Disjoint Sets

Announcements

- PS3 deadline: Monday April 4th
  - Extra office hours – see calendar on Quercus
- Friday lectures this week – usual Zoom links
- Course evaluations

Disjoint Set ADT

MAKE-SET(x) Given element x that does not already belong to one of the sets, create a new set {x} that contains only x. Assign x as the representative of that new set.

FIND-SET(x) Given element x, return the representative of the set that contains x (or NIL if x does not belong to any set).

UNION(x, y) Given two distinct elements x and y, let S_x be the set that contains x and S_y be the set that contains y. Form a new set consisting of $S_x \cup S_y$ and remove S_x and S_y from the collection. Pick a representative for the new set. As a pre-condition, it is required that x and y each be an element of some set.

Kruskal's Algorithm

```
MST-KRUSKAL(G=(V,E),w:E →R)
1   T ← {}
2   sort edges so w(e_1) ≤ w(e_2) ≤ ... ≤ w(e_m)

3   for i ← 1 to m:
4       let (u_i, v_i) = (e_i)
5   # if u_i, v_i in different connected components of T:
        if FIND-SET(u_i) != FIND-SET(v_i):
            UNION(u_i, v_i)
    #   T ← T ∪ {e_i}
```

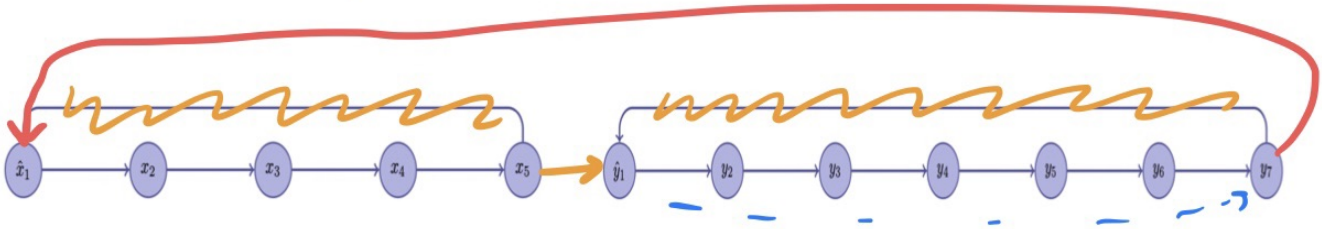
→ strongly connected graph: each vertex has a path to every other

Disjoint Set ADT: Implementations

- Disjoint Set ADT manages sets only. client code manages elements
- Each element x, manages a pointer to DJS data structure
- Complexity analyzed using worst-case sequence complexity – in keeping with context of Kruskal's Algorithm
- Each analysis based on m operations where n is the number of MAKE-SETS

Chalk Talk

Implementation 1: circular linked list



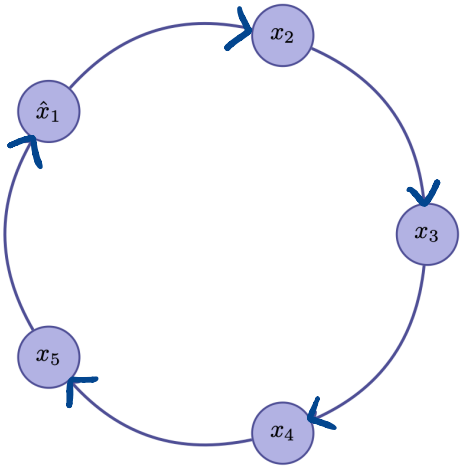
WCSC: m Operations with n MAKE-SETS?

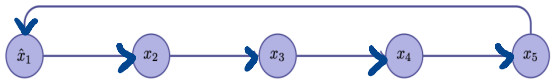
tail pointers

↑ for UNION

Chalk Talk

Implementation 1: circular linked list





Make set O(1) 

Find set O(n)

