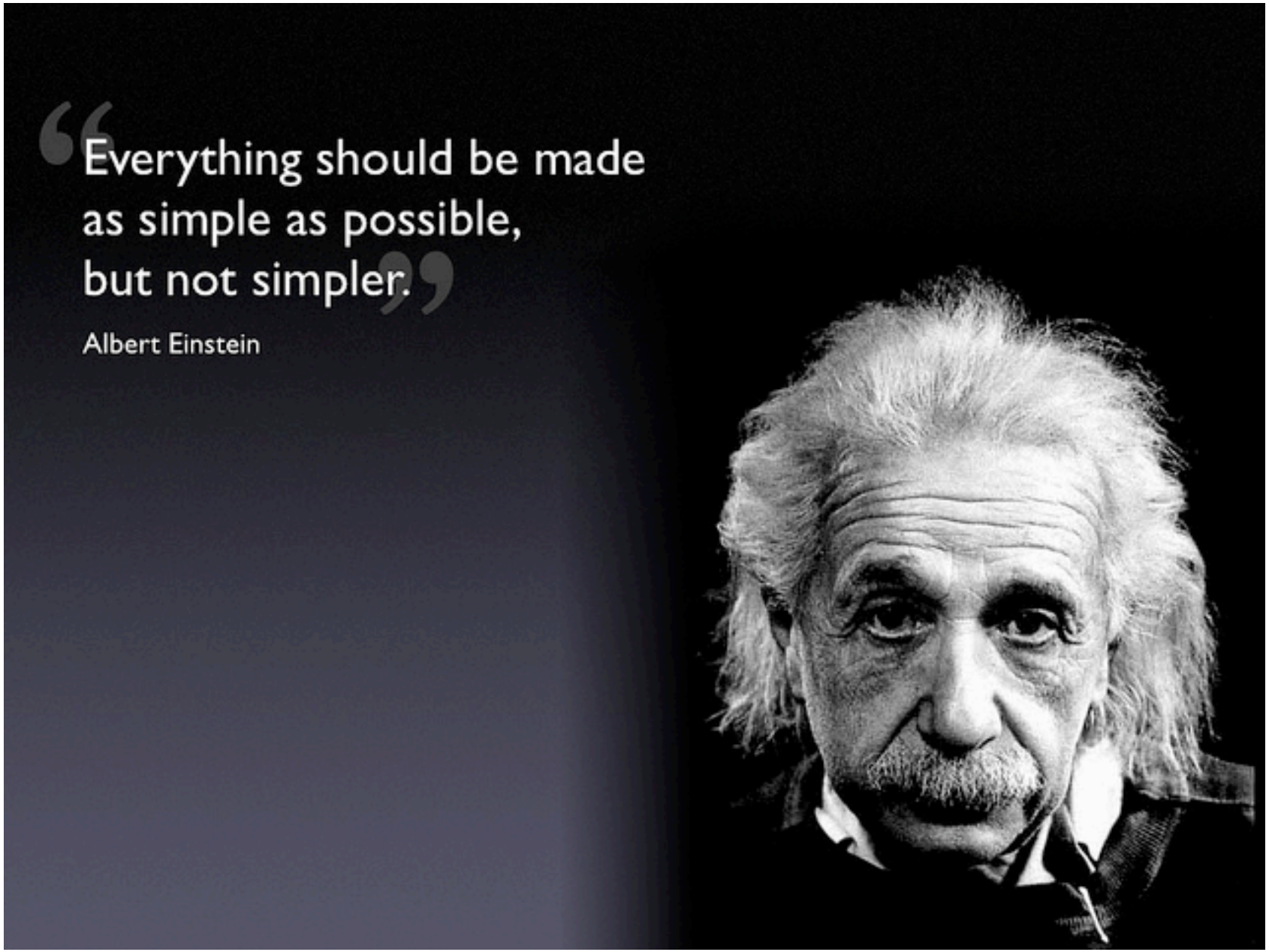
"There should be no such thing as boring mathematics."

"Elegance is not a dispensable luxury but a factor that decides between success and failure."

"Simplicity is prerequisite for reliability."



1930-2002


“Everything should be made as simple as possible, but not simpler.”
Albert Einstein

QUESTION

Naruto has a large tribe: 2^{20} members. After sorting all of them by height, he wants find a family member that is exactly 3.14159 feet tall. What algorithm should he use?

A. Mergesort
B. k-way Merge
C. Binary Search
D. 3-way partition
E. Quickselect

QUESTION

Naruto has a large tribe: 2^{20} members. After sorting all of them by height, he wants find a family member that is exactly 3.14159 feet tall. What algorithm should he use?

A. Mergesort

B. k-way Merge

C. Binary Search

D. 3-way partition

E. Quickselect

PSEUDOCODE FOR BINARY SEARCH

```
function binarySearch (A, key, n)
        low = 0
        high = n - 1
        while low <= high
                mid = (low + high )/2
                if key < A[mid] then
                        high = mid - 1
                else if key > A[mid]
                        low = mid + 1
                else
                        return mid
        return not_found
```

Which line can cause problems when used on real systems?

QUESTION

I have to sort a large array but on a embedded system with very little memory. I can only use Quicksort or Mergesort: which should I use?

A. Mergesort

B. Quicksort

QUESTION

I have to sort a large array but on a embedded system with very little memory. I can only use Quicksort or Mergesort: which should I use?

A. Mergesort

B. **Quicksort**

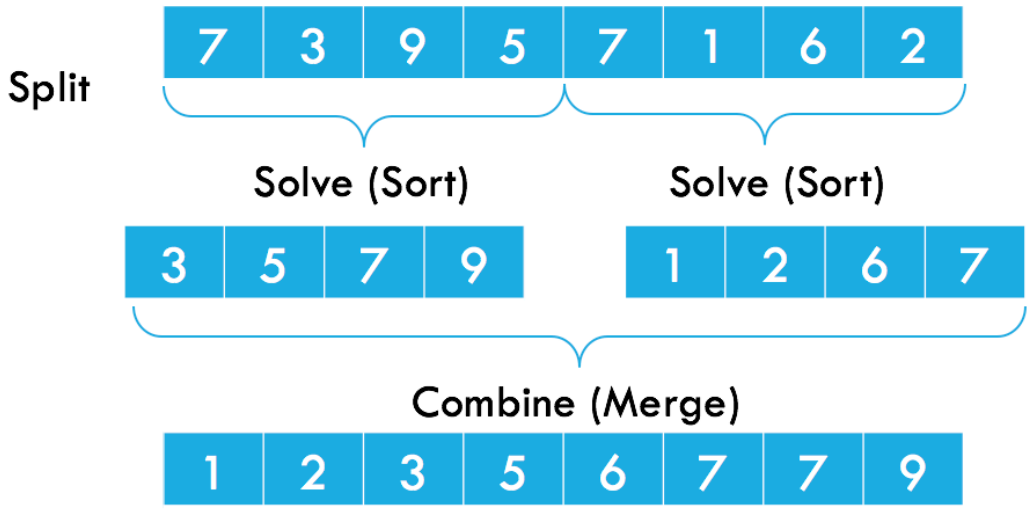
Sorting algorithms: summary

	inplace?	stable?	best	average	worst	remarks
selection	✔		$\frac{1}{2} n^2$	$\frac{1}{2} n^2$	$\frac{1}{2} n^2$	n exchanges
insertion	✔	✔	n	$\frac{1}{4} n^2$	$\frac{1}{2} n^2$	use for small n or partially ordered
shell	✔		$n \log_3 n$	?	$c\, n^{3/2}$	tight code; subquadratic
merge		✔	$\frac{1}{2} n \lg n$	$n \lg n$	$n \lg n$	$n \log n$ guarantee; stable
timsort		✔	n	$n \lg n$	$n \lg n$	improves mergesort when preexisting order
quick	✔		$n \lg n$	$2 n \ln n$	$\frac{1}{2} n^2$	$n \log n$ probabilistic guarantee; fastest in practice
3-way quick	✔		n	$2 n \ln n$	$\frac{1}{2} n^2$	improves quicksort when duplicate keys
heap	✔		$3 n$	$2 n \lg n$	$2 n \lg n$	$n \log n$ guarantee; in-place
?	✔	✔	n	$n \lg n$	$n \lg n$	holy sorting grail

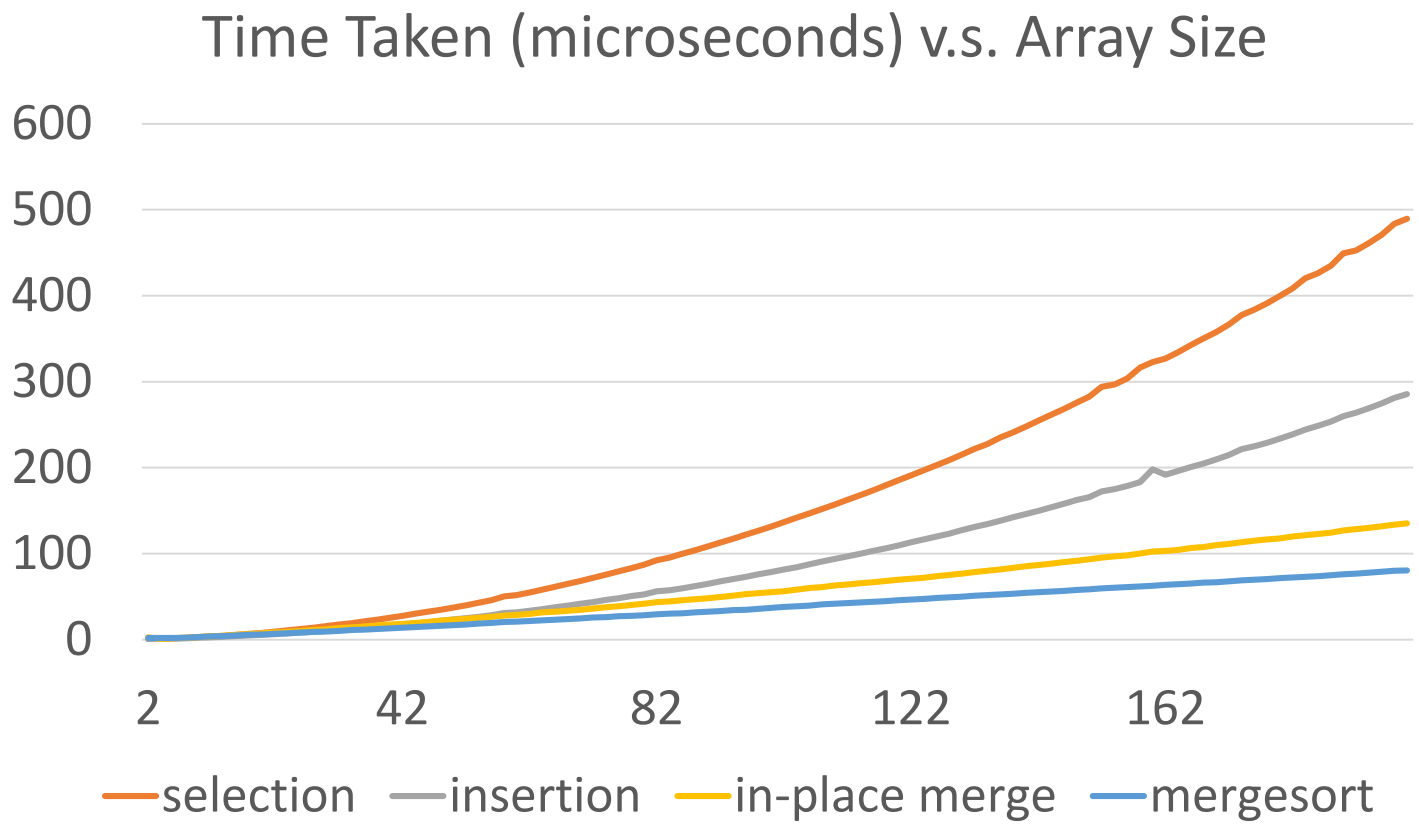
From Sedgewick and Wayne's slides
http://algs4.cs.princeton.edu/lectures/24PriorityQueues.pdf

MERGESORT: THE ALGORITHM

```
function mergeSort(A, low, high)
    if low < high
        mid = (high + low)/2
        mergeSort(A, low, mid)
        mergeSort(A, mid+1, high)
        merge(A, low, mid, high)
```



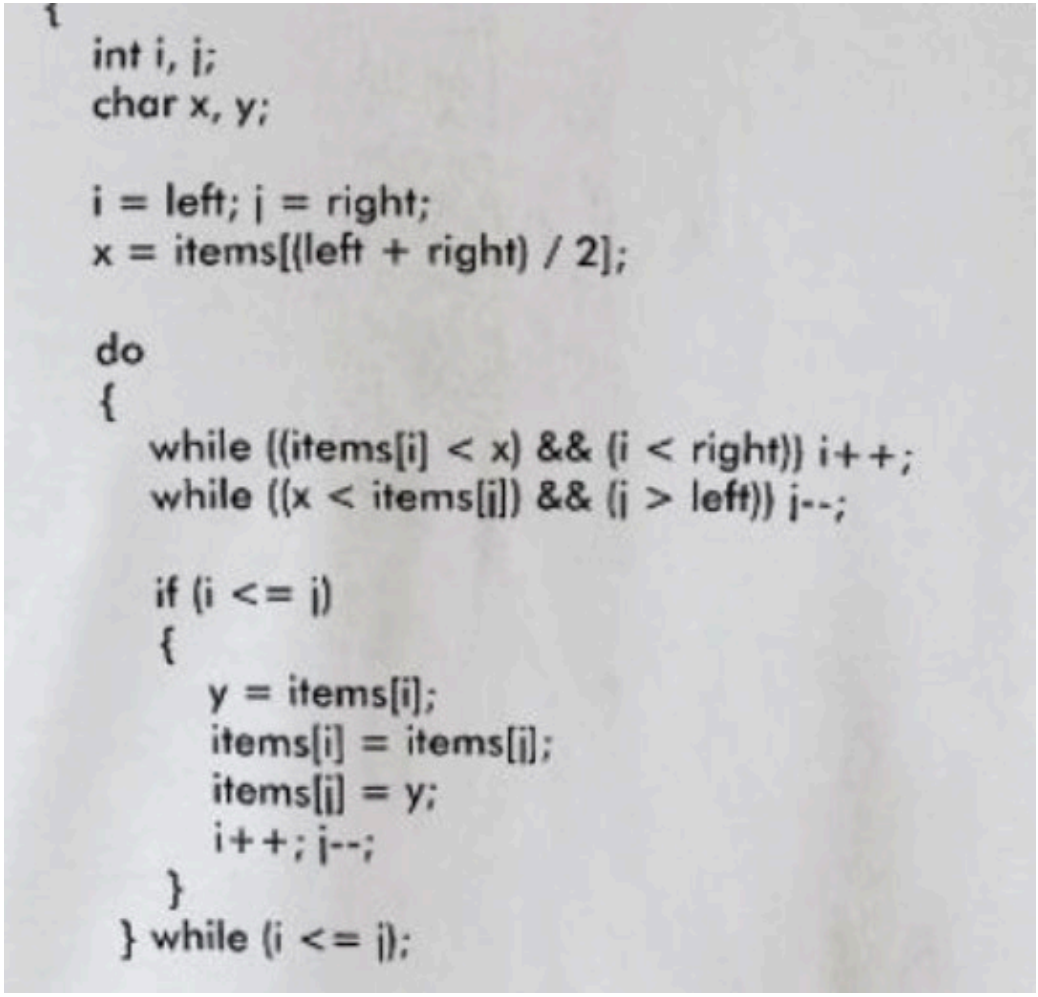
IN PLACE MERGE SORT?



QUESTION

Which algorithm is this snippet of code from?

A. Mergesort
B. Quicksort
C. Heapsort
D. Selectionsort
E. Dijkstra's SSSP



```
{
  int i, j;
  char x, y;

  i = left; j = right;
  x = items[(left + right) / 2];

  do
  {
    while ((items[i] < x) && (i < right)) i++;
    while ((x < items[j]) && (j > left)) j--;

    if (i <= j)
    {
      y = items[i];
      items[i] = items[j];
      items[j] = y;
      i++; j--;
    }
  } while (i <= j);
```

QUESTION

Which algorithm is this snippet of code from?

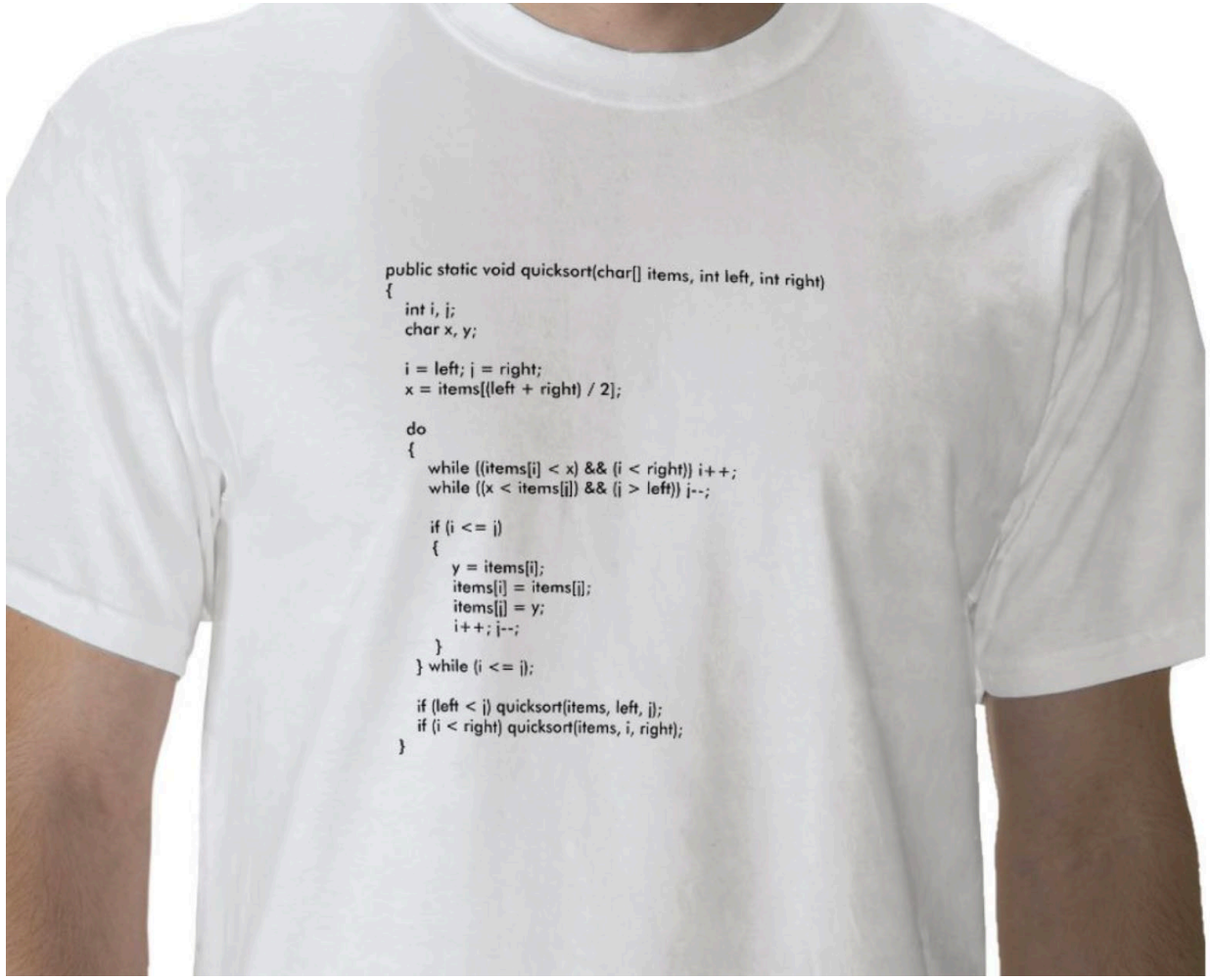
A. Mergesort

B. Quicksort

C. Heapsort

D. Selectionsort

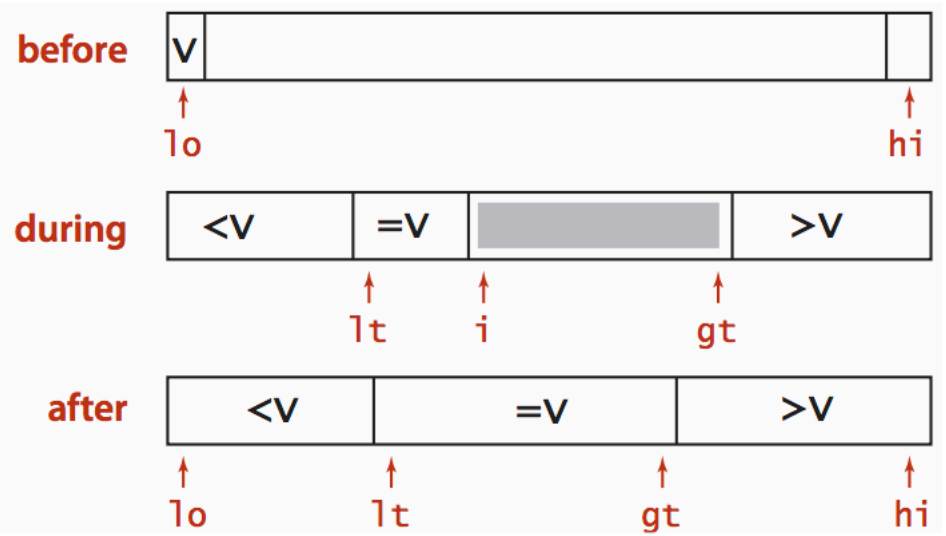
E. Dijkstra's SSSP



3 WAY PARTITION + QUICKSORT

```
quicksort(int[] a, int lo, int hi) {
    if (hi <= lo) return;
    int lt = lo, gt = hi;
    int v = a[lo];
    int i = lo + 1;
    while (i <= gt) {
        if      (a[i] < v) swap(a, lt++, i++);
        else if (a[i] > v) swap(a, i, gt--);
        else  i++;
    }

    quicksort(a, lo, lt-1);
    quicksort(a, gt+1, hi);
}
```



Modified from http://algs4.cs.princeton.edu/23quicksort/Quick3way.java.html

QUESTION

True or False: Quicksort will preserve the ordering of elements that have the same key.

A. True

B. False

QUESTION

True or False: Quicksort will preserve the ordering of elements that have the same key.

A. True

B. **False**

QUICKSORT: STABLE?

A sorting algorithm is stable if two objects with equal keys appear in the same order in the output as in input.

- No changing places if you have the same key!

Why? Sort by lastname only:

input	stable sort	unstable sort
soh, aaron	**chan**, hazel	**chan**, hazel
soh, harold	**soh**, aaron	**soh**, harold
chan, hazel	**soh**, harold	**soh**, aaron

Is quicksort stable?

A. Yesssssss.
B. **Noooooo**
C. Who knows?
D. Stable as an ikea table.

Consider [a,b,c] where c < a =b

QUESTION

I have an unsorted array A of size n that has ***distinct*** elements from 1 to n. e.g., $A = [2, 3, 1]$, or $A = [4, 2, 1, 3, 5]$.