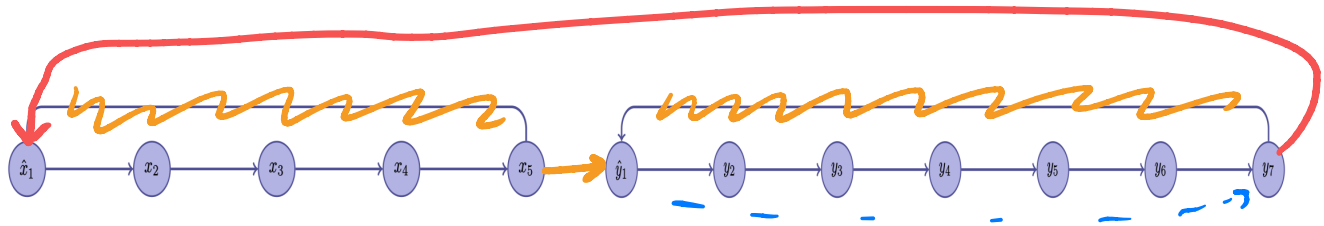
↳ has to go around to each element to see if its the rep

Union ??

Once we find the reps, we can do union in O(1)

Chalk Talk

Implementation 1: circular linked list



WCSC: m Operations with n MAKE-SETS?

tail pointers

One bad sequence $n = \frac{m}{4}$ MAKE-SETS,

$\frac{m}{4}$ UNIONS, $\frac{M}{2}$ FINDSETS on the 2nd element

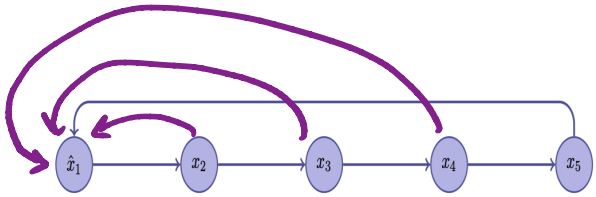
Each FIND-SETS takes $\Omega\left(\frac{m}{4}\right)$ so total time is $\Omega(m^2)$

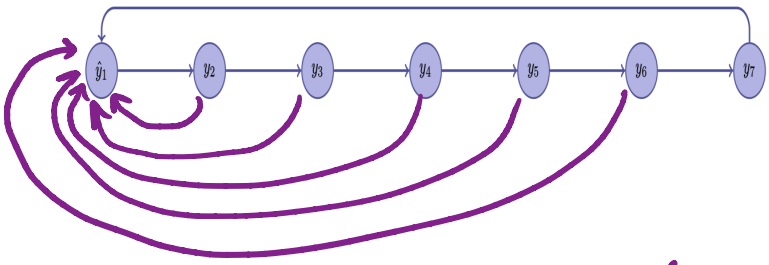
Since the number of elements at any point is $\leq m$, complexity of each operation is $< m$. Total time is $O(m^2)$

$\Rightarrow \Theta(m^2)$ for m operations

Chalk Talk

Implementation 2: Add back pointers to the rep





$n = \frac{m}{2} + 1$ MAKE-SETS

$\frac{m}{2} - 1$ UNIONS

always append larger lists to shorter ones

$\Theta(n^2)$

MAKE-SET $O(1)$

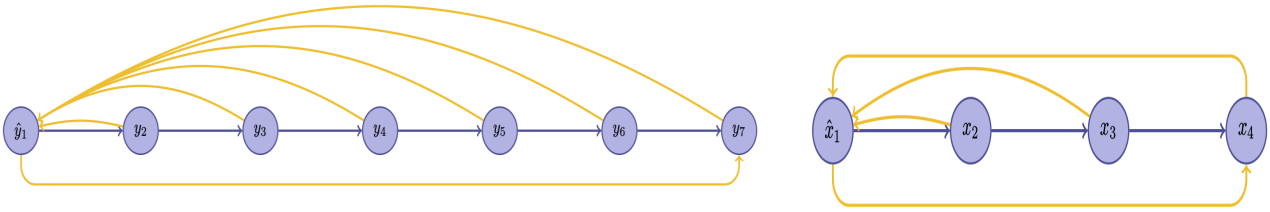
FIND-SET $O(1)$

UNION(x,y) $\Theta(n^2)$

↳ have to change all pointers

Chalk Talk

Implementation 3: UNION BY WEIGHT



Augment DS with # elements in each set

UNION(x,y) → pick the rep from the set x or x with more elements

Chalk Talk

Implementation 3 Analysis

WCSC: m Operations with n MAKE-SETS?