The time complexity required to output A in sorted order is:

A. $O(1)$
B. $O(n)$
C. $O(n \log n)$
D. $O(n^2)$
E. Larger than the above

QUESTION

I have an unsorted array A of size n that has ***distinct*** elements from 1 to n. e.g., $A = [2, 3, 1]$, or $A = [4, 2, 1, 3, 5]$.

The time complexity required to print A in sorted order is:

A. $O(1)$

B. $\boldsymbol{O(n)}$

C. $O(n \log n)$

D. $O(n^2)$

E. Larger than the above

QUESTION

The depth of any 2 leaves in a heap differs at most by 1.

True or False?

A. True

B. False

QUESTION

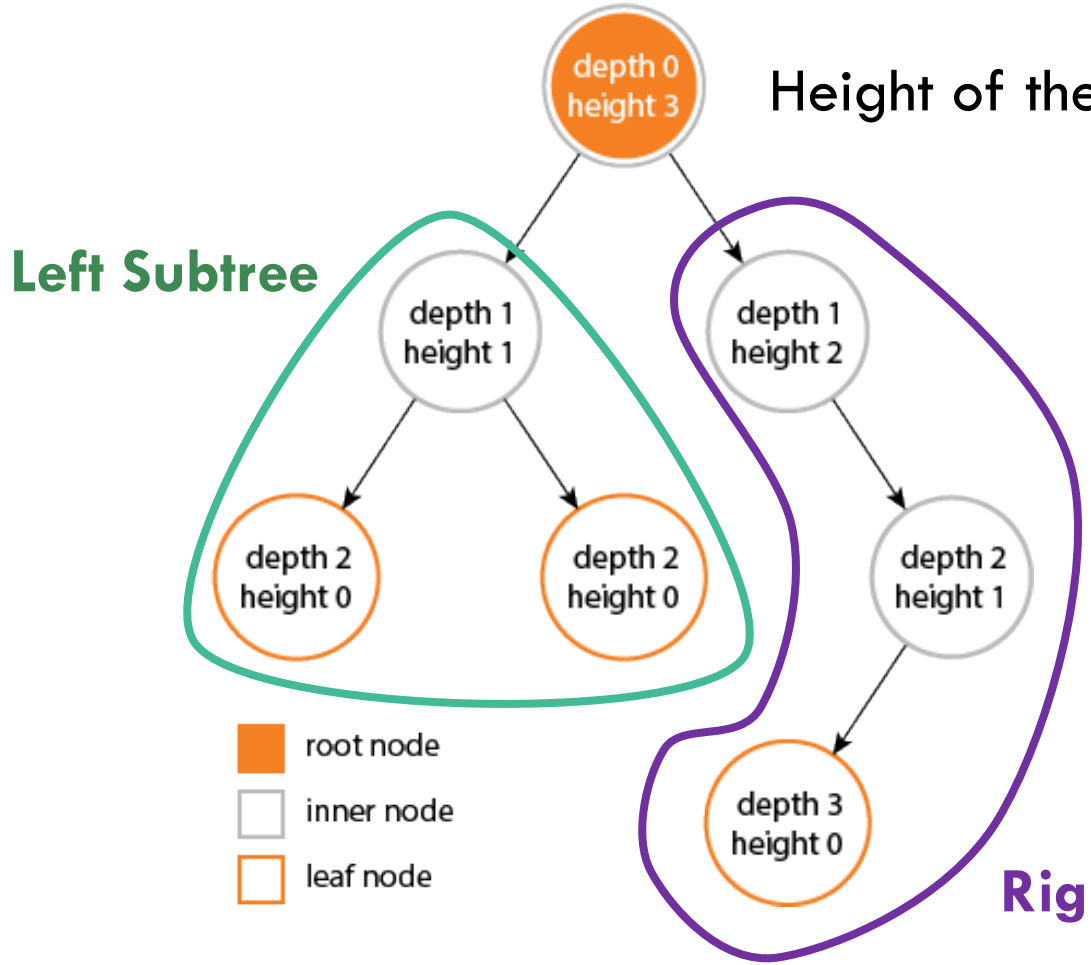
The depth of any 2 leaves in a heap differs at most by 1.

True or False?

A. **True**

B. False

A BINARY TREE



Height of the tree = depth of its deepest node

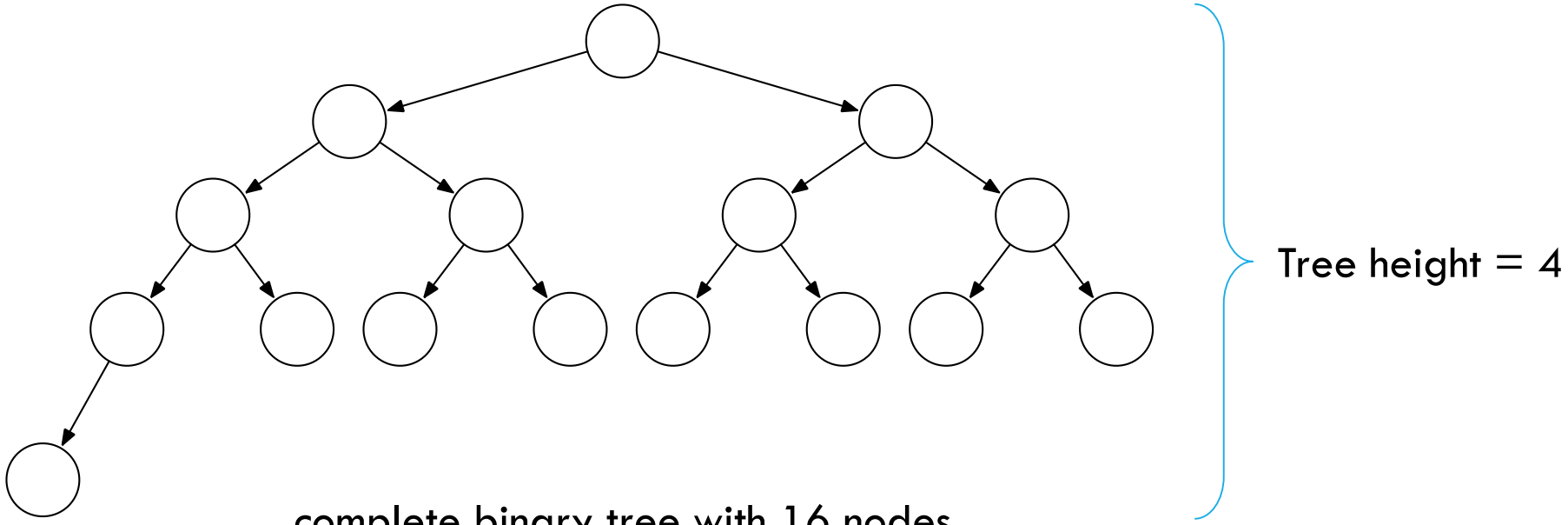
Height of a node: number of edges on the longest path from node to a leaf (via its children)

Depth of a node: number of edges to the root

COMPLETE BINARY TREE

Binary Tree: nodes with links to left and right binary trees (or empty)

Complete tree: Perfectly balanced, except for bottom level.



complete binary tree with 16 nodes

QUESTION

Which data structure is space and time efficient for storing a heap?

A. Linked List
B. Doubly-Linked List
C. Array
D. Tree
E. None of the above

QUESTION

Which data structure is space and time efficient for storing a heap?

A. Linked List

B. Doubly-Linked List

C. **Array**

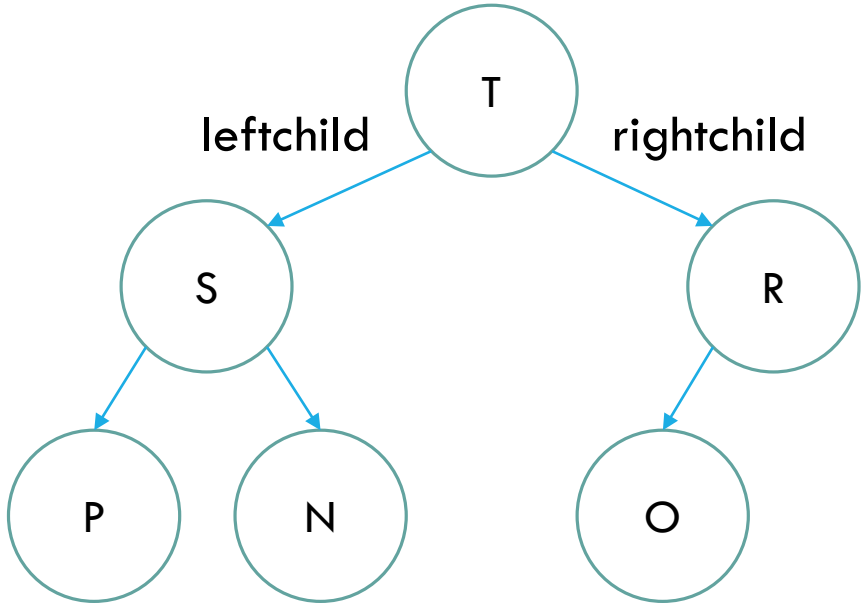
D. Tree

E. None of the above

TREE REPRESENTATION

```
Class Node
      Object key
      Object data

      Node leftchild
      Node rightchild
      Node parent
```



ARRAY REPRESENTATION

Indices start at 1

Take nodes in level order

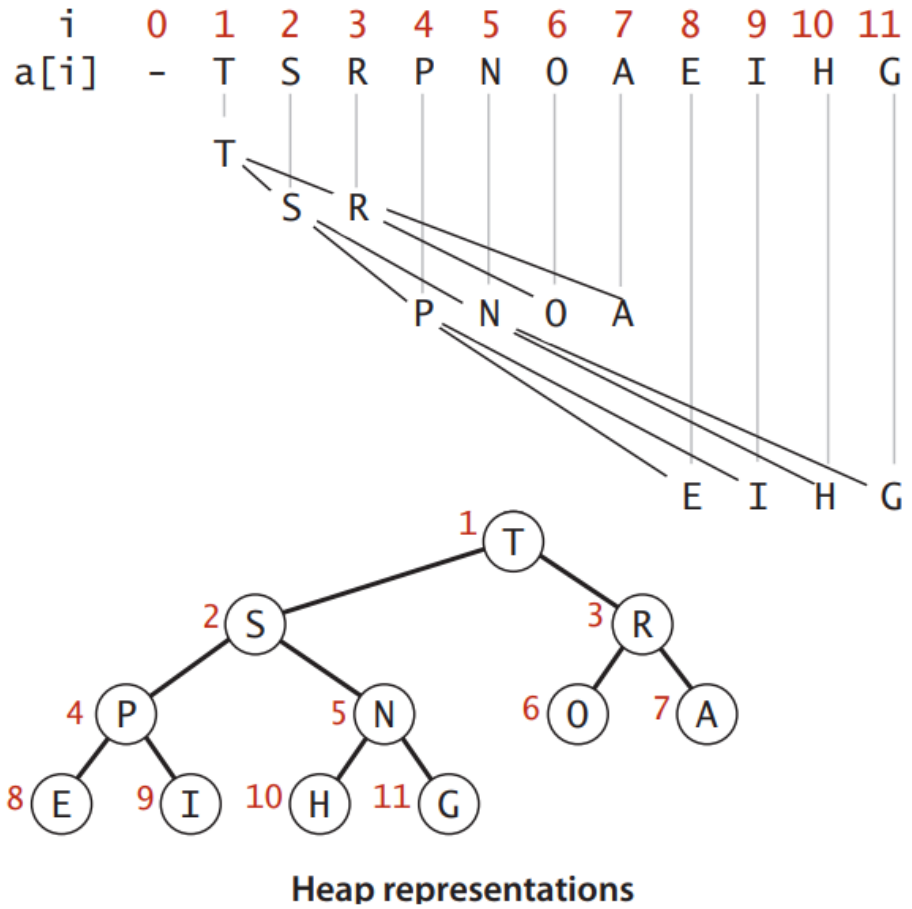
No links needed

Parent of node k is at $\boldsymbol{k/2}$

Children of node k are at:

Left Child: $\boldsymbol{2k}$

Right Child: $\boldsymbol{2k+1}$



Heap representations

QUESTION

If a key in a max-heap is decreased, what operation is needed to restore the max-heap property?

A. Swim
B. Sink
C. Merge
D. None of the above

QUESTION

If a key in a max-heap is decreased, what operation is needed to restore the max-heap property?

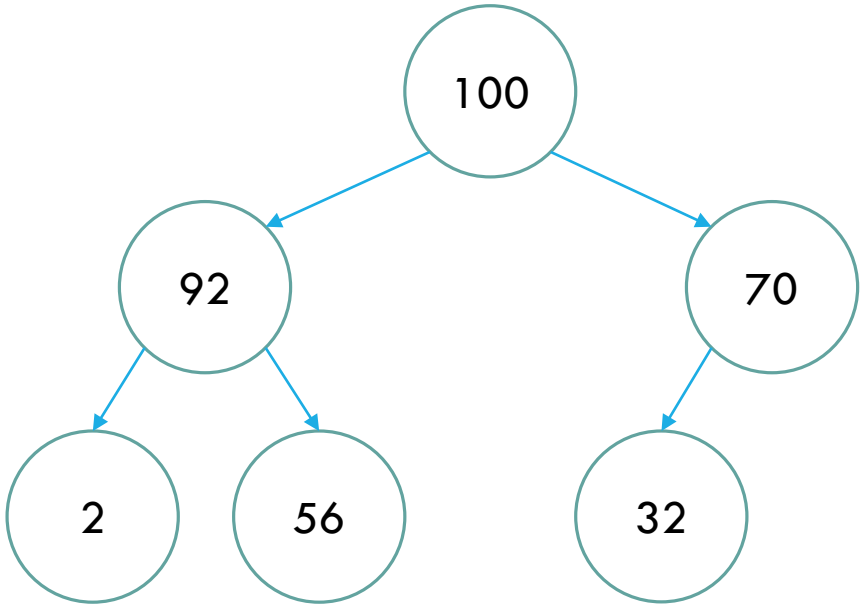
A. Swim

B. Sink

C. Merge

D. None of the above

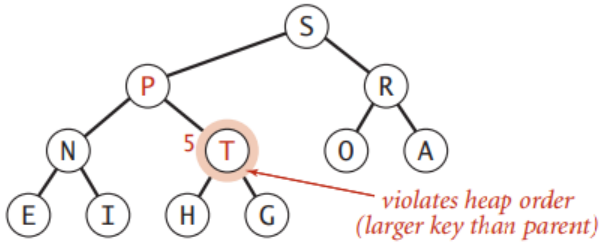
BINARY HEAP

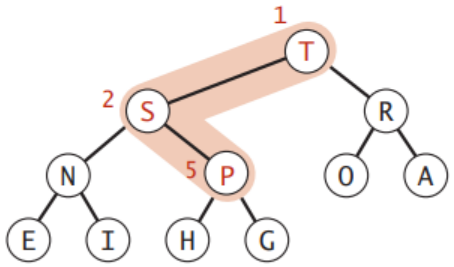


Max heap property:

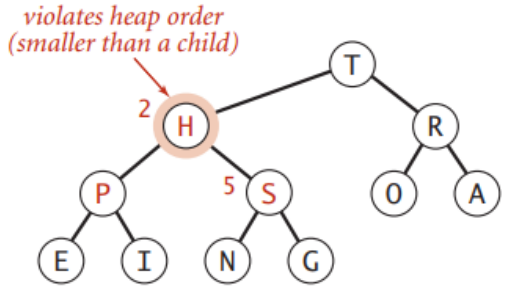
- the key of each node is larger than or equal to the keys in its children

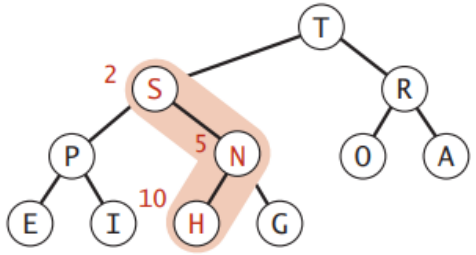
TWO IMPORTANT OPERATIONS: SWIM & SINK





Swim





Sink

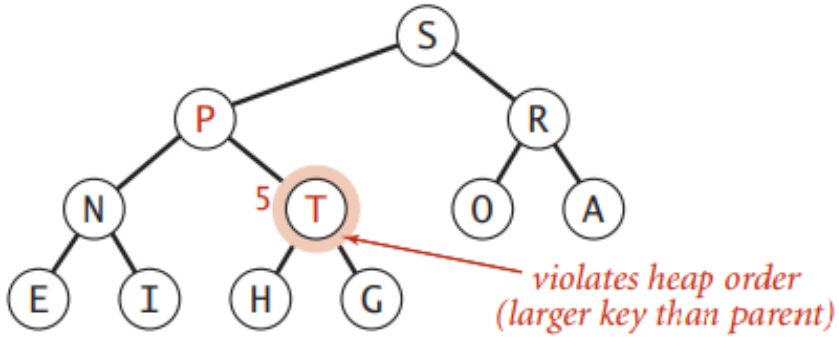
We use these operations to correct binary trees to satisfy the max heap property

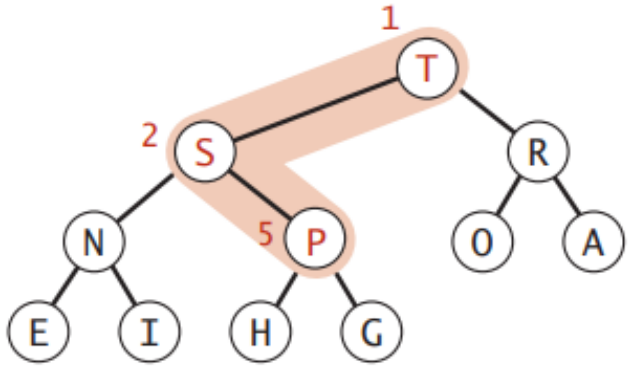
SWIM or (SHIFTUP, BUBBLEUP, INCREASEKEY)

Problem: A key becomes larger than its parent's key.

Solution:

- Swap child with parent
- Repeat until heap order restored





SWIM or (SHIFTUP, BUBBLEUP, INCREASEKEY)

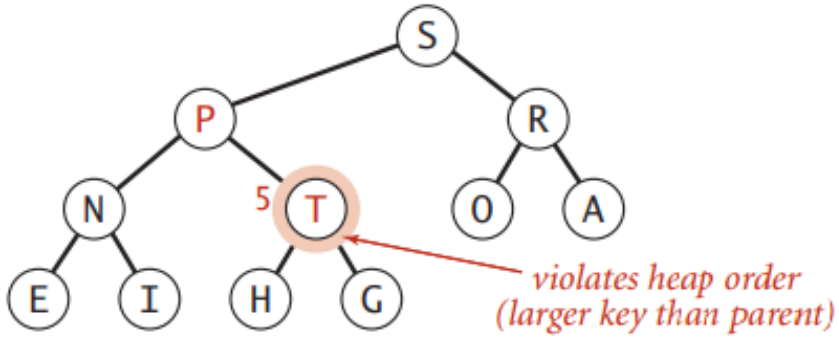
Problem: A key becomes larger than its parent's key.

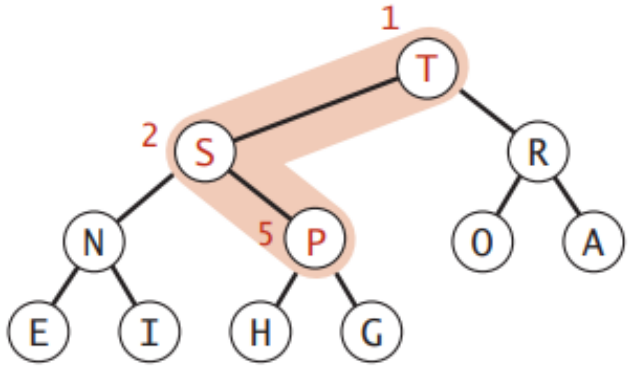
Solution:

- Swap child with parent
- Repeat until heap order restored

Pseudocode for array implementation

```
function swim(A, k)
    while (k > 1) and A[k/2].key < A[k].key
        swap(A[k], A[k/2])
        k = k/2
```



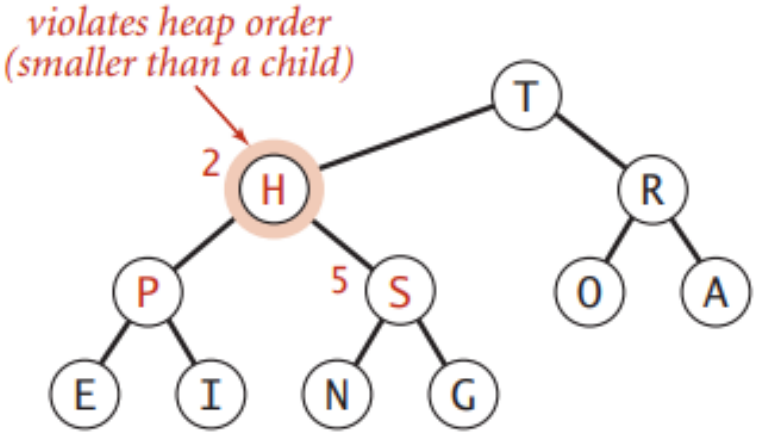


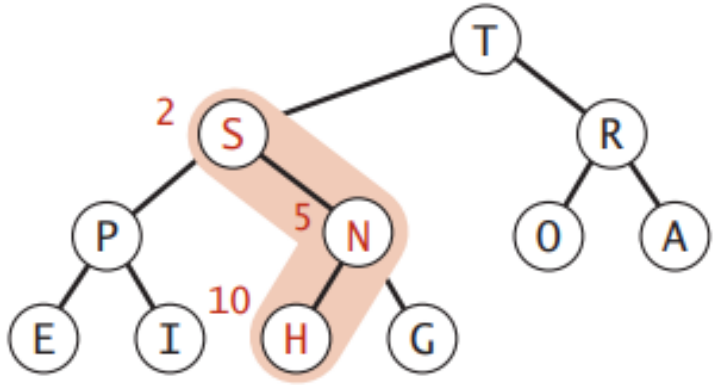
SINK or (SHIFTDOWN, BUBBLEDOWN, HEAPIFY)

Problem: A key becomes smaller than one (or both) of its children's keys.

Solution:

?