Since we have n make-sets, we never have more than n elements

For some arbitrary element x, we want upper bound on the number of times we could update x's back pointer.

x's pointer only changes when x is in the smallest set which means that each time x's pointer was updated, the size of the set containing x at least doubled. This can happen at most log(n) times. Since true for all x's, most number of pointer updates is nlogn

In total m operation = $O(m + n\log n)$

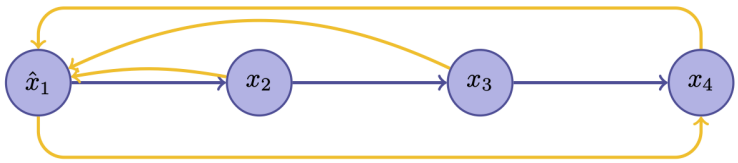
UNION → in addition to time for FIND-SET is O(1)

MAKE-SET → O(1)

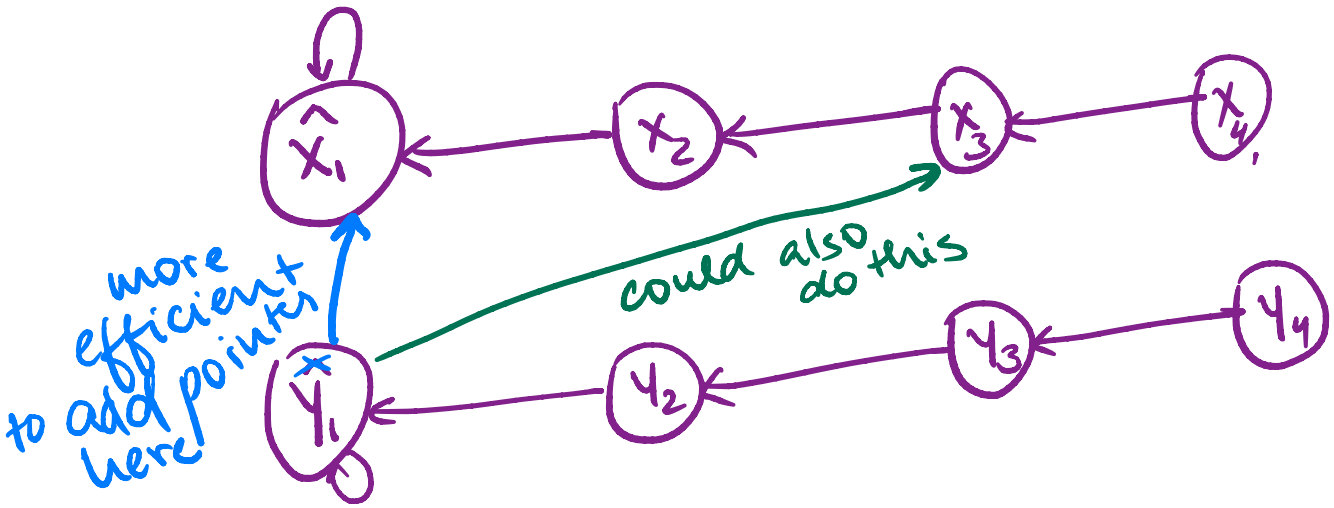
FIND-SET → O(n) worst case

Chalk Talk

New Implementation idea!



Instead of forward pointers and back pointers



UNION (x_3, y_2)

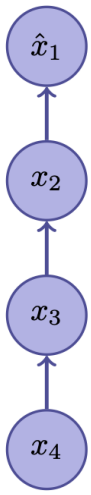
We can't put y_1 at the end to x_4 cuz we don't have pointers to the last element

Chalk Talk

Implementation 4: Trees

Lower Bound on WC:

bad sequence:



$\Omega(m^2)$

- $n = \frac{m}{4}$ make-sets
- $\frac{m}{4} - 1$ unions(x, y) when singleton x_1 is end up with long chain of $\frac{m}{4}$ nodes
- $\frac{m}{2} + 1$ FIND-SET(2) where 2 is a leaf