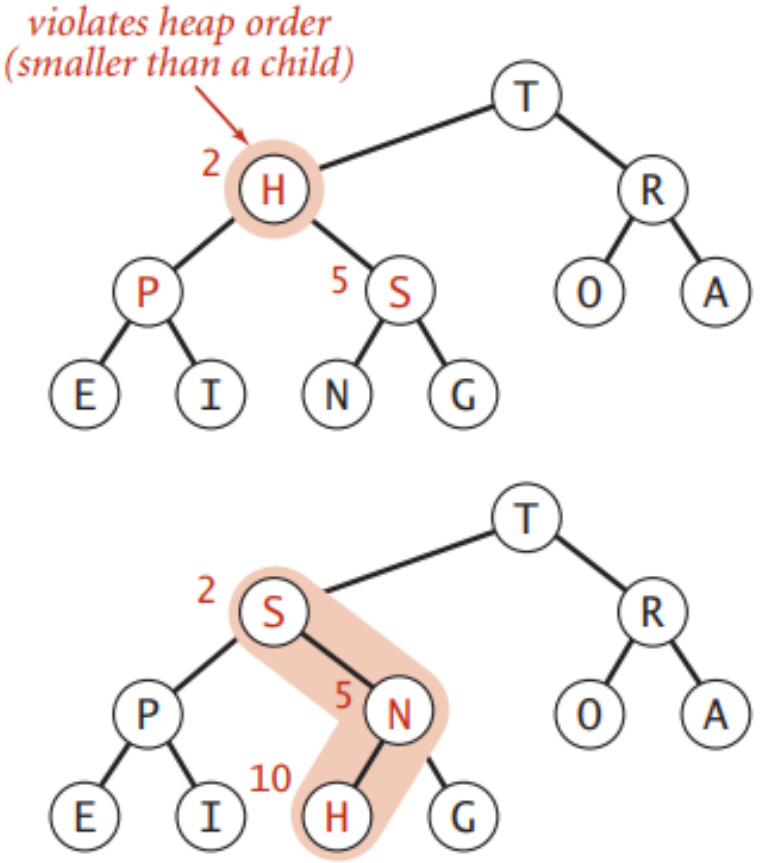


SINK or (SHIFTDOWN, BUBBLEDOWN, HEAPIFY)

Problem: A key becomes smaller than one (or both) of its children's keys.

Solution:

- Swap larger child with parent
- Repeat until heap order restored

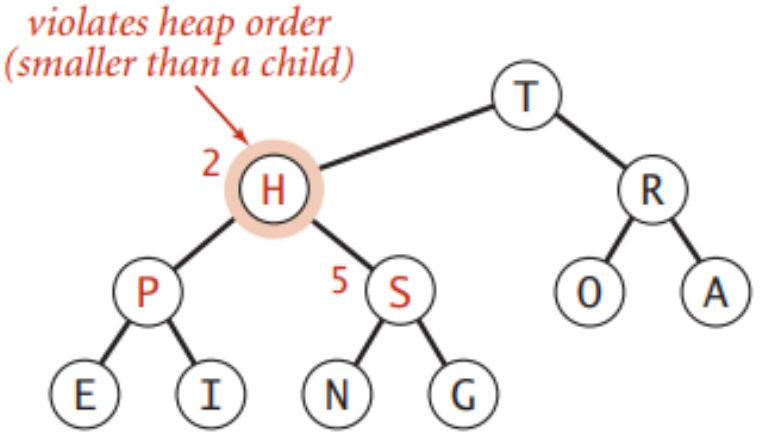


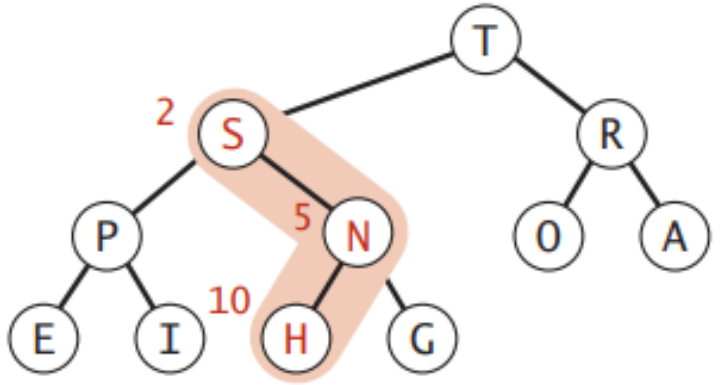
SINK or (SHIFTDOWN, BUBBLEDOWN, HEAPIFY)

Problem: A key becomes smaller than one (or both) of its children's keys.

Solution:

- Swap larger child with parent
- Repeat until heap order restored





Pseudocode for array implementation

```
function sink(A, k)
   while (2k <= n)
        j = 2k
        if (j<n and (A[j].key < A[j+1].key)) j++
        if not (A[k].key < A[j].key) break
        swap(A[k], A[j])
        k = j
```

QUESTION

Who invented the algorithm for creating a heap in linear time?

A. Donald Knuth
B. Robert Floyd
C. Edsger Dijkstra
D. Dlorah Hos
E. None of the above

QUESTION

Who invented the algorithm for creating a heap in linear time?

A. Donald Knuth
B. **Robert Floyd**
C. Edsger Dijkstra
D. Dlorah Hos
E. None of the above

CLEVER CREATION IN $O(n)$ TIME

Invented by Robert Floyd in 1964

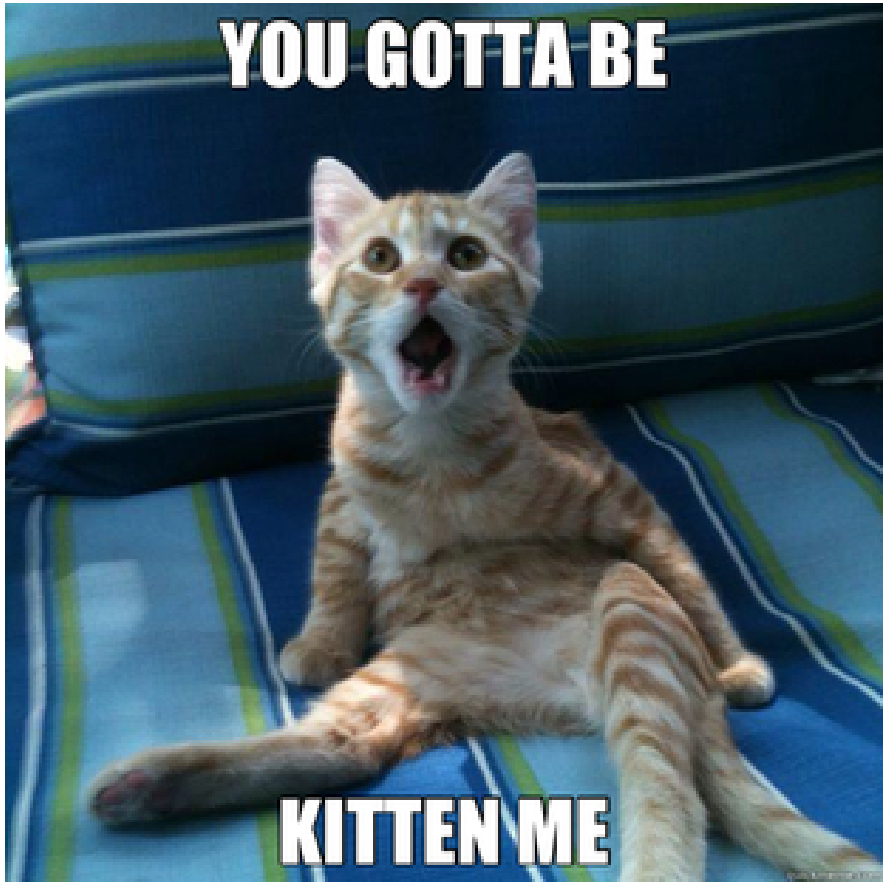
- invented invariants (among other things)
- we'll hear about him again in when we meet graphs!

The idea:

- View the input array as a binary tree
- "Bottom up" fixing of the tree to satisfy MaxHeap property

CODE FOR CREATION IN $O(n)$ TIME

```
function buildHeap(A)
    for i from A.length/2 to 1
        sink(i)
```



QUESTION

I need to sort an array of n elements, but with two constraints:

- only $O(1)$ additional space, and
- $O(n \log n)$ worst case time guarantee

What algorithm should I use?

A. Insertion sort

B. Mergesort

C. Quicksort

D. Heapsort

E. Bubble sort

QUESTION

I need to sort an array of n elements, but with two constraints:

- only $O(1)$ additional space, and
- $O(n \log n)$ worst case time guarantee

What algorithm should I use?

A. Insertion sort

B. Mergesort

C. Quicksort

D. Heapsort

E. Bubble sort

HEAPSORT CODE

```
function heapsort(A)
        n = A.length
        // create the binary heap
        for i = n/2 to 1
                sink(A, k, n)
        // swap and sink
        while (n > 1)
                swap(A, 1, n);
                sink(A, 1, --n);
```

Creating a binary heap in place takes $O(n)$

Swapping and Sinking takes _____

Note: uses modified sink and swim with array length

HEAPSORT CODE

```
function heapsort(A)
        n = A.length
        // create the binary heap
        for i = n/2 to 1
                sink(A, k, n)
        // swap and sink
        while (n > 1)
                swap(A, 1, n);
                sink(A, 1, --n);
```

Creating a binary heap in place takes $O(n)$

Swapping and Sinking takes $O(n \log n)$

Note: uses modified sink and swim with array length

QUESTION

True/False:

In a Binary Search Tree, a node's key will always be smaller than its left child and larger than its right child.

A. True

B. False

QUESTION

True/False:

In a Binary Search Tree, a node's key will always be smaller than its left child and larger than its right child.

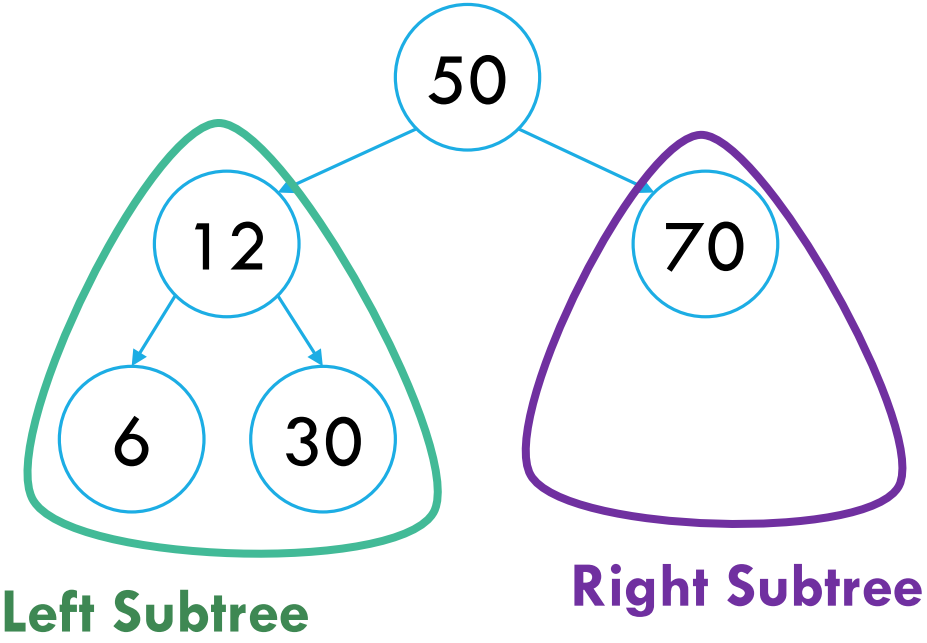
A. True

B. **False**

BINARY SEARCH TREES

Binary Search Tree Properties:

- A node's left subtree only contains nodes with keys strictly less than the node's key
- A node's right subtree only contains nodes with keys strictly larger than the node's key
- The left and right subtrees are binary trees.
- All keys belong to a total order*.



* $\forall u, v \in K$ when $u \neq v$ either $u < v$ or $v > u$ (no two different keys can be considered equal)

QUESTION

Naruto's uncle, Uzumaki, was killed in an unfortunate skiing accident. Everyone in Naruto's family is stored in a Balanced BST by height, and the next tallest family member inherits Uzumaki's estate. How can we efficiently find this member?

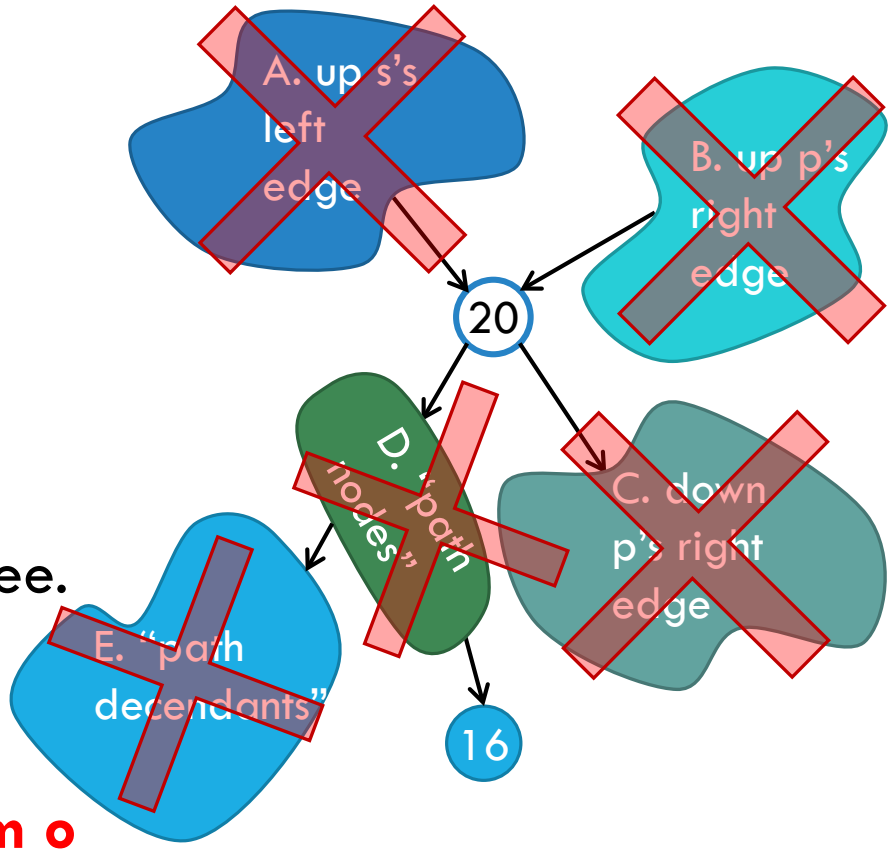
A. Binary Search
B. Successor
C. Predecessor
D. Array Halving

FINDING THE SUCCESSOR NODE

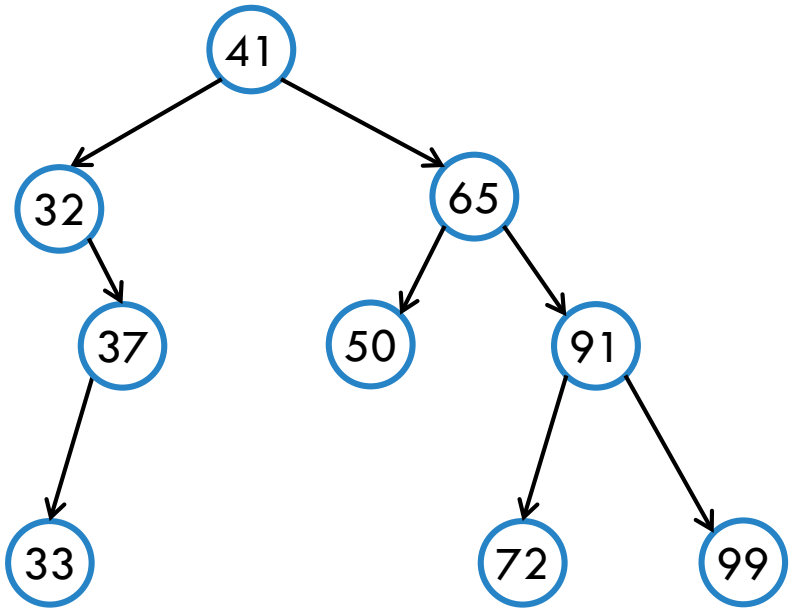
Given a node o, there are 2 cases:

- it has a right subtree
  - return searchMin(o.m_rightTree)
- it doesn't have a right subtree
  - **Claim:** successor will be higher up in the tree.
  - 2 cases:
    - o is the left child: successor is o's parent
    - o is the right child: **first ancestor up from o that has key > k. ("first right ancestor")**

Examine each in turn



QUESTION



To make this tree height balanced, what should I do and to which nodes?

A. Left rotate 32

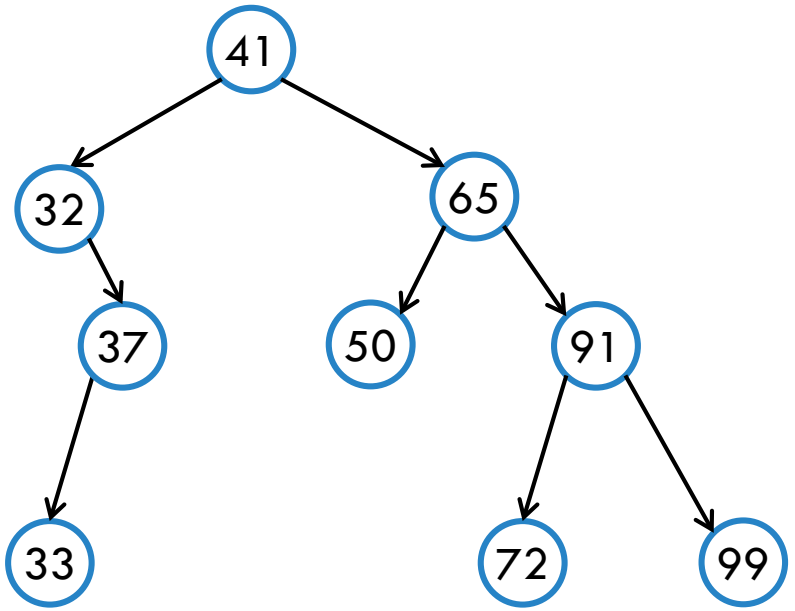
B. Left rotate 37

C. Right rotate 37

D. Right rotate 37, left rotate 32

E. Left rotate 32, right rotate 33

QUESTION



To make this tree height balanced, what should I do and to which nodes?

A. Left rotate 32

B. Left rotate 37

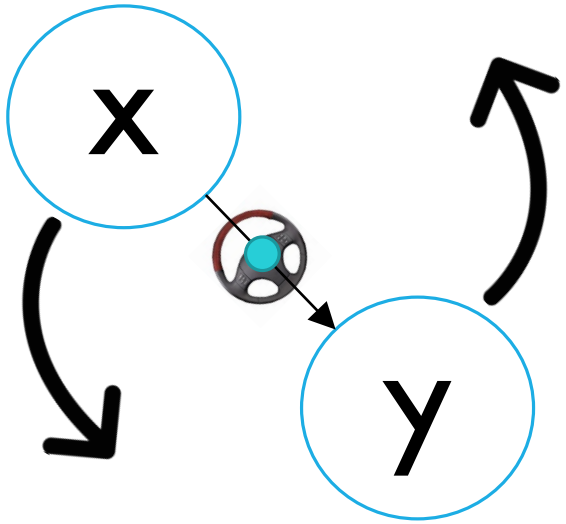
C. Right rotate 37

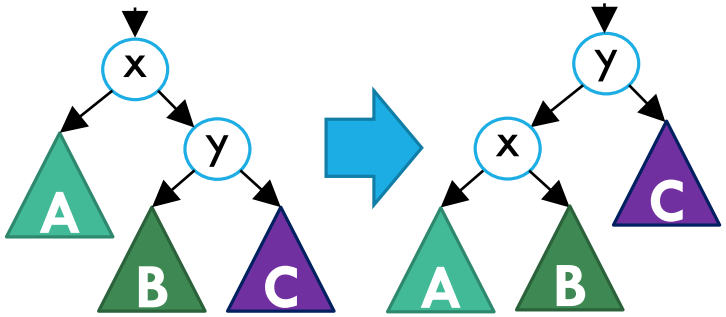
D. **Right rotate 37, left rotate 32**

E. Left rotate 32, right rotate 33

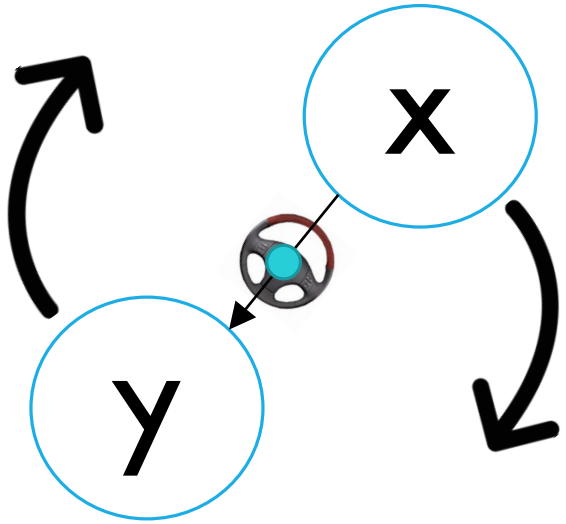
HOW I REMEMBER IT

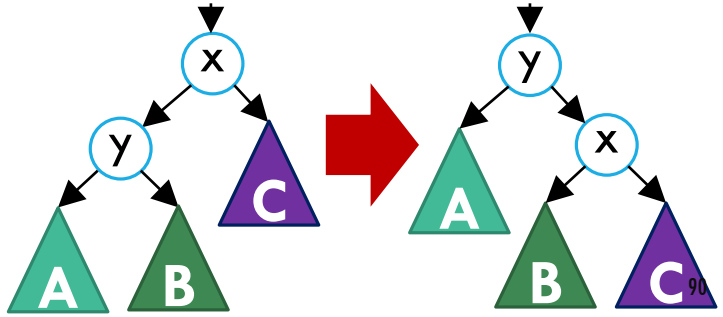
left / anti-clockwise





right / clockwise



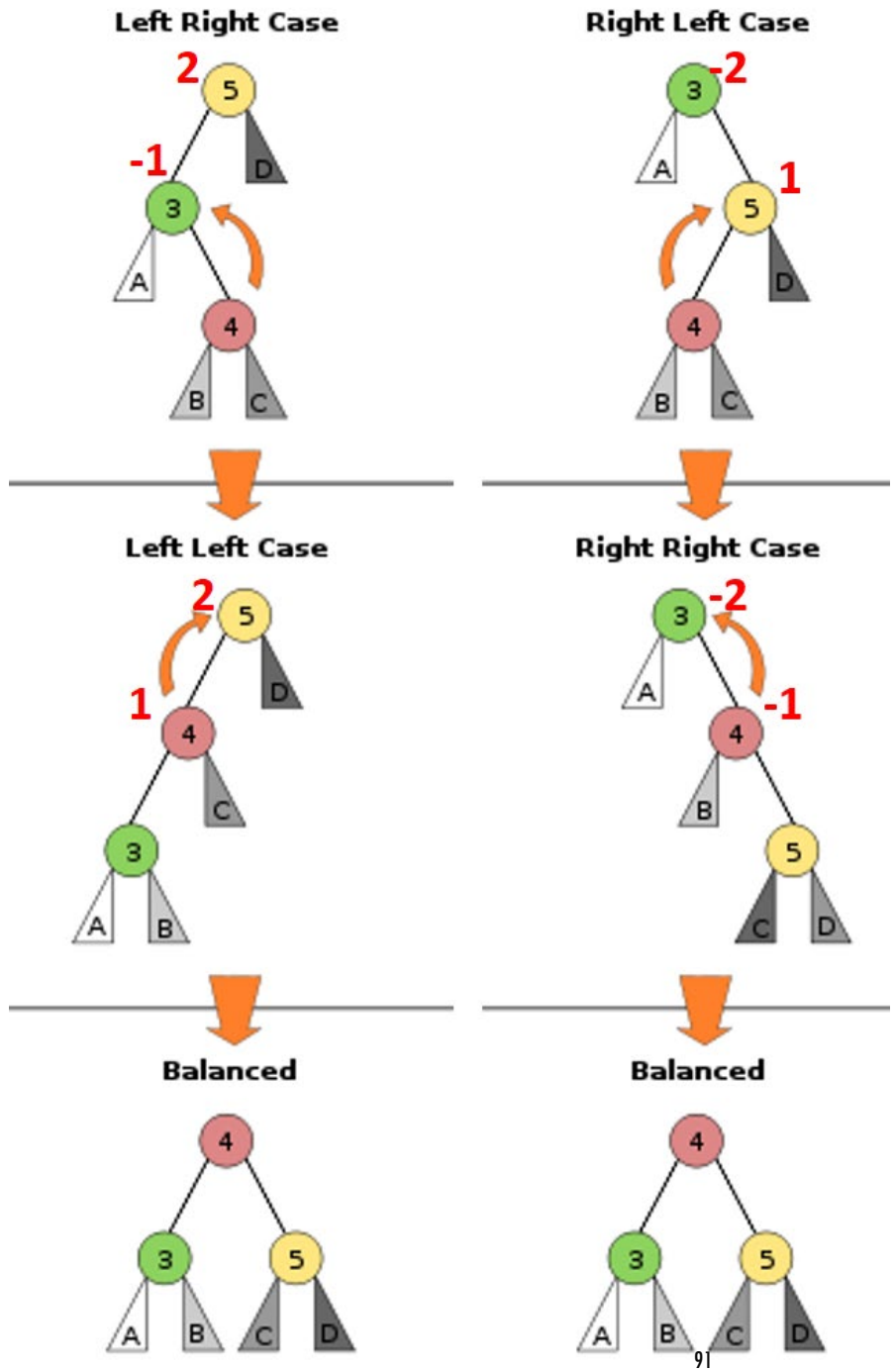


ALGORITHM TO FIX IMBALANCES

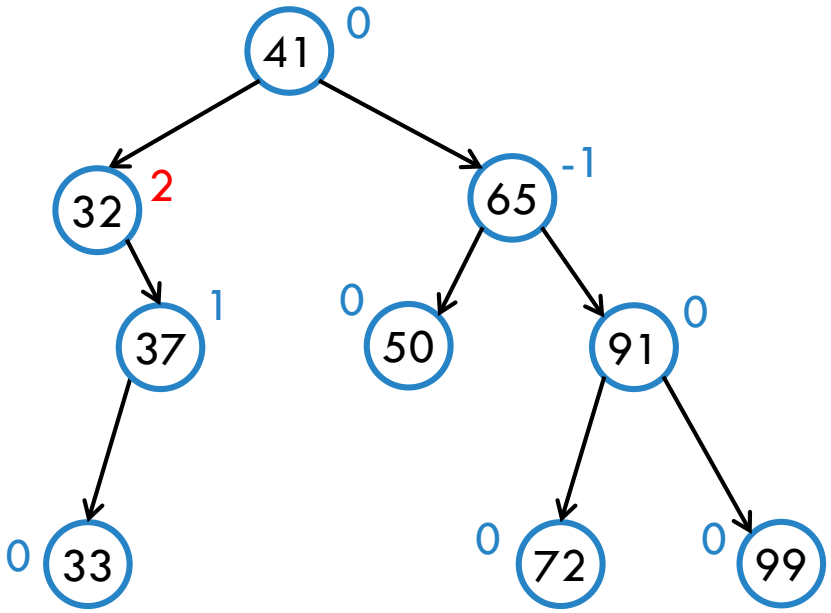
For both insertion and deletions:

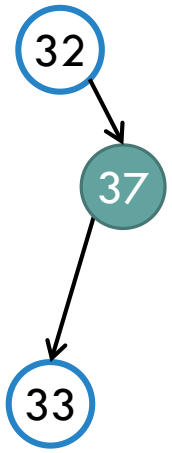
- walk up the tree (to the root) from the inserted/deleted node & update balance factors.
- if we find a height-balance violation, fix it via a rotation:

```
if tree is right heavy
      if tree's right subtree is left heavy
            right rotate, left rotate
      else
            left rotate
else if tree is left heavy
      if tree's left subtree is right heavy
            left rotate, right rotate
      else
            right rotate
```



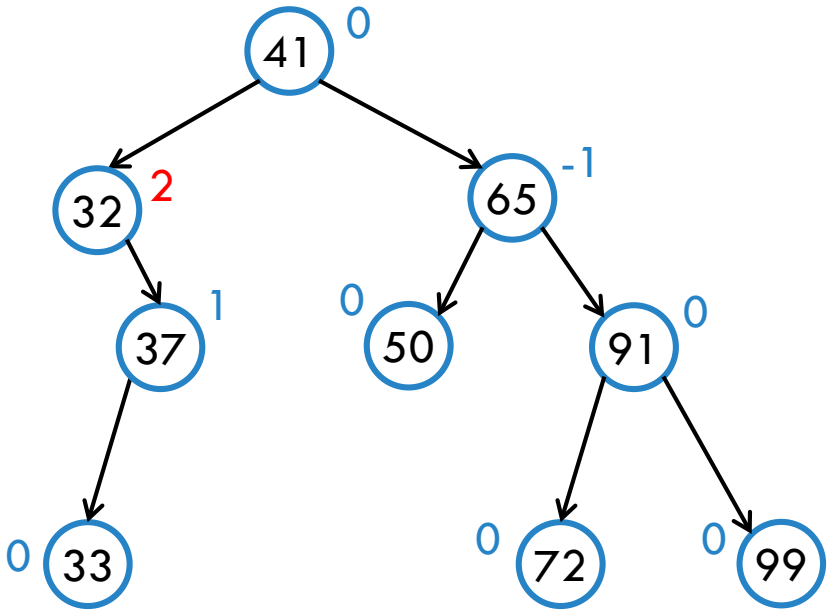
LET'S TRY THE DOUBLE ROTATION!

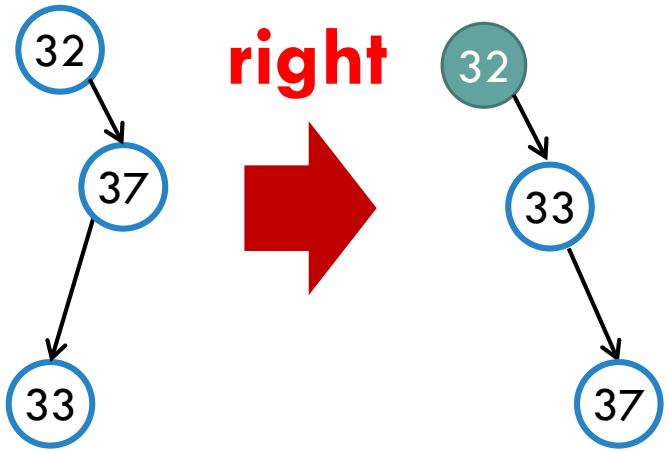




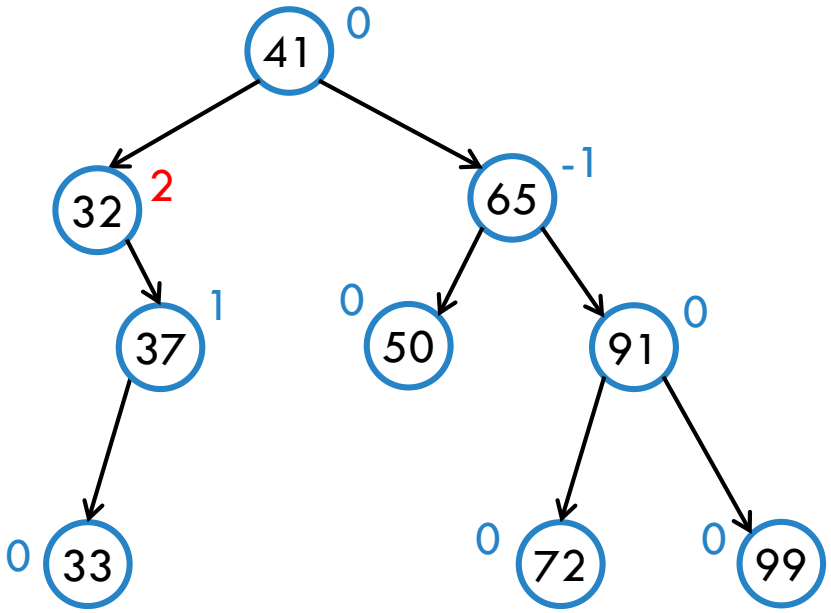
left or **right**

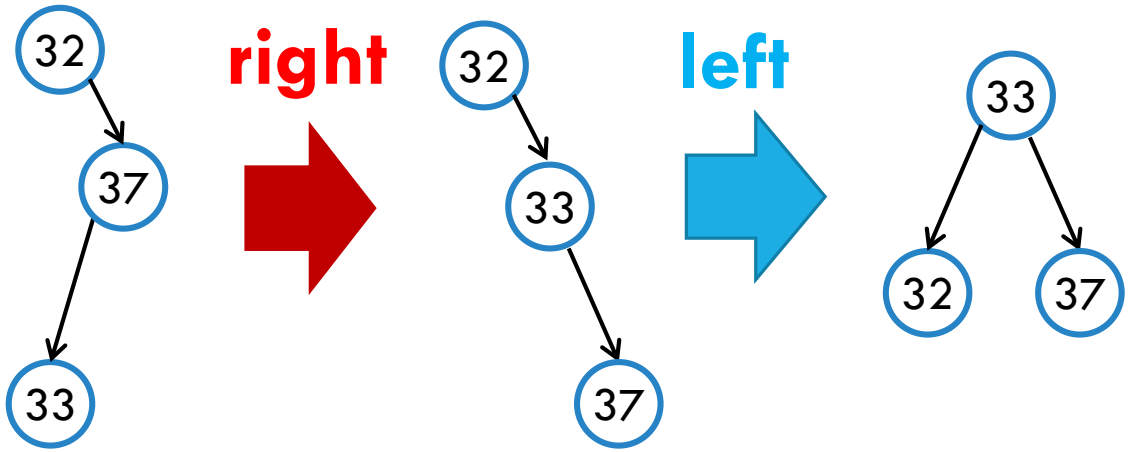
LET'S TRY THE DOUBLE ROTATION!



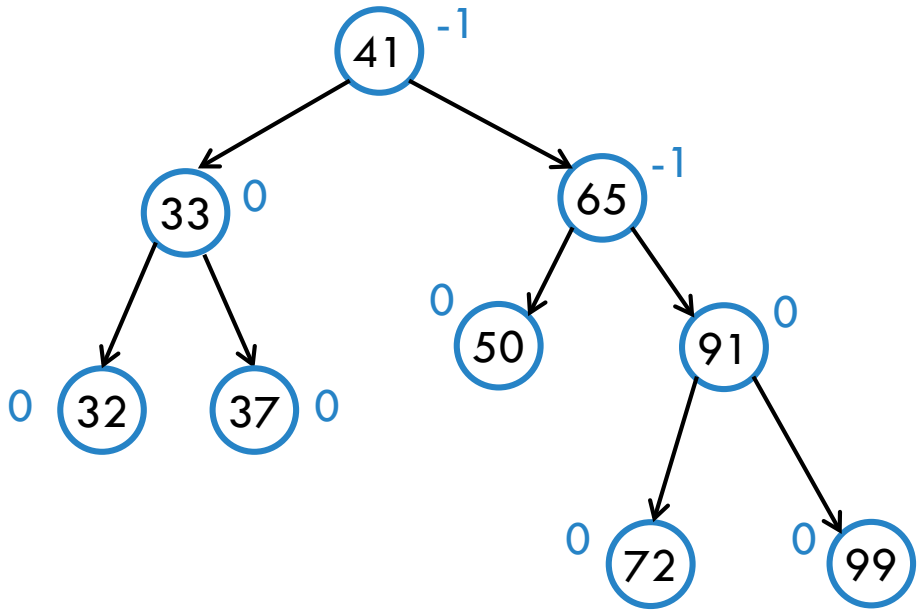


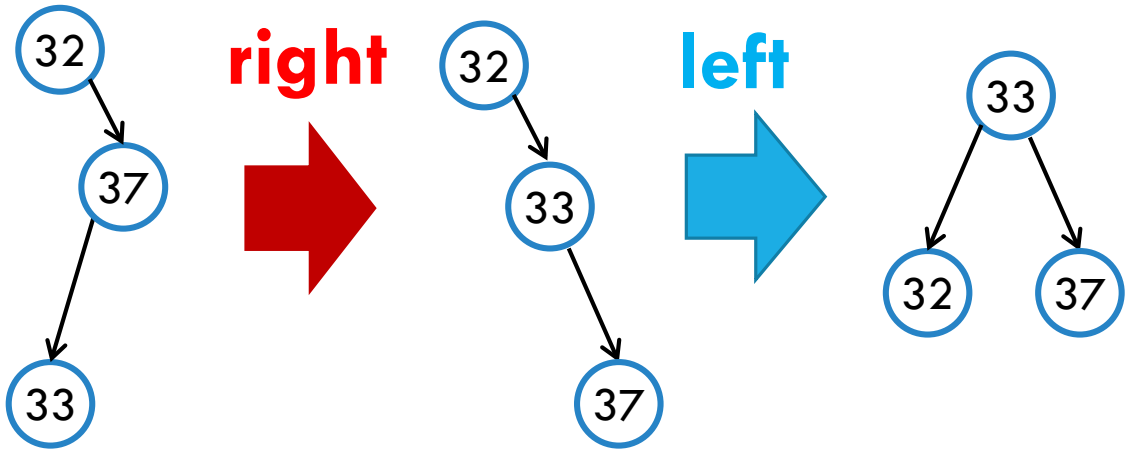
LET'S TRY THE DOUBLE ROTATION!





LET'S TRY THE DOUBLE ROTATION!





It worked!

QUESTION

In a hash table, which of the following operations are fast (e.g., constant or log time)?

A. Floor
B. Insert
C. Successor
D. Max

QUESTION

In a hash table, which of the following operations are fast (e.g., constant or log time)?

A. Floor

B. Insert

C. Successor

D. Max

WHAT DO WE GIVE UP?

no free lunch!

Data Structure	Avg. Insert Time	Avg. Search Time	Avg. Max/Min	Avg. Floor/ Ceiling
Unordered Array / Linked List	O(1)	O(n)	O(n)	O(n)
Ordered Array / Linked List	O(n)	**O(log n)**	O(1)	O(log n)
Balanced Binary Search Tree (AVL)	O(log n)	O(log n)	O(log n)	O(log n)
Hash Table	**O(1)**	**O(1)**	**O(n)**	**O(n)**

SIMPLE UNIFORM HASHING ASSUMPTION

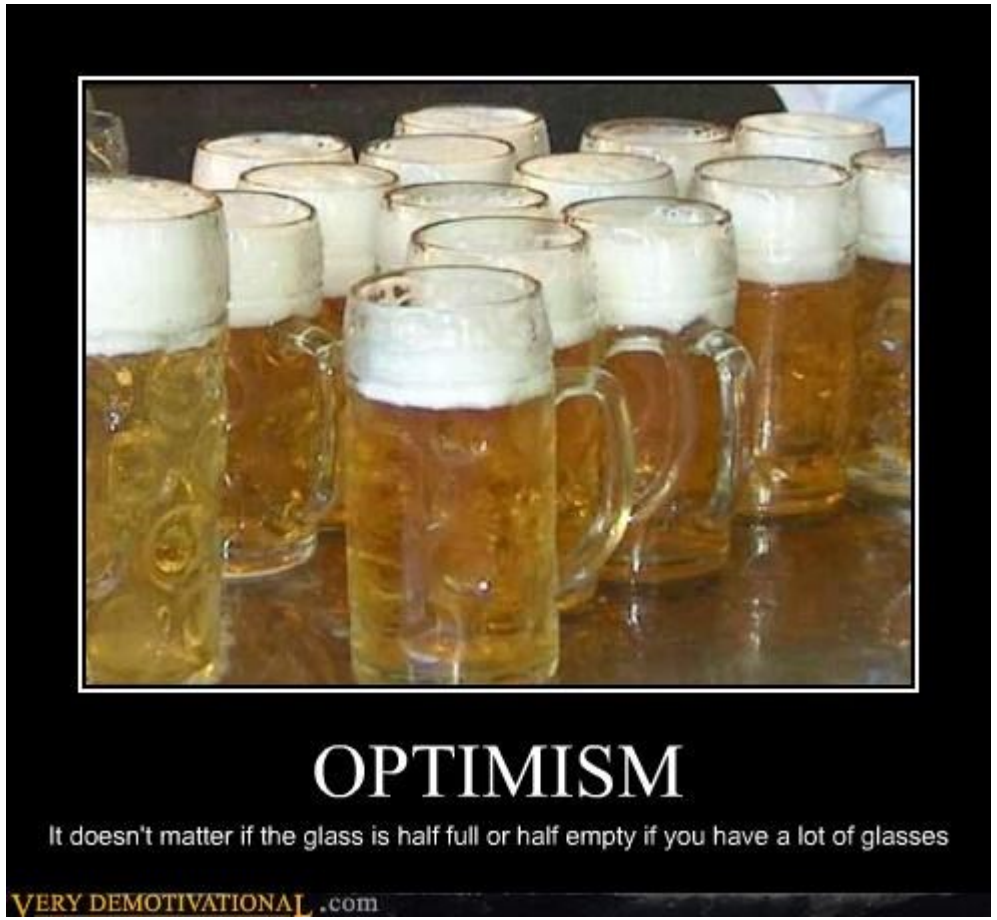
An optimistic assumption:

Every key is ***equally likely*** to map to every bucket

Keys are mapped independently.

Intuition:

- Each key is put in a random bucket.
- As long as enough buckets, not too many keys in any one bucket.



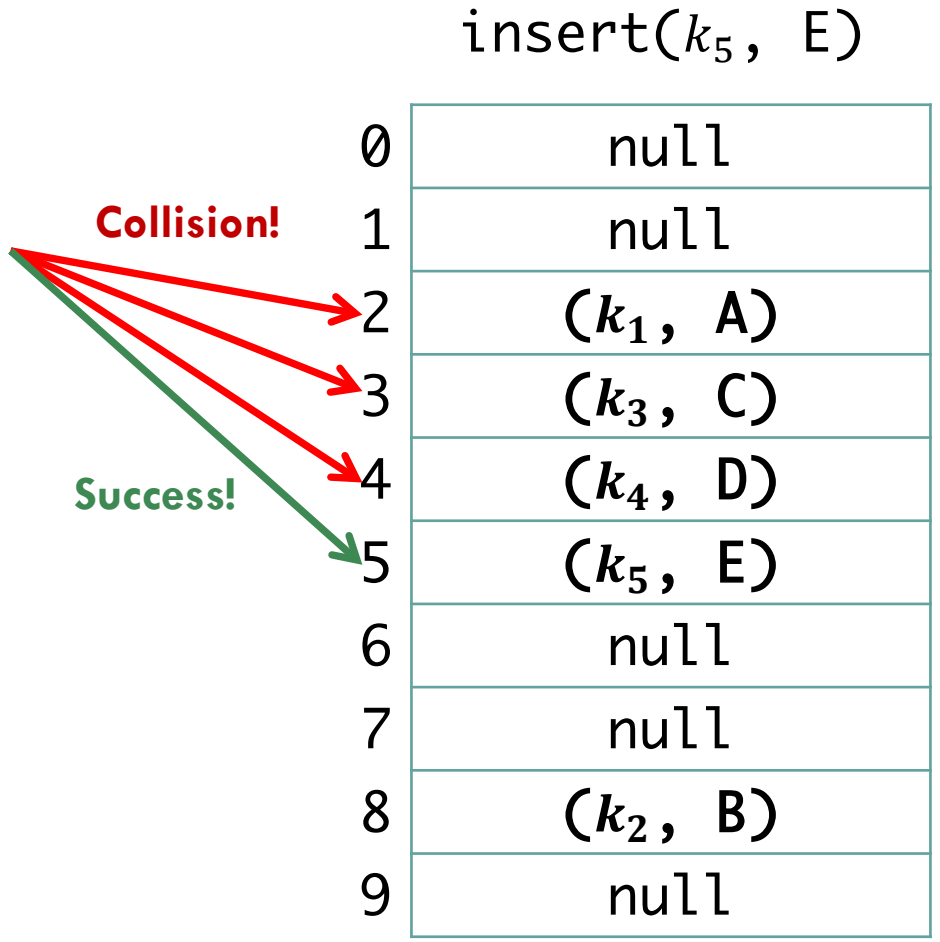
OPEN ADDRESSING: LINEAR PROBING

Idea: On collision, probe until you find an empty slot.

Question: How to probe?

Linear Probing: keep checking next bucket until you find an empty slot.

$$\text{index } i = (h(k) + \text{step} \times 1) \bmod m$$



A PROBLEM: PRIMARY CLUSTERS

PEER-Q
76767

cluster = collection of consecutive occupied slots

In a hash table of size 10, consider 2 clusters:

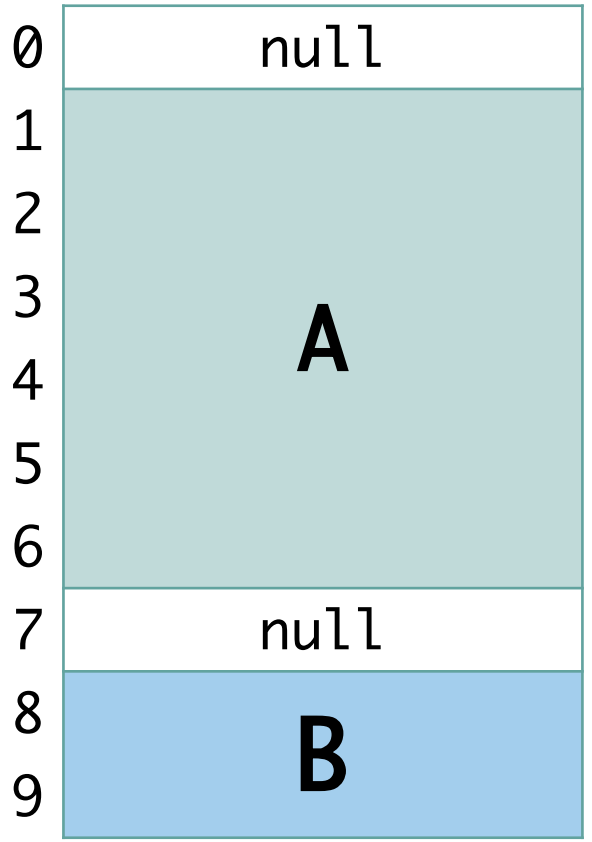
- A: size 5
- B: size 2

Probability that a new inserted key k has a bucket in

- cluster A? 5/10
- cluster B? 2/10

What happens after k is added?

Cluster grows by 1 element



WHY DO CLUSTERS MATTER?