Upper-bound on WC: same argument as we made in implementation 1

m operations every operation cost $< m$

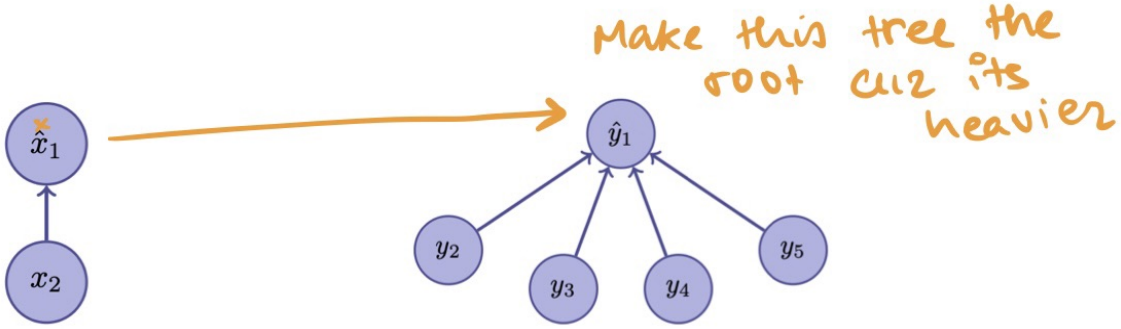
$\therefore$ total cost $< m^2$

$O(m^2)$

Chalk Talk

Implementation 5: TREE UNION BY WEIGHT

- select the "heavier" tree's rep as the new root
- Augment the tree to store the weight (# of elements in tree)



Chalk Talk

Implementation 5 analysis

WCSC: m Operations with n MAKE-SETS?

makeset O(1)

Union O(1) → once we have reps

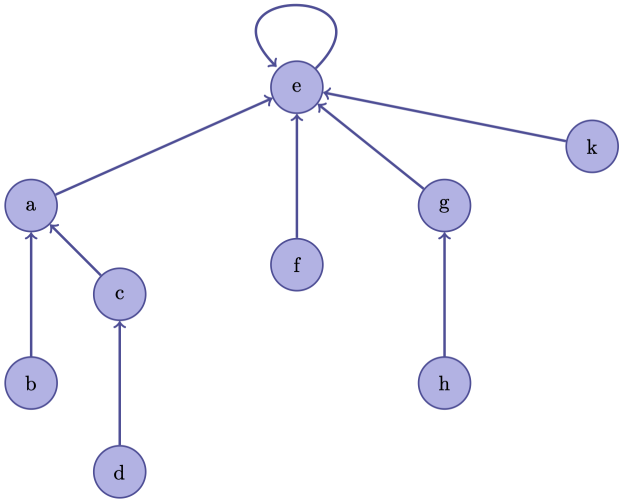
complexity of find-set:

during any sequence of m operation
n of which are MAKESET, the
max height of the tree is O(log n)

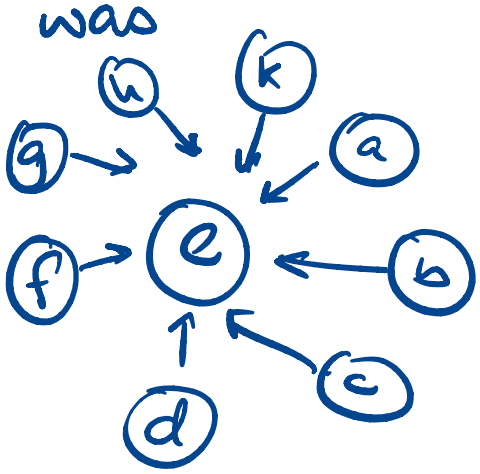
Proof → induction on height h of trees
Individual running time for
find-set O(logn)
total time for m operations O(mlogn)

Chalk Talk

Implementation Improvement idea!



if it was



FIND-SET would be much faster

Chalk Talk

Implementation 6: PATH COMPRESSION

- During FIND-SET, keep track of nodes visited on the path from x to the root (using a stack, queue or recursion)
- once root (i.e rep) is found, update the parent pointers of each node encountered (x.p) to point directly to the root
- this is at most double the time for FIND-SET but speeds up future operations

WCSC: m Operations with n MAKE-SETS

$\Theta(m \log m)$ → proof is messy

Chalk Talk

Implementation 7: Union By Rank

Motivation? Now that we have path compression, weight isn't nearly as important as height of tree. Instead of looking at the exact weight/height, maintain an upper bound on the height

Called rank

- Rank of leaf is 0
- Rank of an internal node is 1+ max rank of its children

Chalk Talk

Implementation 7 continued

MAKE_SET: set rank to 0

FIND-SET: ranks remains . Use path compression

Actual height might unchanged change due to PC, rank is unchanged

UNION(x,y):

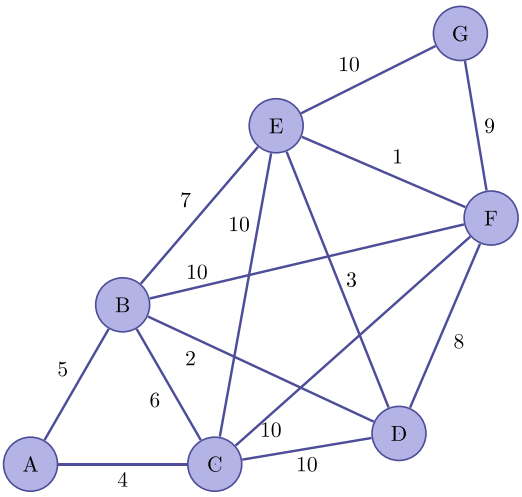
Node with higher rank is new root. Its rank is unchanged

If x, y have equal rank, choose either as new root and its rank is increased by 1

as a convention, chose x as the new rep.

Chalk Talk

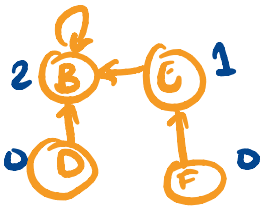
Implementation 7 continued

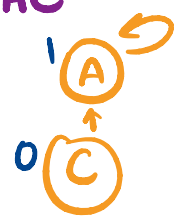


Initial MAKE_SET:

1> 

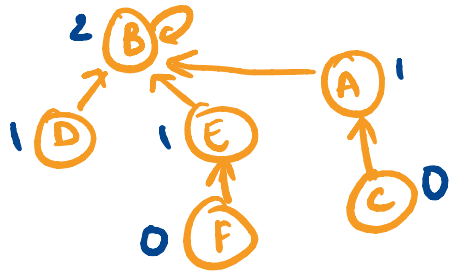
2> BD 

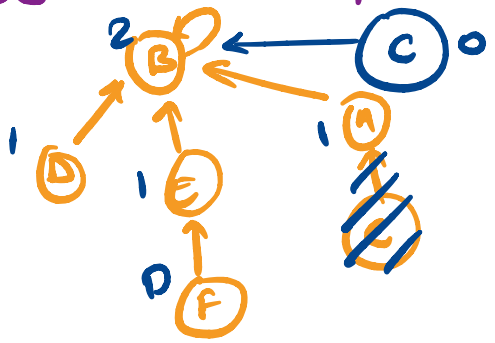
3> DE 

4> AC 

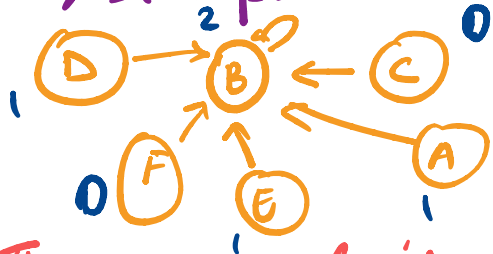
UNION OPERATION (find-set on each vertex before union)

EF, BD, DE, AC, AB, BC, BE, DF, FG, ...

5> AB 

6> BC : Path compression 

7> BE : nothing happens here

8> DF : path compression 

The ranks don't change in PC!

Chalk Talk

Implementation 7: Union By Rank

WCSC: m Operations with n MAKE-SETS

$O(m \log^* n)$

$\alpha(n)$

→ inverse ackerman function

→ iterated log: the no. of times the log function must be iteratively applied before result ≤ 1

$$\log^* n = \begin{cases} 0 & \text{if } n \leq 1 \\ 1 + \log^*(\log n) & \text{otherwise} \end{cases}$$

So the wcsc for implementation 7 is

$O(m\,\alpha(n))$

→ Any disjoint set would have an amortised cost of $\Omega(\alpha(n))$ for FIND_SET

Worksheet Q4

Mark in the Google Doc row for your group or in the chat when you are finished Q4 and then come back to the main room.

http://tinyurl.com/breakouts-263

If you are watching the video, pause long enough to answer Q4.