With table size m and $n = \alpha m$ keys, the average number of linear probes is:

$\sim \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$ for search hits

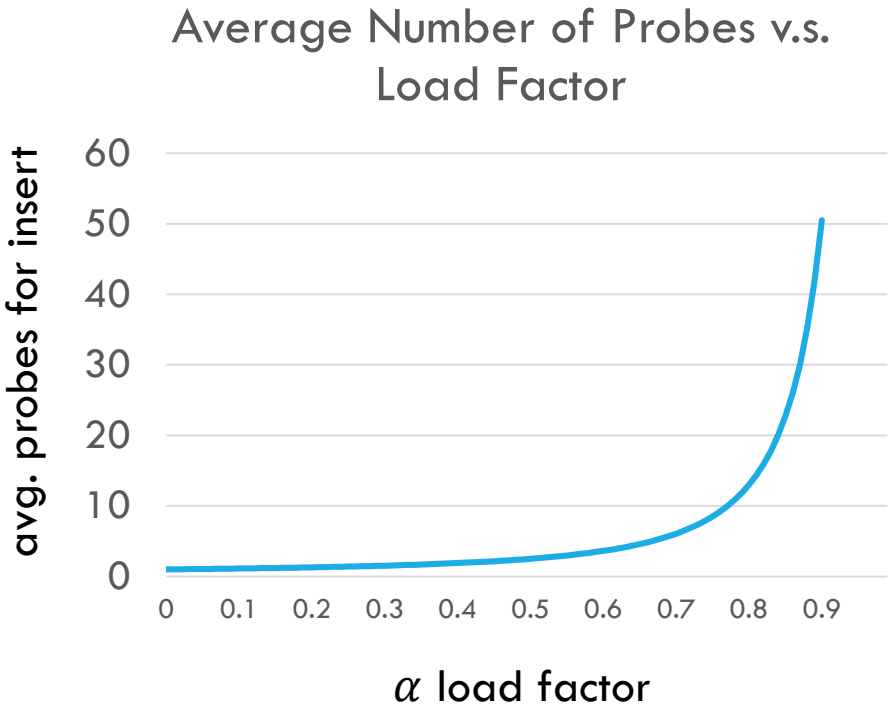
$\sim \frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right)$ for search misses

Recall $\alpha \leq 1$ is the "load factor"

(Proof beyond scope of CS2040S)

λ	1/4	1/2	2/3	3/4	9/10
successful	1.2	1.5	2.0	3.0	5.5
unsuccessful	1.4	2.5	5.0	8.5	50.5

http://www.cs.cmu.edu/afs/cs/academic/class/15210-f13/www/lectures/lecture24.pdf



OPEN ADDRESSING: QUADRATIC PROBING

Linear probing: index $i = (h(k) + \text{step} \times 1) \bmod m$

Quadratic probing: index $i = (h(k) + \text{step}^2) \bmod m$

Example:

- $h(k) = 3, m = 7$
- Step 0: $i = h(k) = 3$
- Step 1: $i = (h(k) + 1) \bmod 7 = 4$
- Step 2: $i = (h(k) + 4) \bmod 7 = 0$

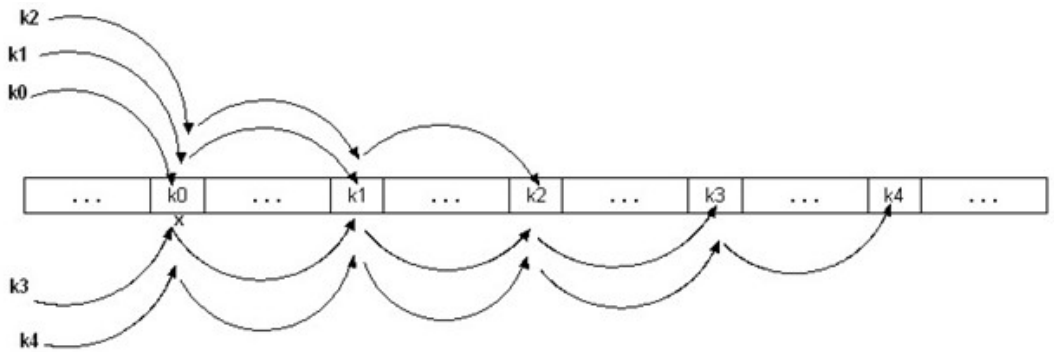
Is this a good probing method?

ONE PROBLEM: SECONDARY CLUSTERING

Milder form of the clustering problem.

Because: if two keys have the same probe position, their probe sequences are the same.

Clustering around different points (rather than the primary probe point)



How many probe sequences can there be? m

DOUBLE HASHING

Use a second hashing:

index i $= (h_1(k) + \text{step} \times h_2(k)) \bmod m$

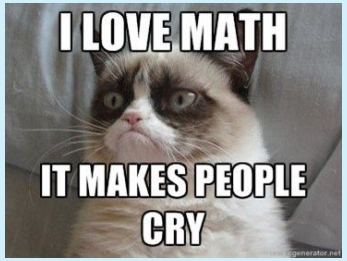
Avoids secondary clustering by providing more unique probing sequences.

Intuition:

- $h_1(k)$ **provides good "random" base address**
- $h_2(k)$ **provides good "random" sequence**

Up to how many unique indexing sequences does double hashing provide?

A. m
B. $2m$
C. m^2
D. 2^m
E.



QUESTION

I have a hash table of size 2^r where $r = 8$. Should I use the division or multiplication hashing method?

A. Division

B. Multiplication

C. Either would work just fine.

QUESTION

I have a hash table of size 2^r where $r = 8$. Should I use the division or multiplication hashing method?

A. Division

B. **Multiplication**

C. Either would work just fine.

DIVISION METHOD

$h(k) = k \bmod m$

- $m = 7, \quad h(17) = 3$
- $m = 20, \; h(100) = 0$
- $m = 20, \; h(97) = 17$
- $m = 13, \quad h(102) = 12$

Two keys k_1 and k_2 **collide** when:
$k_1 = k_2 \pmod m$

Want:

A. Consistent
B. Fast to compute, $O(1)$
C. as uniform as possible $\in [0..m-1]$

How to choose m?

CHOICE OF $m = 2^x$

Want:

A. Consistent
B. Fast to compute, $O(1)$
C. as uniform as possible $\in [0..m-1]$

Does it satisfy the desirability C?

Problem: Regularity of keys

- Input keys are *often* not uniformly distributed

If all input keys are even:

- $h(k) = k \bmod m$ must be even.

No longer uniform: will waste ½ the space and increase collisions!

Fast to compute, but does not use table well.
Also, will only use lowest order bits of k (Why?)

DIVISION METHOD

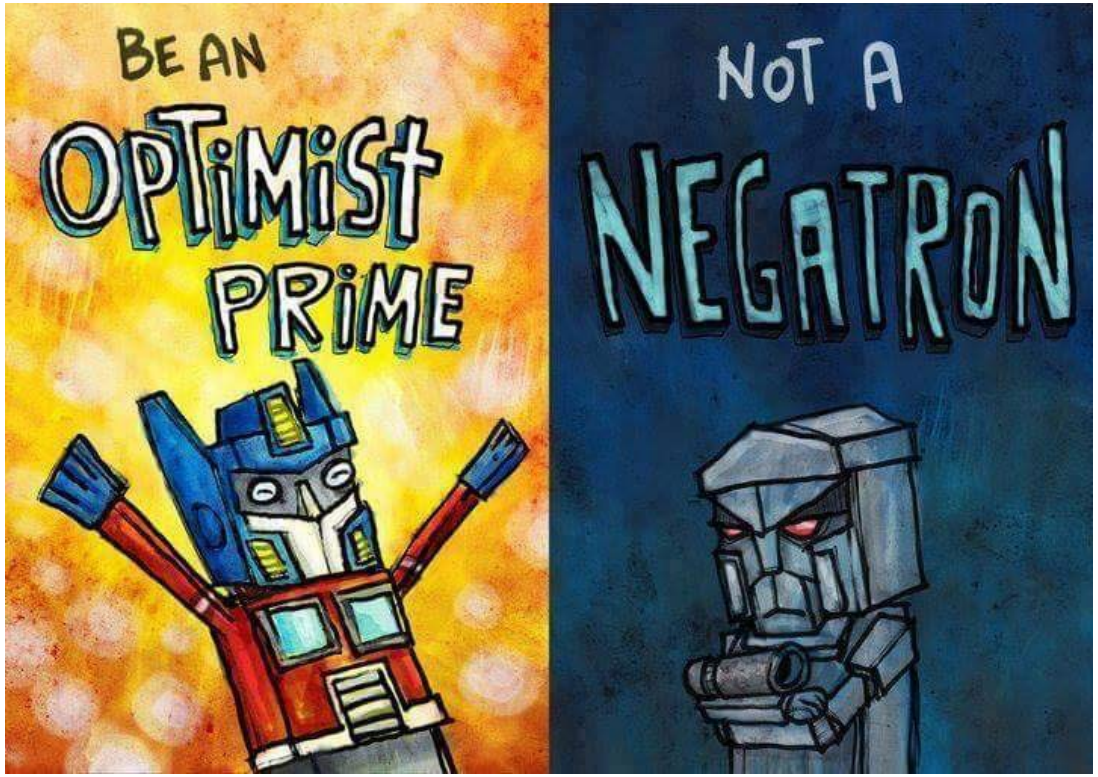
Choose m to be **prime**

- Avoid powers of 2 and powers of 10

In practice: popular and easy

But not always the most effective.

Slow (no more shifts)



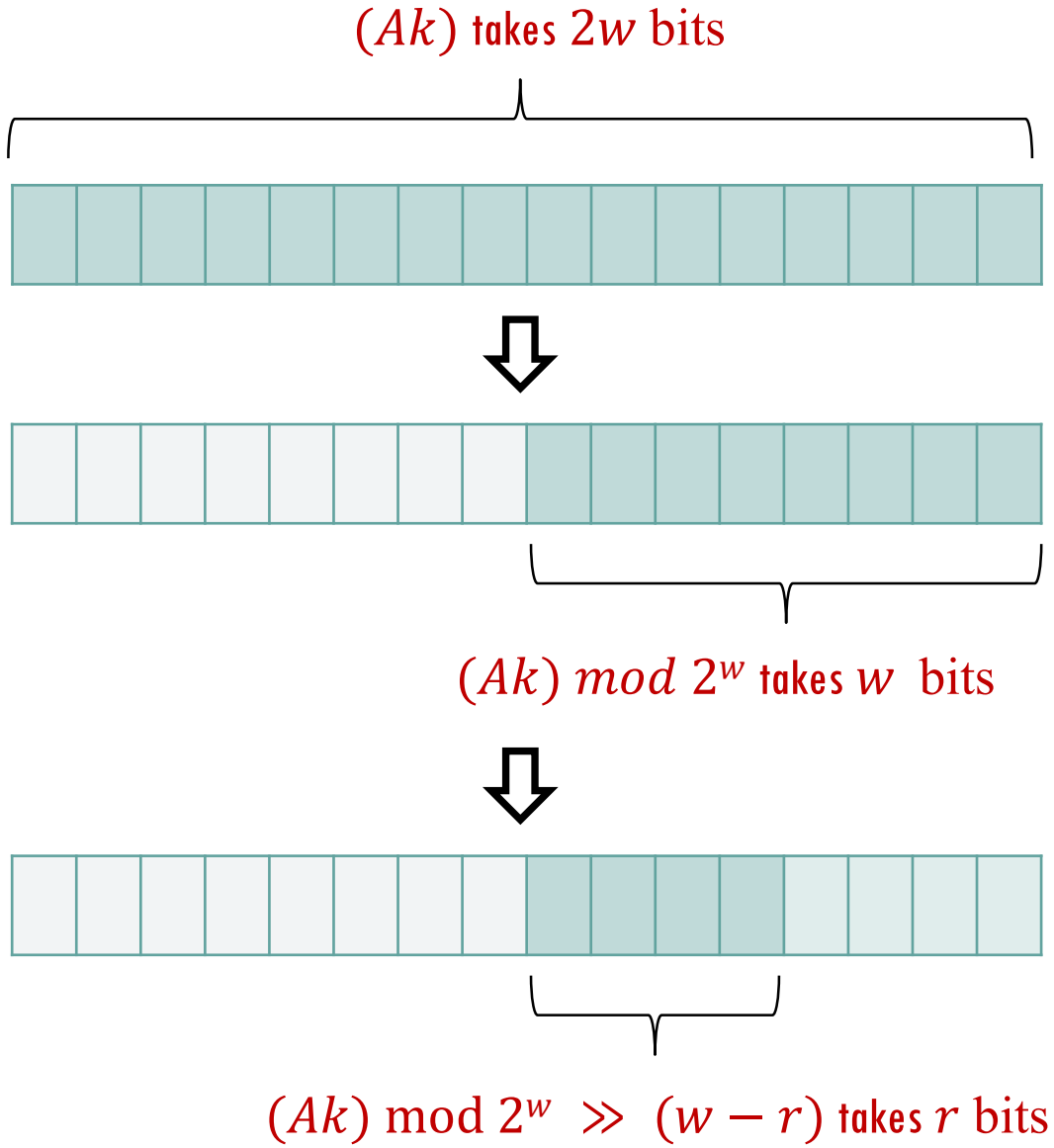
MULTIPLICATION METHOD

Fix

- table size: $m = 2^r$
- word size: w (size of a key in bits)
- constant: $2^{w-1} < A < 2^w$

Then:

$$h(k) = (Ak) \bmod 2^w \gg (w - r)$$

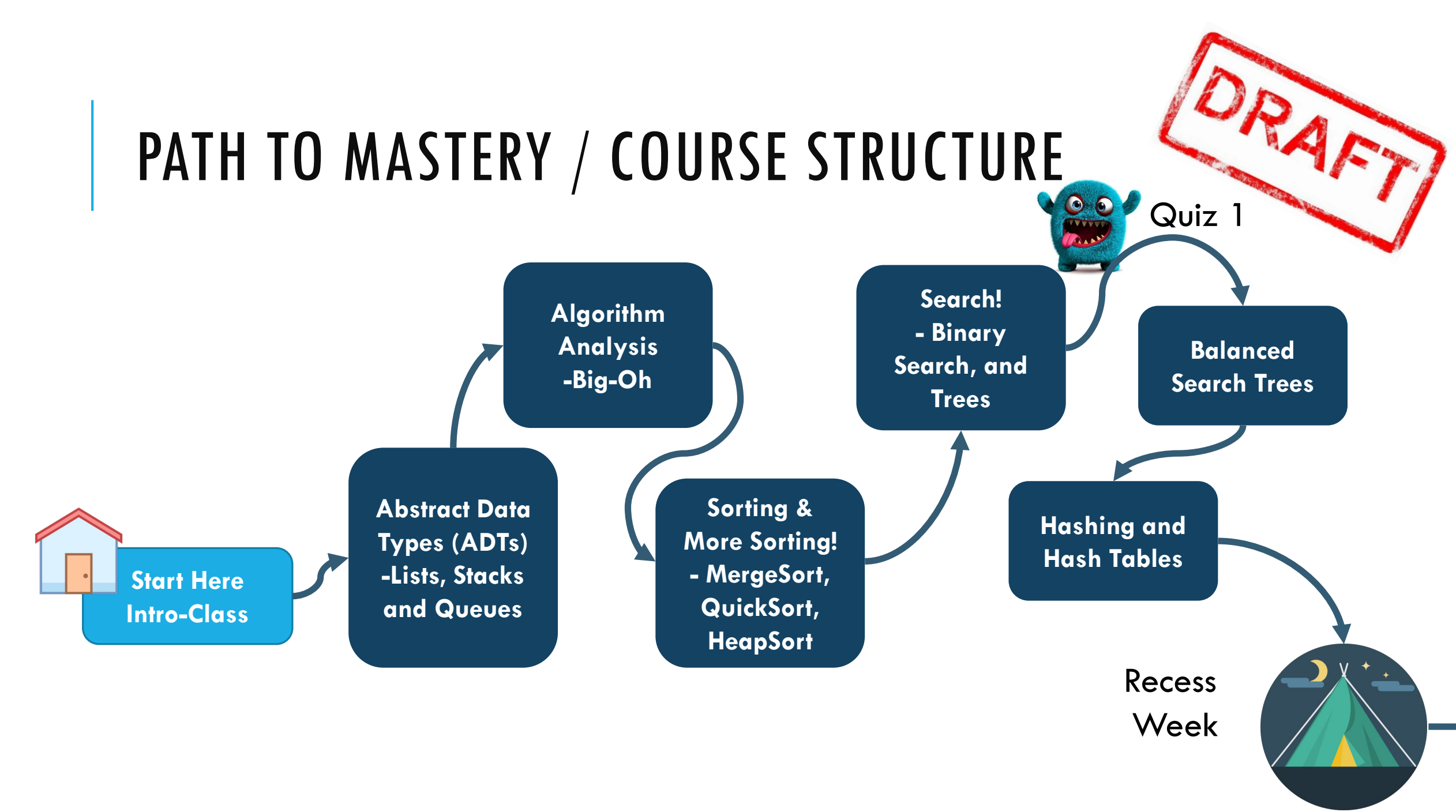


QUESTIONS?

Poll Everywhere

https://bit.ly/2LvG9bq

PATH TO MASTERY / COURSE STRUCTURE



PATH TO MASTERY / COURSE STRUCTURE

Intro to Graphs

Searching on Graphs! - Depth/Breadth-First

Topological Sorting

Single-Source Shortest Paths Dijkstra, Bellman-Ford, All-Pairs Shortest Paths

Minimum Spanning Trees (MST)

Bloom Filters, Cuckoo Hashing

YOU ARE HERE

Quiz 2

Quiz 3

Final Boss!

QUESTION

I have to store a clique. Which takes more space (in terms of big-O): an adjacency matrix or an adjacency list?

A. Adjacency List

B. Adjacency Matrix

C. They both take the same space

QUESTION

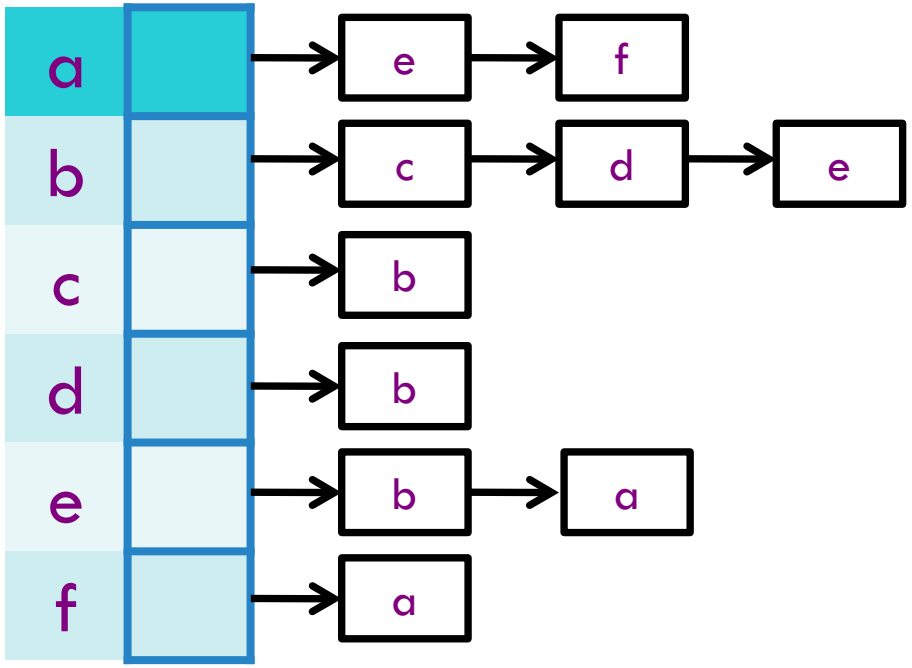
I have to store a clique. Which takes more space (in terms of big-O): an adjacency matrix or an adjacency list?

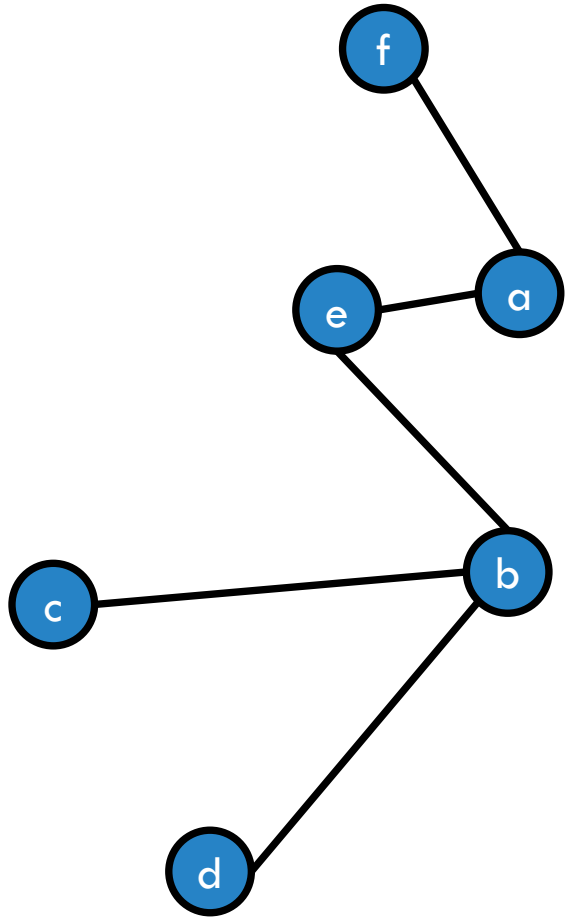
A. Adjacency List

B. Adjacency Matrix

C. **They both take the same space**

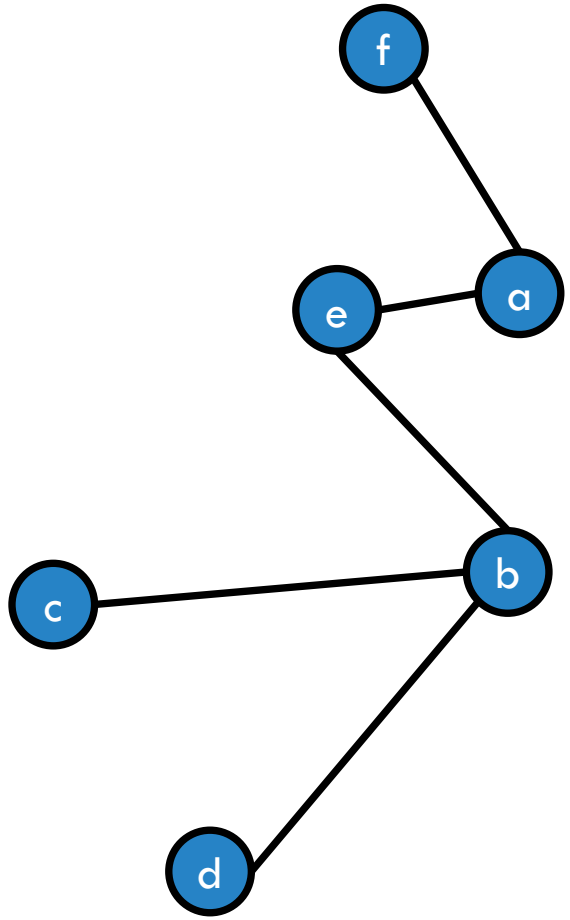
ADJACENCY LIST





ADJACENCY MATRIX

	a	b	c	d	e	f
a	0	0	0	0	1	1
b	0	0	1	1	1	0
c	0	1	0	0	0	0
d	0	1	0	0	0	0
e	1	1	0	0	0	0
f	1	0	0	0	0	0



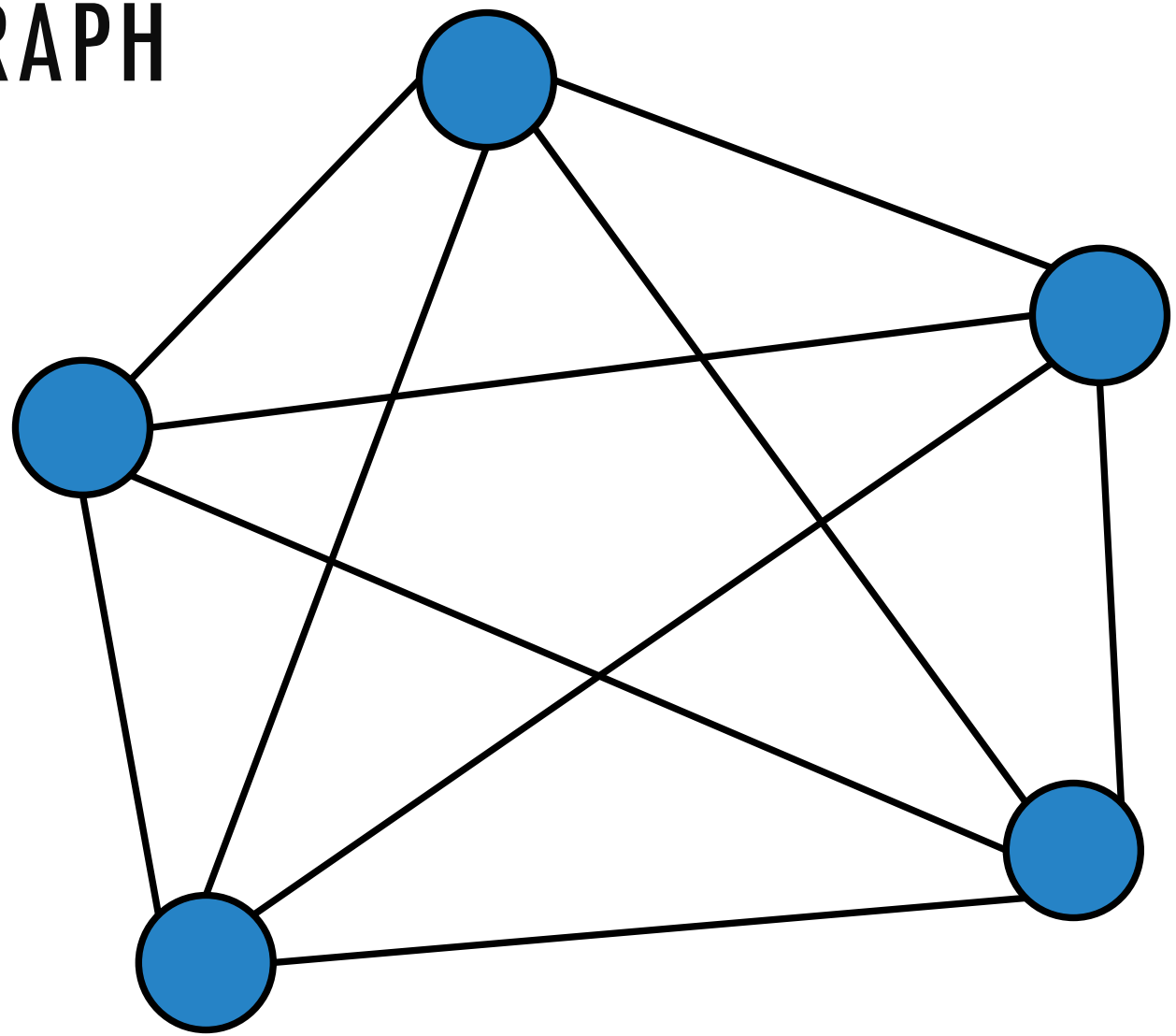
CLIQUE: A COMPLETE GRAPH

Fully connected

All pairs connected by edges

Question: In a clique, what is the longest path from one node to another?

One. Every node is only 1 hop from any other node. The **diameter** of a clique = 1



ADJACENCY **LIST**

Memory usage for graph G:

array of size $|V|$

linked lists of size $|E|$

Total: $O(V + E)$

For a cycle: $O(V)$

For a clique: $O(V^2)$

ADJACENCY **MATRIX**

Memory usage for graph G:

array of size $|V| \times |V|$

Total: $O(V^2)$

For a cycle: $O(V^2)$

For a clique: $O(V^2)$

QUESTION

I want to construct a graph of countries and flights connecting the countries. A very common operation I will have to execute is:

“Find me all flights departing from country X”

Which representation is preferred: adjacency list or adjacency matrix?

A. Adjacency List

B. Adjacency Matrix

QUESTION

I want to construct a graph of countries and flights connecting the countries. A very common operation I will have to execute is:

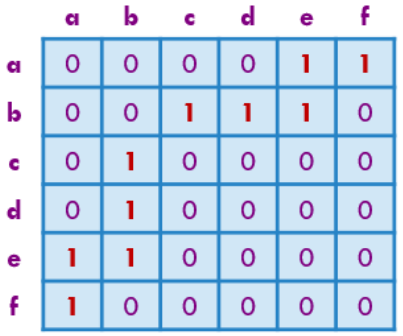
“Find me all flights departing from country X”

Which representation is preferred: adjacency list or adjacency matrix?

A. **Adjacency List**

B. Adjacency Matrix

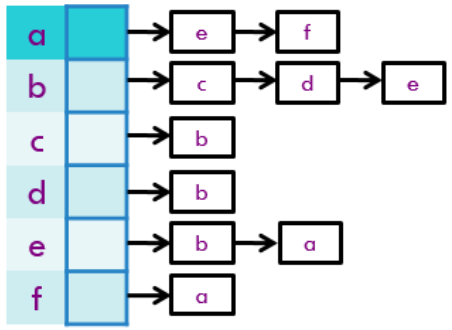
TRADE-OFFS



	a	b	c	d	e	f
a	0	0	0	0	1	1
b	0	0	1	1	1	0
c	0	1	0	0	0	0
d	0	1	0	0	0	0
e	1	1	0	0	0	0
f	1	0	0	0	0	0

Adjacency Matrix:

- Fast query: are v and w neighbors?
- Slow query: find me any neighbor of v.
- Slow query: enumerate all neighbors.



Adjacency List:

- Fast query: find me any neighbor.
- Fast query: enumerate all neighbors.
- Slower query: are v and w neighbors?

QUESTION

I'm in a grid-maze where I need to find the quickest route to the exit. I can only move left, right, up or down. Taking each step requires 2 seconds.

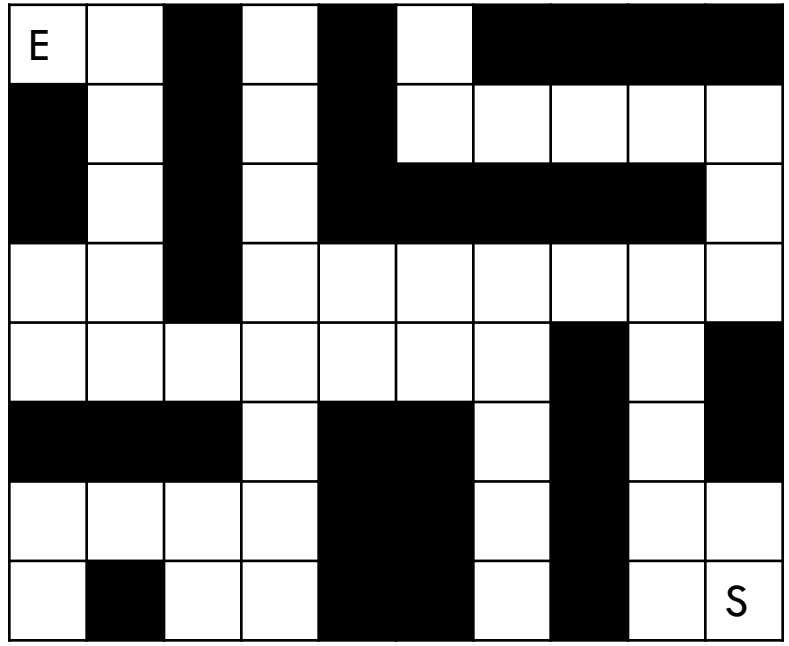
Which of the following algorithms is most relevant for solving this problem?

A. Breadth First Search

B. Depth First Search

C. Bellman-Ford

D. Floyd-Warshall



QUESTION

I'm in a grid-maze where I need to find the quickest route to the exit. I can only move left, right, up or down. Taking each step requires 2 seconds.

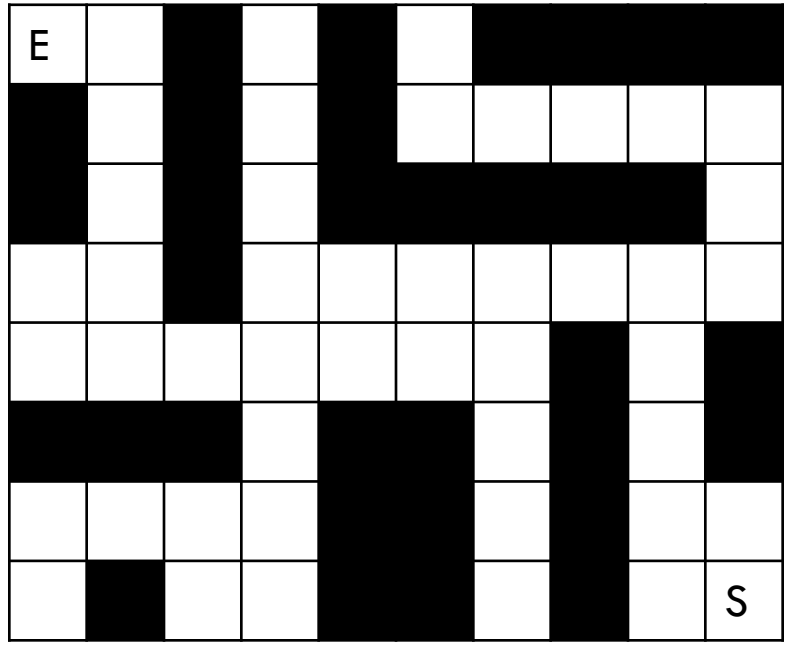
Which of the following algorithms is most relevant for solving this problem?

A. **Breadth First Search**

B. Depth First Search

C. Bellman-Ford

D. Floyd-Warshall



BREADTH-FIRST SEARCH

Explore level by level

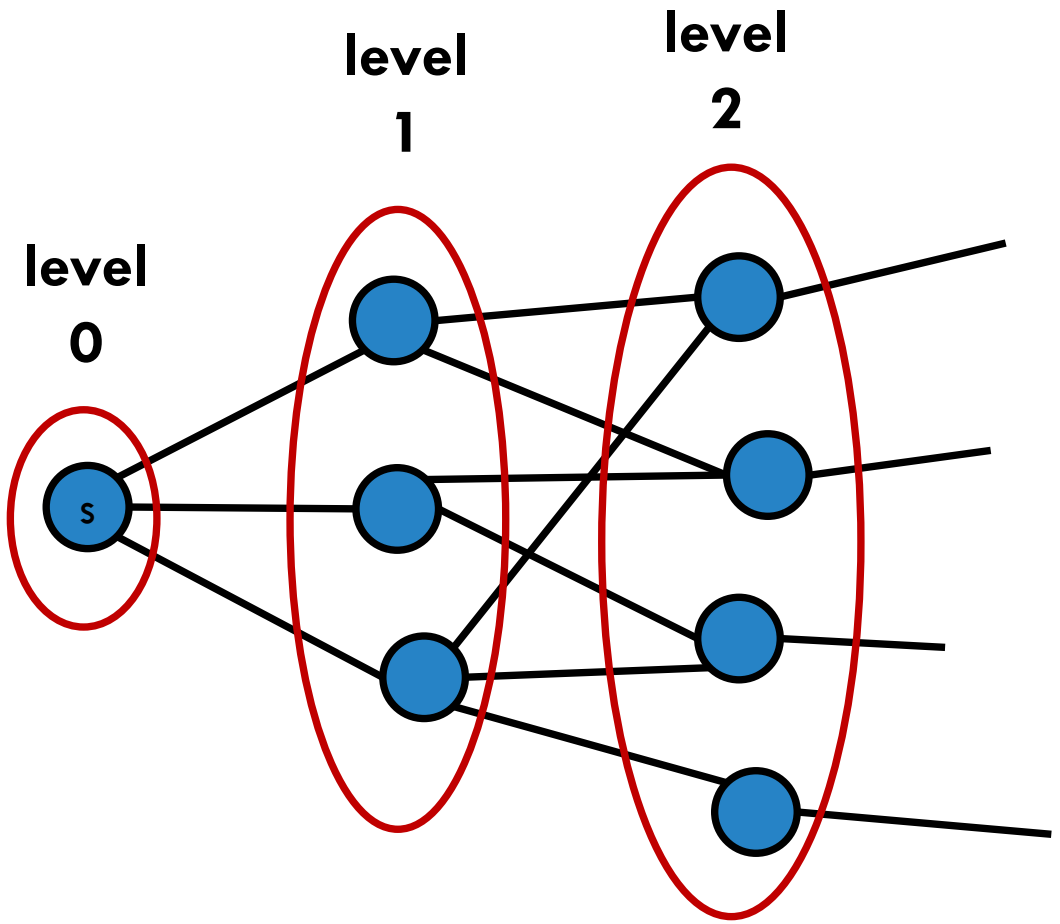
Frontier: current level

Initial frontier: {s}

Advance frontier.

Don't go backward!

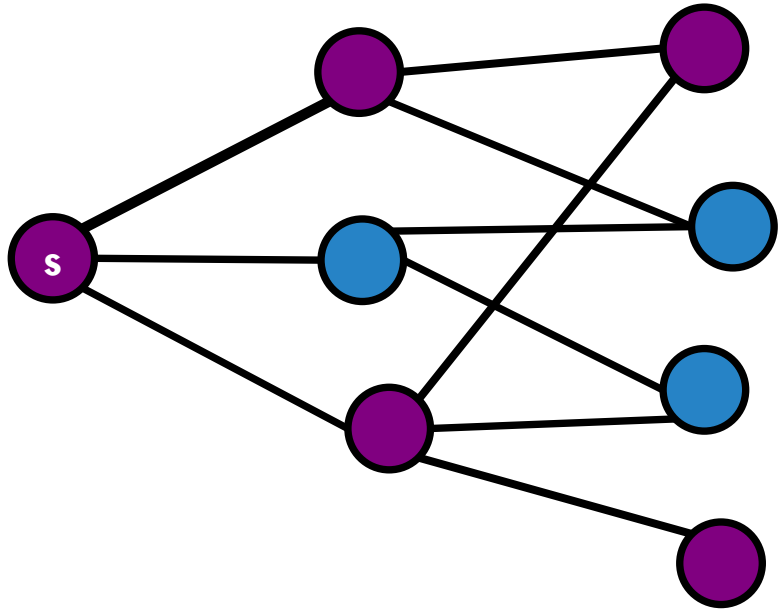
Finds shortest paths.



DEPTH-FIRST SEARCH (DFS)

Strategy

- Follow path until you get stuck
- Backtrack until you find a new edge
- Recursively explore it
- Don't repeat a vertex.



QUESTION

I have a pretty complicated list of things to study before the CS2040 exam. Some topics have pre-requisite topics (e.g., I should study algorithm analysis before I study graphs algorithms).

How can order the topics so that I can optimize my studying?

A. Breadth-First Search
B. Topological Sort
C. Dijkstra's Algorithm
D. None of the above.

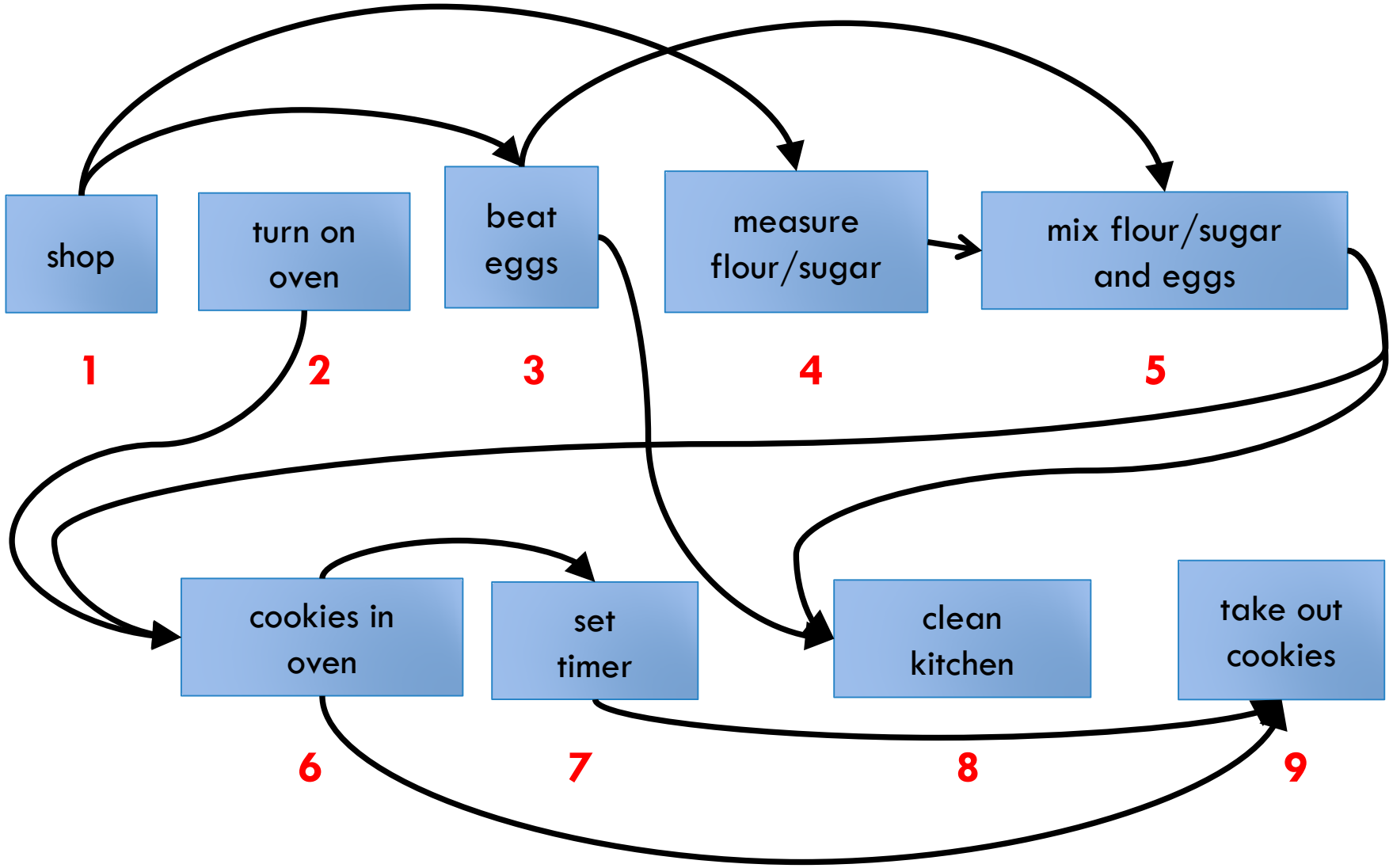
QUESTION

I have a pretty complicated list of things to study before the CS2040 exam. Some topics have pre-requisite topics (e.g., I should study algorithm analysis before I study graphs algorithms).

How can order the topics so that I can optimize my studying?

A. Breadth-First Search
B. **Topological Sort**
C. Dijkstra's Algorithm
D. None of the above.

TOPOLOGICAL ORDERING



KAHN'S ALGORITHM

Assume adjacency list

Start at any node v with no incoming edges.

Add v to our list

Remove v and all its outgoing edges.

Repeat

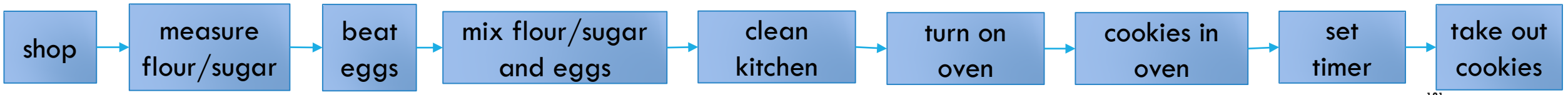
Pseudocode:

```
L = list()
S = list()
add all nodes with no incoming edge to S
while S is not empty:
    remove node v from S
    add v to tail of L
    for each of v's neighbors u
        remove edge e where source is v
        if u has no other incoming edges
            add u to S
```

$O(V)$

$O(E)$

What is the time complexity? $O(V + E)$



TOPOLOGICAL SORT USING DFS (ASSUME DAG)

Idea: Process node when it is "last" visited.

```
L = list()
while there are unvisited nodes
    v = select unvisited node
    DFS(G, v, L)
```

```
DFS(G,v,L)
    if v is visited
        return
    else
        for each of v's neighbor u
            DFS(G, u, L)
    visit(v)
    L.pushFront(v)
```

QUESTION

The code for relaxation is:

```
relax(int u, int v){
      if (dist[v] < dist[u] + weight(u,v))
         dist[v] = dist[u] + weight(u,v);
}
```

A. True

B. False

QUESTION

The code for relaxation is:

```
relax(int u, int v){
      if (dist[v] < dist[u] + weight(u,v))
         dist[v] = dist[u] + weight(u,v);
}
```

A. True

B. **False**

SHORTEST PATHS

```
relax(int u, int v){
      if (dist[v] > dist[u] + weight(u,v))
         dist[v] = dist[u] + weight(u,v);
}
```

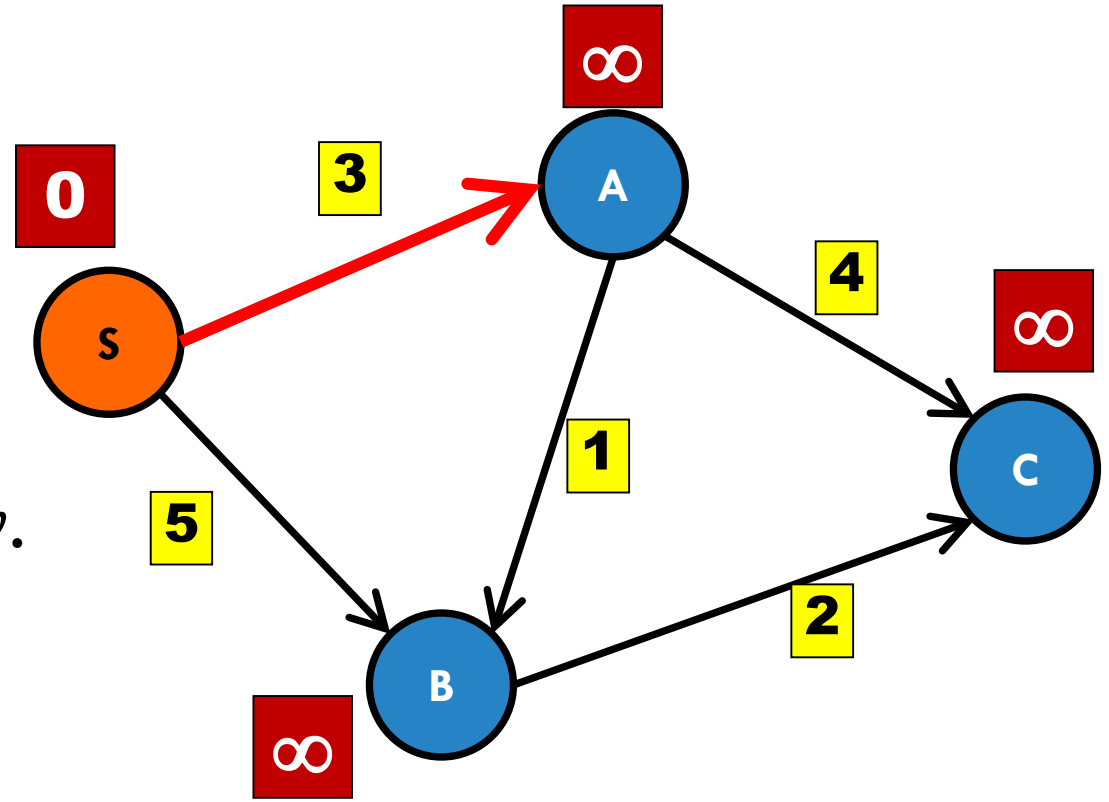
Maintain estimate for each distance:

relax(S, A)

The idea:

relax(w,v):
Test if the best way to get from $s \rightarrow v$ is to go from $s \rightarrow w$, then $w \rightarrow v$.

Update `dist`



SHORTEST PATHS

```
relax(int u, int v){
      if (dist[v] > dist[u] + weight(u,v))
         dist[v] = dist[u] + weight(u,v);
}
```

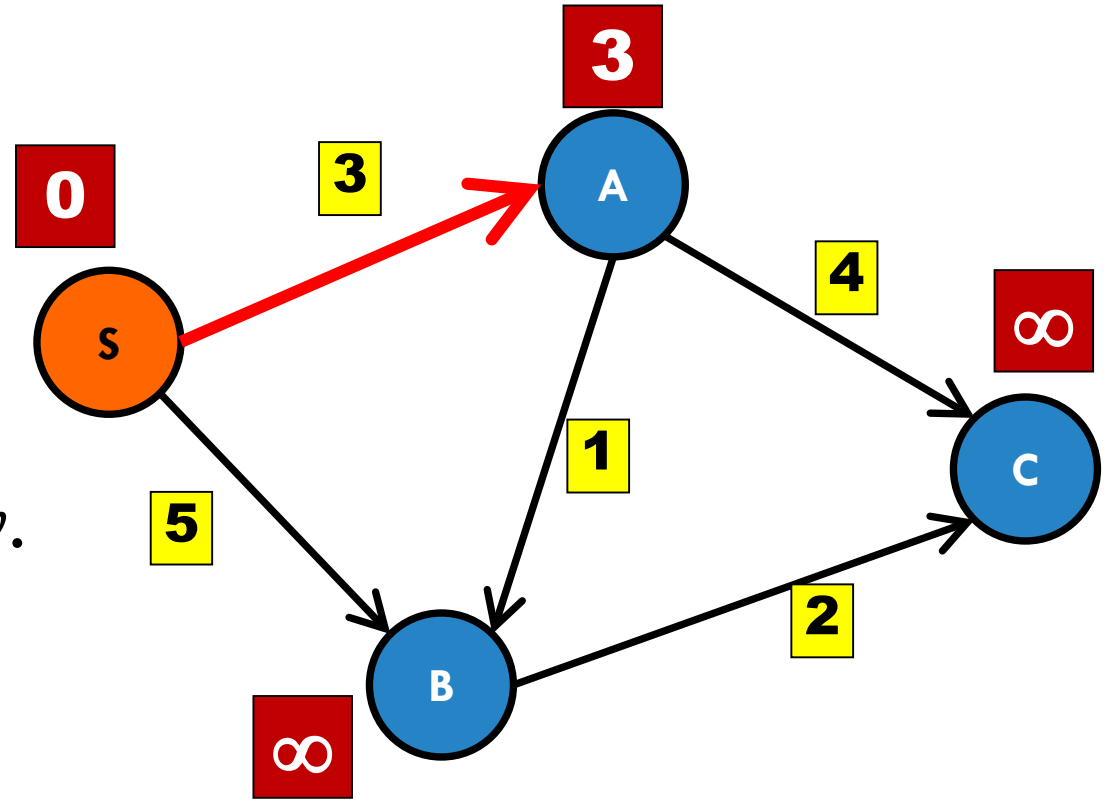
Maintain estimate for each distance:

relax(S, A)

The idea:

relax(w,v):
Test if the best way to get from $s \rightarrow v$ is to go from $s \rightarrow w$, then $w \rightarrow v$.

Update `dist`



QUESTION

Who first invented the Bellman-Ford algorithm?

A. Alfonso Shimbel
B. Richard Bellman
C. Lester Ford
D. Edward Moore

QUESTION

Who first invented the Bellman-Ford algorithm?

A. **Alfonso Shimbel**
B. Richard Bellman
C. Lester Ford
D. Edward Moore

BELLMAN-FORD

Bellman–Ford algorithm

From Wikipedia, the free encyclopedia

The **Bellman–Ford algorithm** is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph.[1] It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers. The algorithm was first proposed by Alfonso Shimbel in 1955, but is instead named after Richard Bellman and Lester Ford, Jr., who published it in 1958 and 1956, respectively.[2] Edward F. Moore also published the same algorithm in 1957, and for this reason it is also sometimes called the **Bellman–Ford–Moore algorithm**.[1]

QUESTION

Poor Shimbel. But he's in good company: many things aren't named after their original discoverers. What "law" describes this phenomena?

A. Merton's Law
B. Soh's Law
C. Stigler's Law
D. Moore's Law

QUESTION

Poor Shimbel. But he's in good company: many things aren't named after their original discoverers. What "law" describes this phenomena?

A. Merton's Law
B. Soh's Law
C. **Stigler's Law**
D. Moore's Law

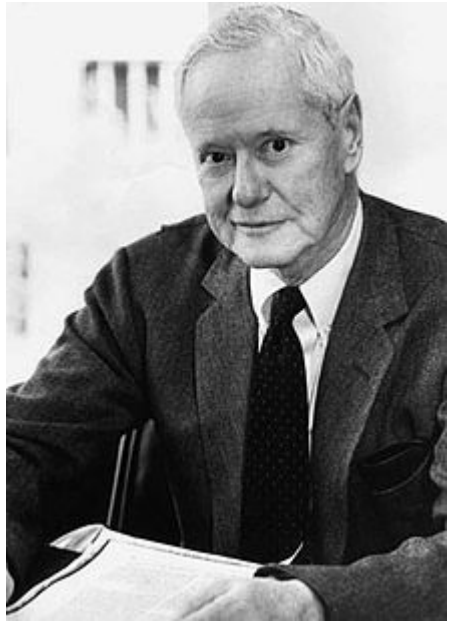
STIGLER'S LAW OF EPONYMY

No scientific discovery is named after its original discoverer.

proposed by University of Chicago statistics professor Stephen Stigler.

Stigler named Columbia University sociology professor **Robert K. Merton** as the original discoverer of “Stigler's law”

Stigler’s law follows Stigler’s law.



QUESTION

True/False: Bellman-Ford involves relaxing edges and *requires* relaxing all the edges for |V|-1 iterations.

A. True

B. False

QUESTION

True/False: Bellman-Ford involves relaxing edges and *requires* relaxing all the edges for |V|-1 iterations.

A. True

B. **False**

EARLY TERMINATION?

```
n = V.length
for i = 1 to n-1
  for Edge e in Graph
      relax(e)
```

When can we terminate early?

A. When a relax operation has no effect.
B. When two consecutive relax operations have no effect.
C. When an entire sequence of |E| relax operations have no effect.
D. Never. Only after |V| complete iterations.

QUESTION

I want to get from SoC to ION Orchard so that I can get some LeTAO cheesecakes.

Which algorithm should I use to find the shortest path (distance)?

A. Bellman-Ford

B. Dijkstra's Algorithm

C. Floyd Warshall

D. Breadth-First Search

QUESTION

I want to get from SoC to ION Orchard so that I can get some LeTAO cheesecakes.

Which algorithm should I use to find the shortest path (distance)?

A. Bellman-Ford

B. **Dijkstra's Algorithm**

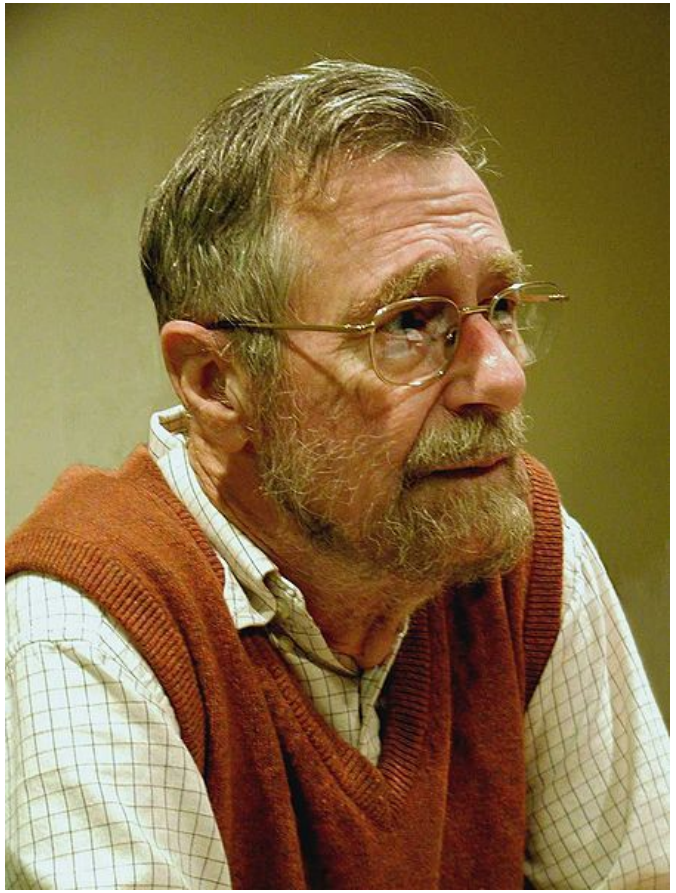
C. Floyd Warshall

D. Breadth-First Search

DIJKSTRA'S ALGORITHM

Basic idea:

- Maintain distance **estimate** for every node.
- Begin with empty shortest-path-tree.
- Repeat:
  - Consider vertex with minimum **estimate**.
  - Add vertex to **shortest-path-tree**.
  - Relax all outgoing edges.



1930-2002

QUESTION

Which ADT and data structure should I use to store the vertices for use in the standard Dijkstra's algorithm?

A. Priority Queue (Heap)

B. Priority Queue (AVL Tree)

C. Priority Queue (Linked List)

D. Stack (Array)

E. Stack (AVL Tree)

QUESTION

Which ADT and data structure should I use to store the vertices for use in the standard Dijkstra's algorithm?

A. Priority Queue (Heap)

B. **Priority Queue (AVL Tree)**

C. Priority Queue (Linked List)

D. Stack (Array)

E. Stack (AVL Tree)

DIJKSTRA'S ALG: PERFORMANCE

PQ Implementation	insert	deleteMin	decreaseKey	Total
Array	1	V	1	$O(V^2)$
AVL Tree	$\log V$	$\log V$	$\log V$	$O((V+E)log\,V)$
d-way Heap	$d\log_d V$	$d\log_d V$	$\log_d V$	$O(E\log_{\frac{E}{V}} V)$
Fibonacci Heap	1	$\log V$	1	$O(E + V\,log\,V)$

QUESTION

I give you two identical graphs G and G' that differ only in the following respect: for every edge $e \in G$, the corresponding graph $e' \in G'$ has weight $w(e') = w(e)^2$

Then every shortest path in G is also a shortest path in G'.

True or False?

A. True

B. False

QUESTION

I give you two identical graphs G and G' that differ only in the following respect: for every edge $e \in G$, the corresponding graph $e' \in G'$ has weight $w(e') = w(e)^2$

Then every shortest path in G is also a shortest path in G'.

True or False?

A. True

B. **False**

QUESTION

Given a graph G, if I partition the nodes into two subsets, the maximum edge across the cut is never in the MST.

A. True

B. False

QUESTION

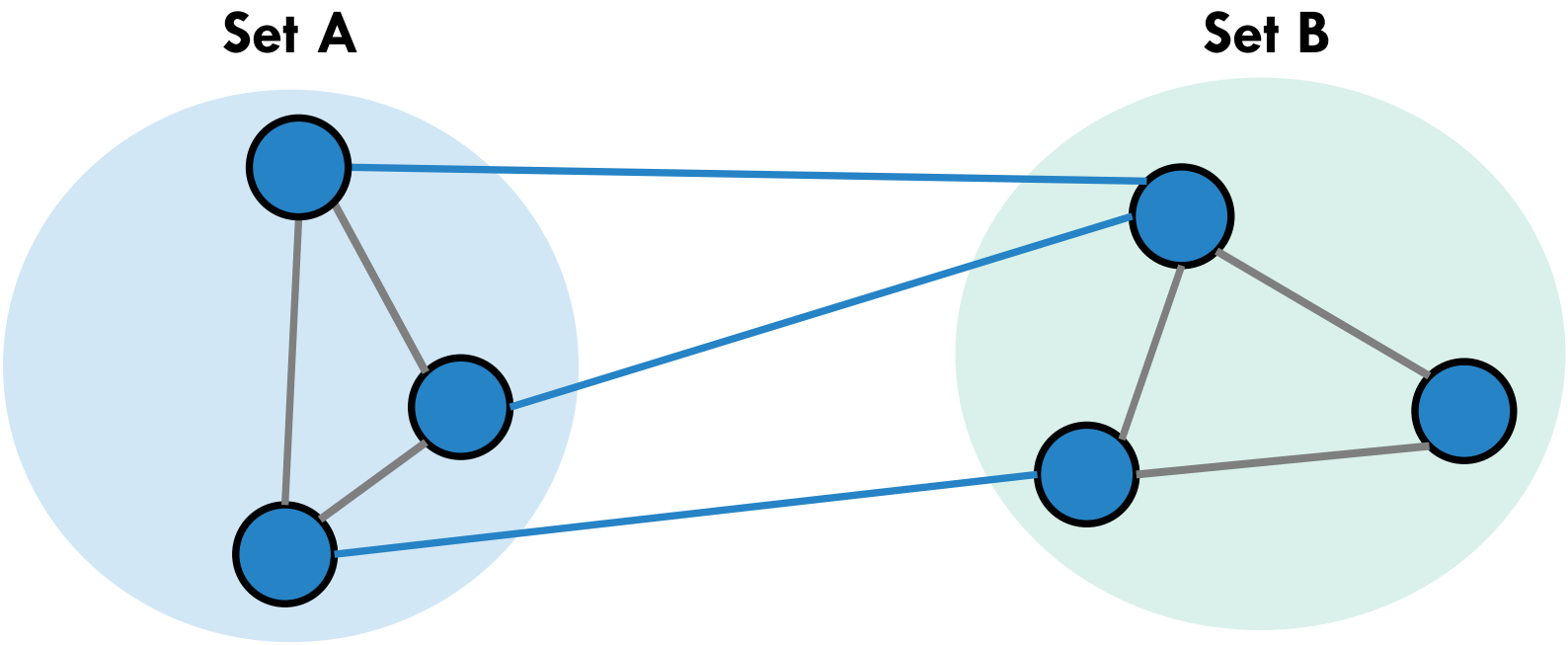
Given a graph G, if I partition the nodes into two subsets, the maximum edge across the cut is never in the MST.

A. True

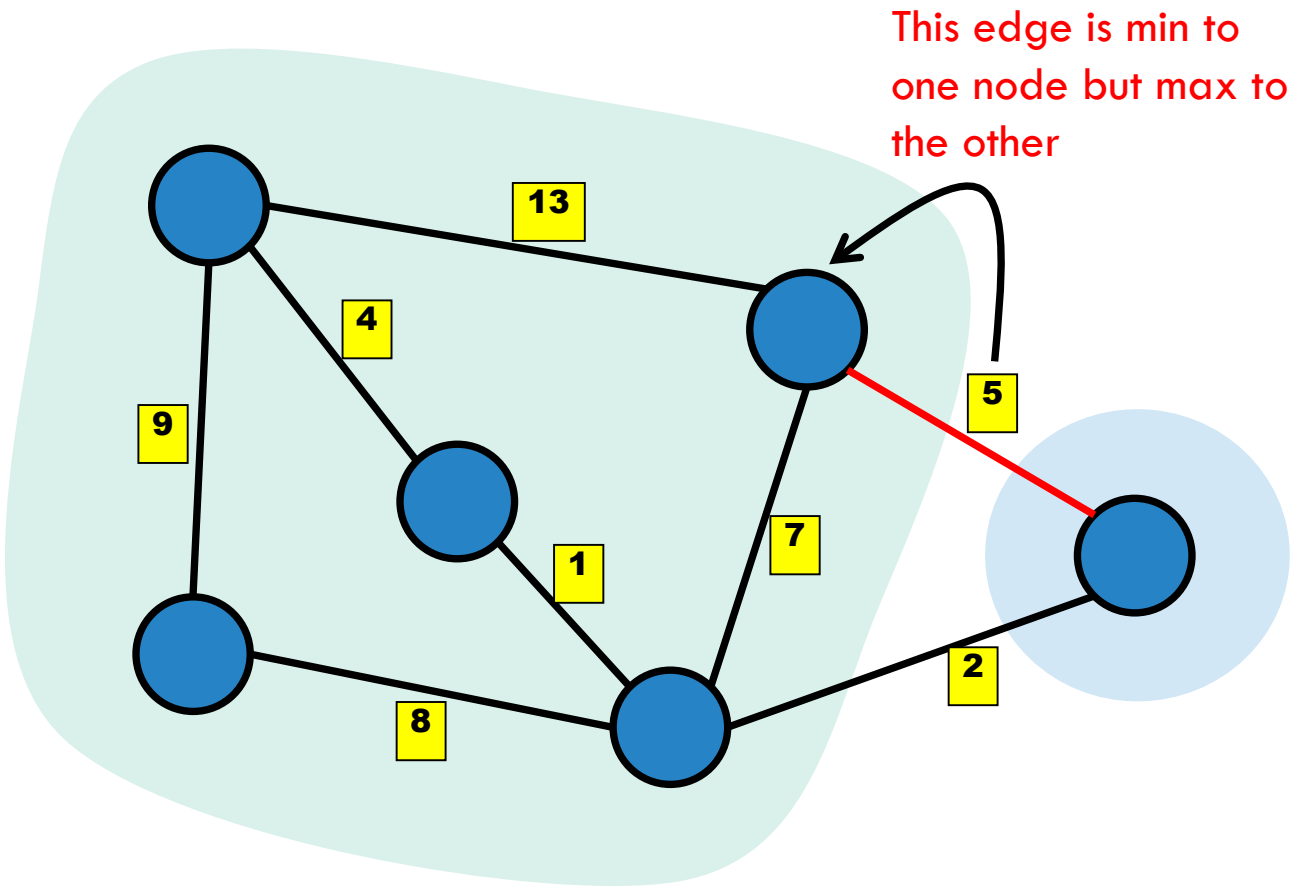
B. **False**

WHAT IS A CUT?

Definition: A ***cut*** of a graph $G = (V, E)$ is a partition of the vertices V into two disjoint subsets.



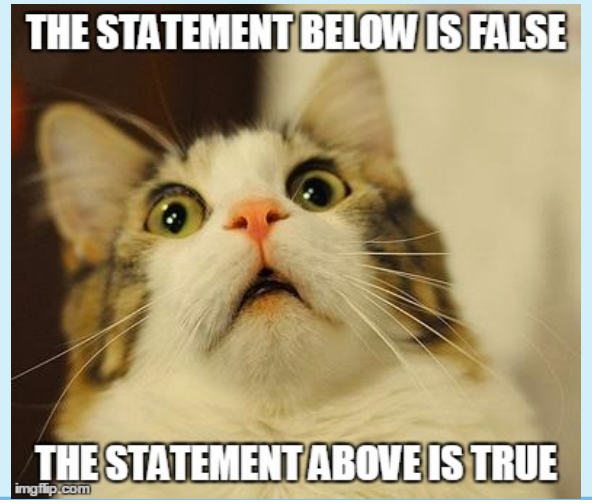
TRUE OR FALSE?



For every vertex, the maximum incident/outgoing edge is **never** part of the MST.

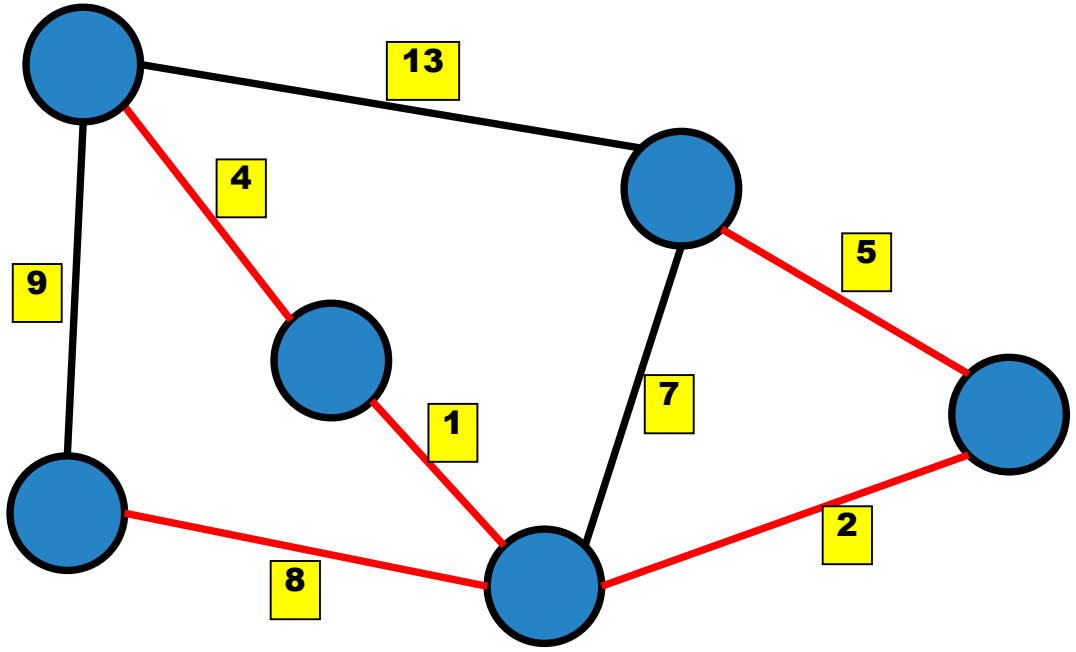
A. True! You're not tricking me again!

B. False.

C. 

MST PROPERTIES: SUMMARY

1. No cycles
2. If you cut an MST, the two pieces are both MSTs.
3. Cycle property
   - For every cycle, the maximum weight edge is not in the MST.
4. Cut property
   - For every cut D, the minimum weight edge that crosses the cut is in the MST.



QUESTION

I have a graph with edges already sorted by their weights. To construct a MST, which algorithm is most relevant?

A. Kruskal's algorithm

B. Prim's algorithm

C. Floyd-Warshall

D. Dijkstra's Shortest Path Algorithm

QUESTION

I have a graph with edges already sorted by their weights. To construct a MST, which algorithm is most relevant?

A. **Kruskal's algorithm**

B. Prim's algorithm

C. Floyd-Warshall

D. Dijkstra's Shortest Path Algorithm

KRUSKAL'S ALGORITHM

Basic idea:

- Graph F : a set of trees (initially each vertex is a separate tree)
- Set of edges $S = \{e \in E\}$
- While S is nonempty and F is not spanning:
  - Remove minimum weight edge from S
  - If removed edge connects two trees
    - add it to the F (combine the trees)

Sort takes $O(E \log E)$

$$O(E \log E) = O(E \log V^2) = O(E \log V)$$

For E edges, find/union take $O(E\alpha)$

Total: $O(E \log V)$

REVIEW: PRIM'S ALGORITHM

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: $S = \{A\}$
- Repeat:
  - Identify cut: $\{S, V-S\}$
  - Find minimum weight edge on cut.
  - Add new node to S.

Prim's "grows" the tree one node at a time.

Analysis:

Each vertex added/removed once from the priority queue:

$$O(V \log V)$$

Each edge → one decreaseKey:

$$O(E \log V)$$

QUESTION

Let $G = (V, E)$ be a weighted undirected graph and T be a MST of G. For any pair of vertices u and v, the path between u and v in T must be the shortest path between u and v in G.

Is the above statement True or False?

A. True

B. False

QUESTION

Let $G = (V, E)$ be a weighted undirected graph and T be a MST of G. For any pair of vertices u and v, the path between u and v in T must be the shortest path between u and v in G.

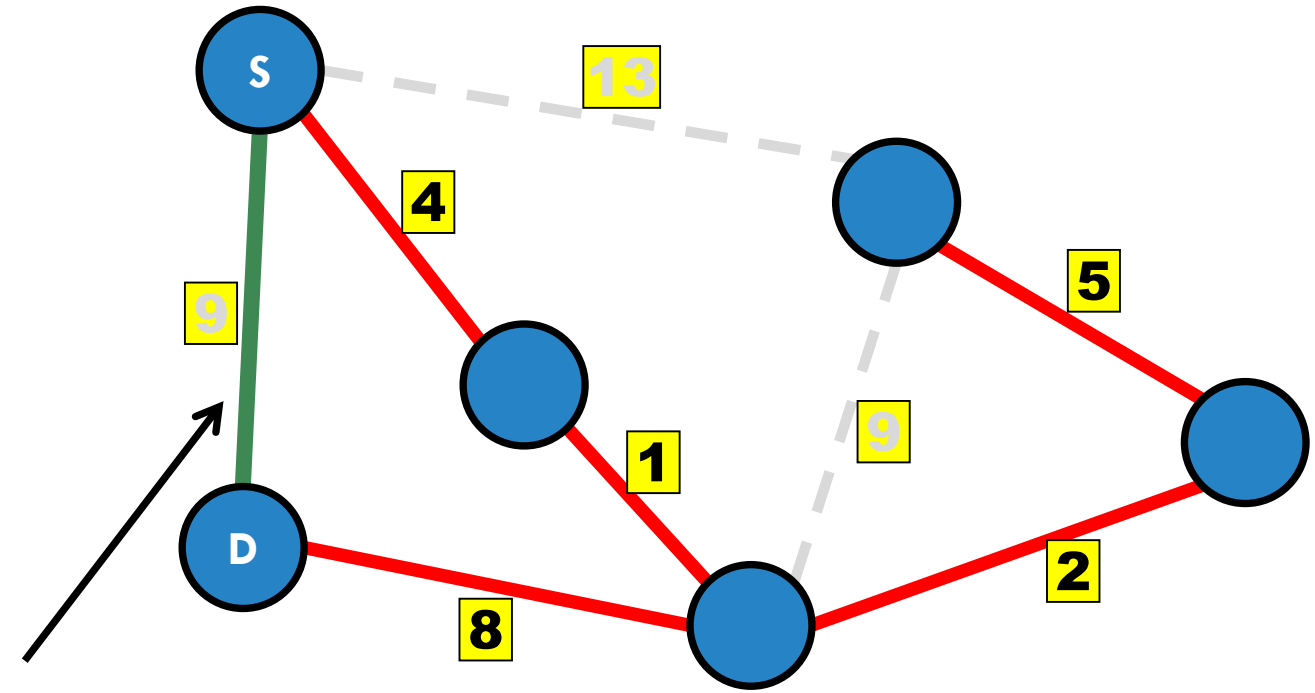
Is the above statement True or False?

A. True

B. **False**

MINIMUM SPANNING TREE

MST is not the same as shortest paths



Shortest path from S to D

Can we use the MST to find the shortest paths?

A. Yes
B. Only on connected graphs
C. Only on dense graphs
D. Nope.
E.

QUESTION

The depth of a BFS tree on an undirected graph from an arbitrary vertex v is the diameter of the graph.

True or False?

A. True

B. False

QUESTION

The depth of a BFS tree on an undirected graph from an arbitrary vertex v is the diameter of the graph.

True or False?

A. True

B. **False**

COUNTER-EXAMPLE

QUESTION

You work for Foogle Maps. A very frequent query on Foogle Maps is "*How long does it take to get from A to B?*" where A and B are two arbitrary locations on the map. What algorithm is most relevant for this problem?

A. Dijkstra's Algorithm
B. Breadth-First Search
C. Floyd-Warshall
D. Bellman-Ford
E. Kruskal's Algorithm

QUESTION

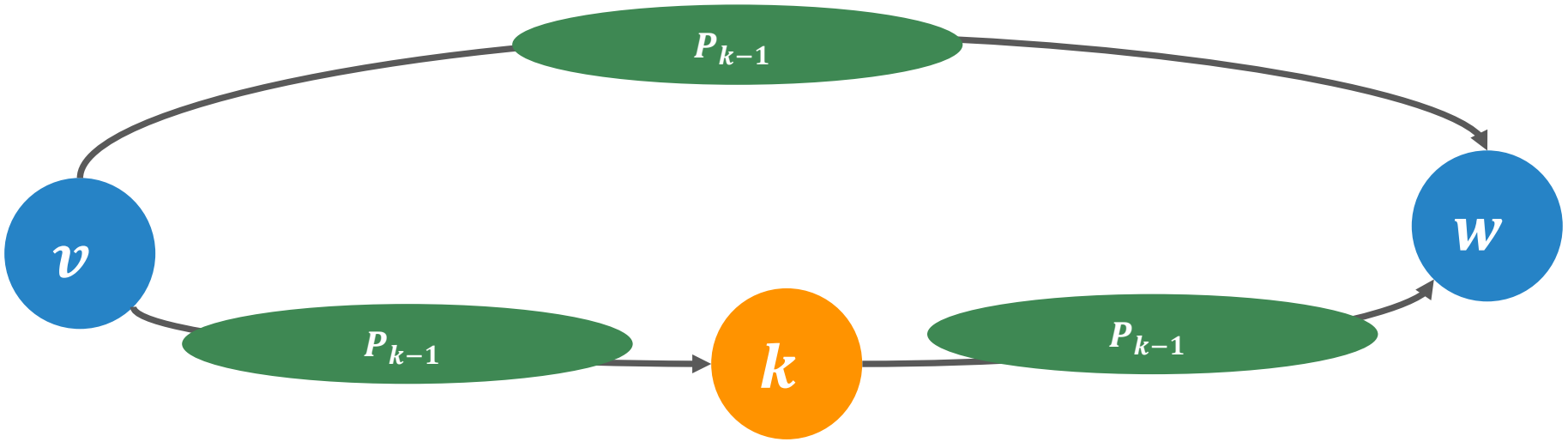
You work for Foogle Maps. A very frequent query on Foogle Maps is "*How long does it take to get from A to B?*" where A and B are two arbitrary locations on the map. What algorithm is most relevant for this problem?

A. Dijkstra's Algorithm
B. Breadth-First Search
C. **Floyd-Warshall**
D. Bellman-Ford
E. Kruskal's Algorithm

FLOYD-WARSHALL

Use the **precalculated** subproblems:

$$S[v, w, P_k] = \min(\ S[v, w, P_{k-1}],\quad S[v, k, P_{k-1}] + S[k, w, P_{k-1}]\ )$$



FLOYD-WARSHALL: PSEUDOCODE

```
Function FloydWarshall(G)
    S = Array of size |V|x|V| //memoization table S has |V| rows and |V| columns

    // Initialize every pair of nodes
    for v = 0 to |V|-1
        for w = 0 to |V|-1
            S[v,w] = E[v,w]

    // For sets P0, P1, P2, P3, …, for every pair (v,w)
    for k = 0 to |V|-1
        for v = 0 to |V|-1
            for w = 0 to |V|-1
                S[v,w] = min(S[v,w], S[v,k]+S[k,w])
    return S
```

What is the running time? $O(V^3)$

QUESTION

When is the CS2040 exam?

A. 5-Dec-2019 (9am)
B. 4-Dec-2019 (9am)
C. 3-Dec-2019 (9am)
D. 2-Dec 2019 (12pm)

QUESTION

When is the CS2040 exam?

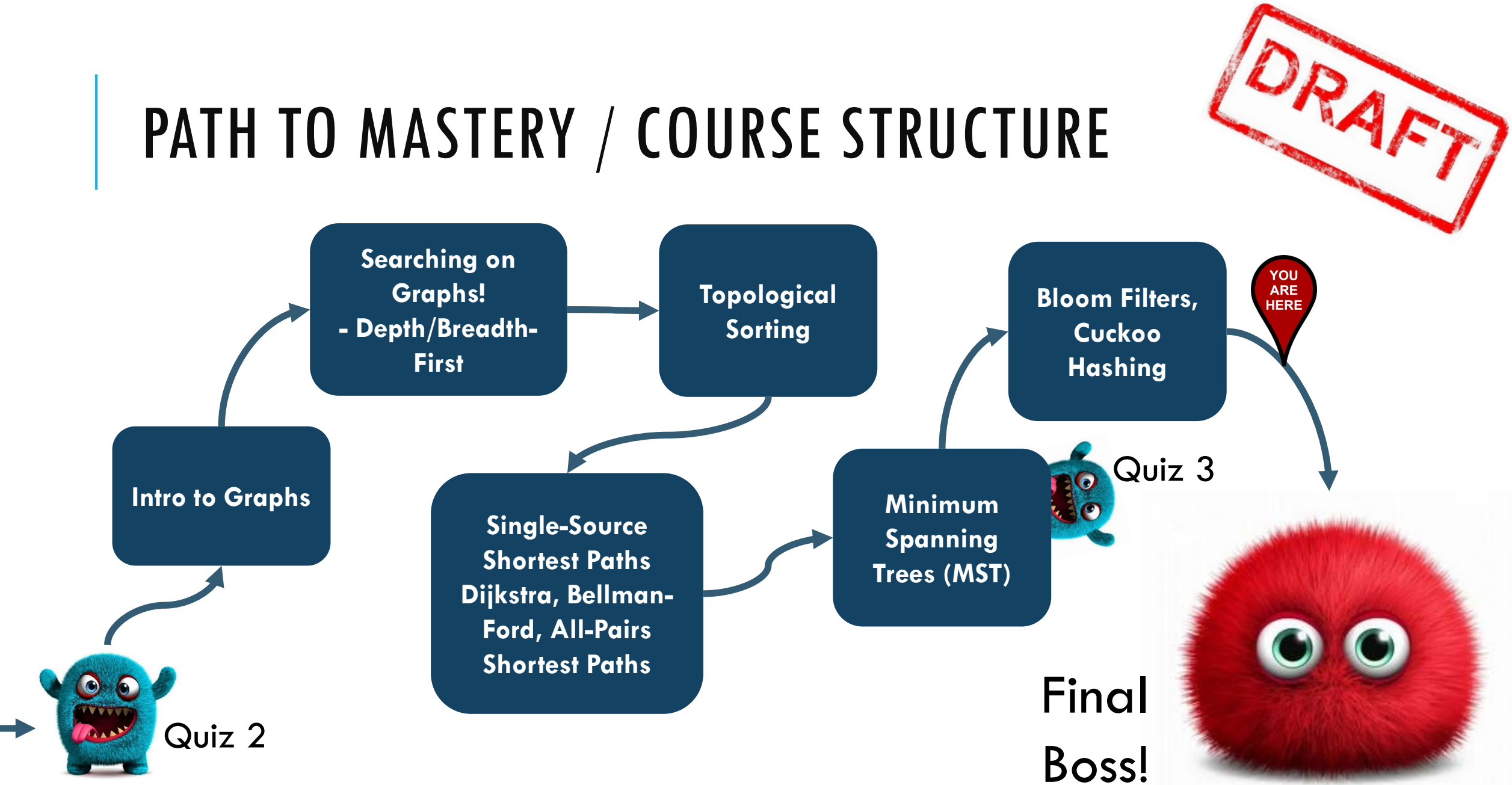
A. 5-Dec-2019 (9am)

B. 4-Dec-2019 (9am)

C. 3-Dec-2019 (9am)

D. 2-Dec 2019 (12pm)

PATH TO MASTERY / COURSE STRUCTURE



~~QUIZ~~ FINAL ADVICE

Don't panic!

Go through Visualgo and the lecture notes.

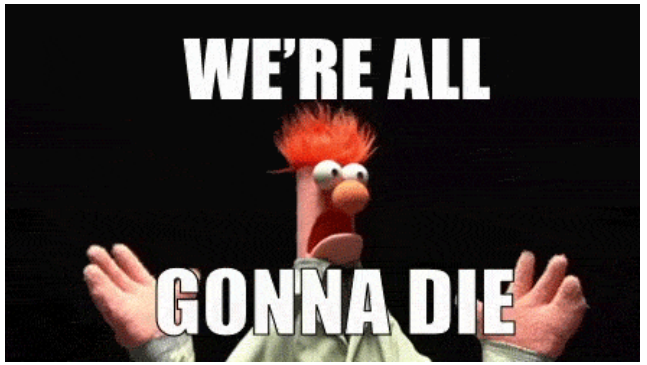
Sleep well the night before.

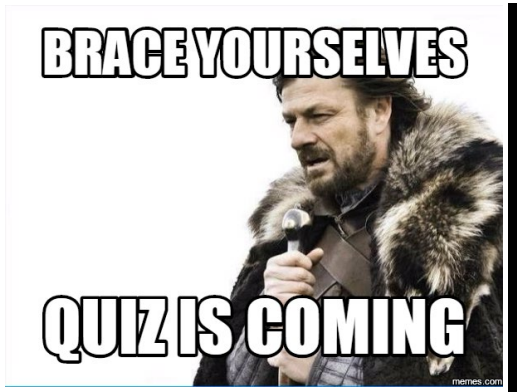
Sometimes, questions just *look* hard.

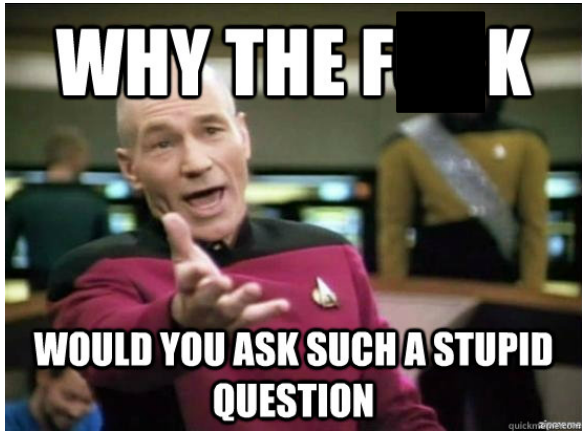
If you are unsure about something, raise your hand and ask.

Many questions will require some on-the-spot thinking.

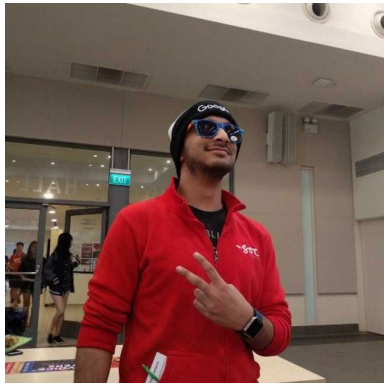
Pick the best answer.

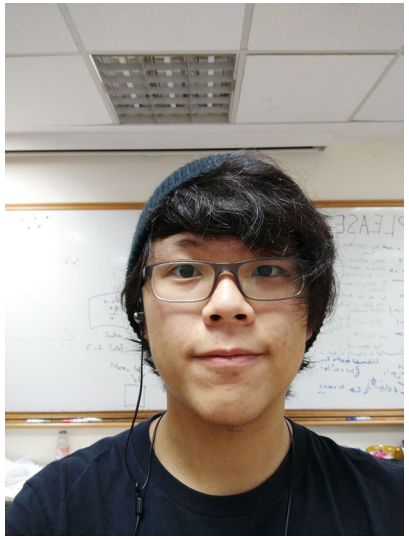


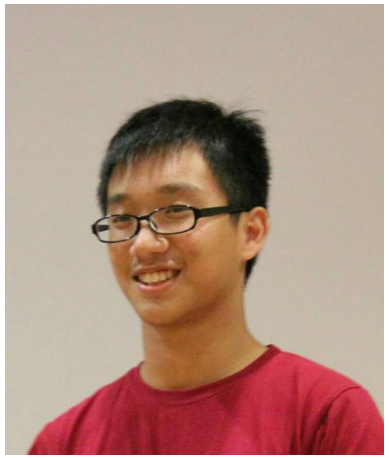
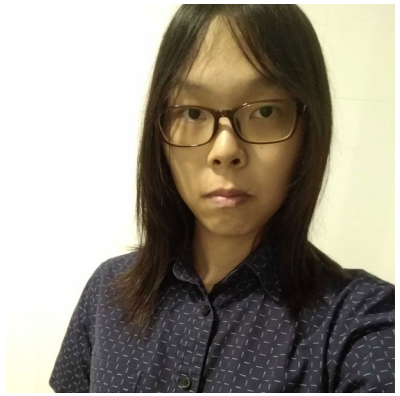


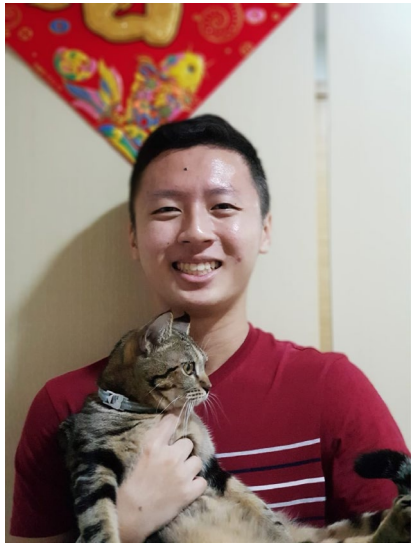


QUESTIONS?

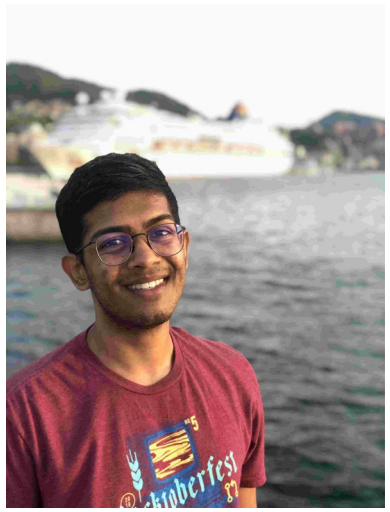










Thank you!!!

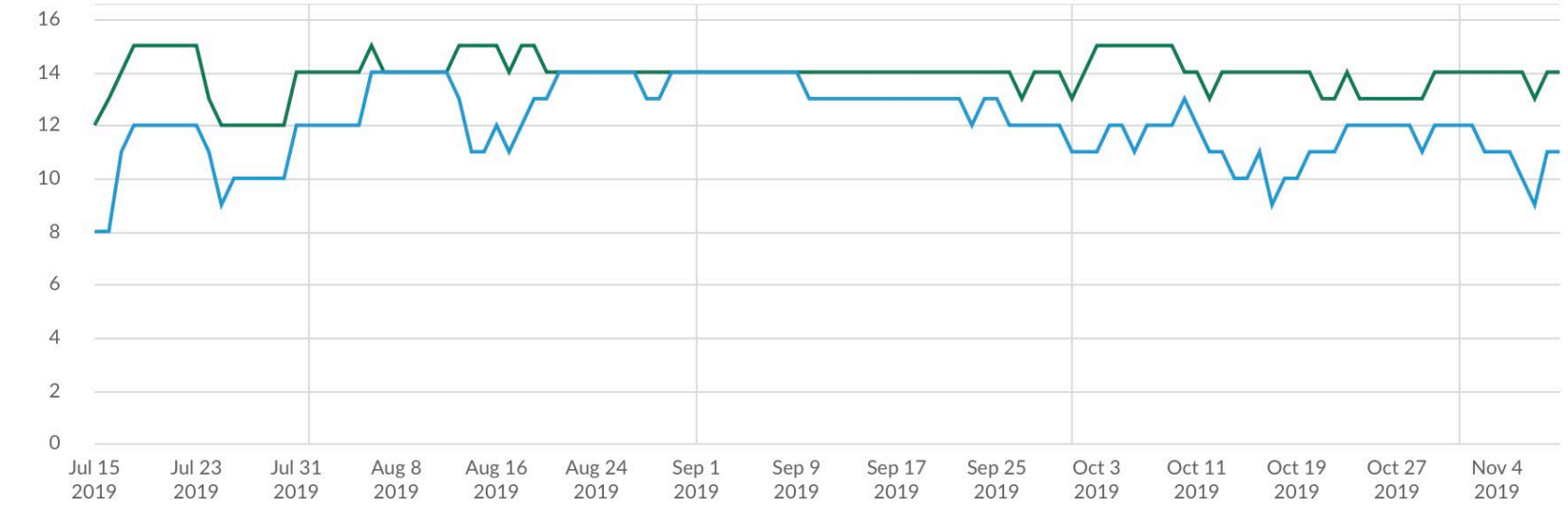
BEHIND THE SCENES

On Slack

Active members

See how many people are active — meaning they viewed at least one public channel.

Weekly Daily



On Piazza

Class At A Glance

119	total posts*
941	total contributions**
134	instructors' responses
112	students' responses
7 min	avg. response time

TOP STUDENT CONTRIBUTORS & ANSWERERS!

Top Student Contributors

57 contributions; 94 days online	Christian James Welly
37 contributions; 69 days online	Nelson Tan
33 contributions; 84 days online	Chen Shenghao
32 contributions; 77 days online	Nathanael Seen Zhong Qi
25 contributions; 86 days online	Mario Lorenzo

THANK YOU!

Good luck for the finals!!!

Please provide feedback!

Please verify your scores on Luminus!