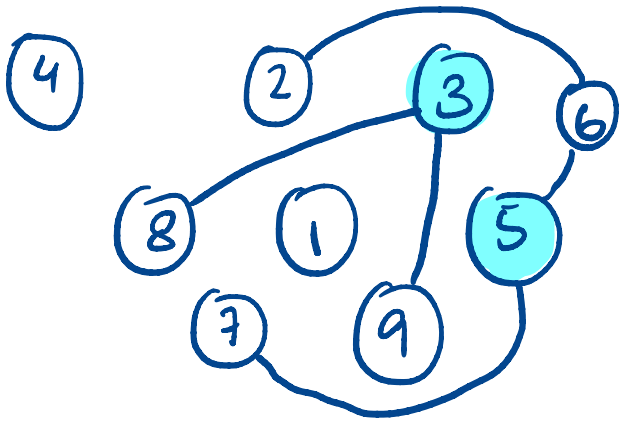
Worksheet Q5

Mark in the Google Doc row for your group or in the chat when you are finished Q5 and then come back to the main room.

http://tinyurl.com/breakouts-263

If you are watching the video, pause long enough to answer Q5a.

1. Draw your own pictures to represent the sets that would result from tracing the following sequence of operations. MAKE-SET(4), MAKE-SET(2), MAKE-SET(3), MAKE-SET(8), MAKE-SET(1), MAKE-SET(5), MAKE-SET(7), MAKE-SET(9), UNION(8,3), UNION(5,7), UNION(9,3), MAKE-SET(6), UNION(6,2), UNION(7,6)



   (a) How many sets are there after the 14 operations have executed? 4

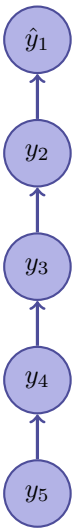
   (b) What is the value of the expression FIND-SET(8) == FIND-SET(9)? True

   (c) Does the order of the MAKE-SET operations matter? Nope, except before unions

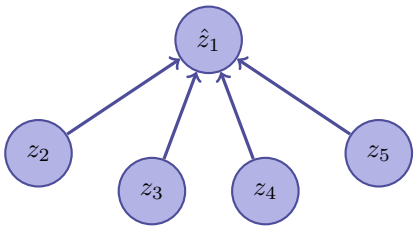
2. Using Implementation 4 from our lecture answer the following.

   (a) Trace the following sequence of operations drawing the data-structure. MAKE-SET(a),MAKE-SET(b),MAKE-SET(c),MAKE-SET(d),MAKE-SET(e), UNION(a,b), UNION(c,d), UNION(e,d), UNION(b,d)

   (b) Give a sequence of operations that would result in this data-structure.



(c) Give a sequence of operations that would result in this data-structure.



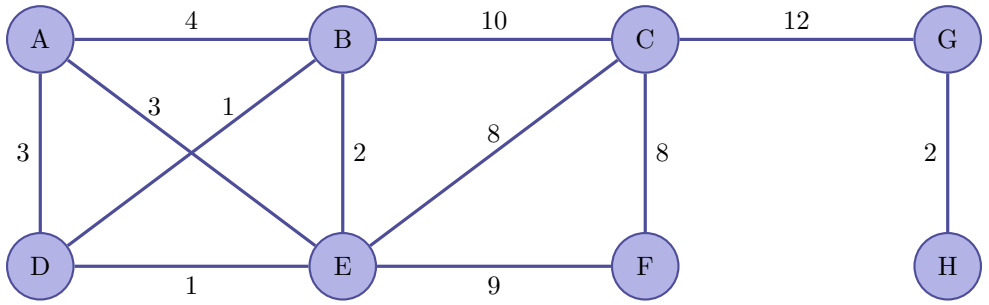
(d) Suppose that after the operations in part (a), we call FIND-SET(x) where x is chosen uniformly at random from a,b,c,d or e. What is the average number of elements that have to be examined?

3. Using Implementation 5 from our lecture answer the following.

(a) Trace the following sequence of operations drawing the data-structure. MAKE-SET(a),MAKE-SET(b),MAKE-SET(c),MAKE-SET(d),MAKE-SET(e), MAKE-SET(f), MAKE-SET(g), MAKE-SET(h), MAKE-SET(k), UNION(a,b), UNION(c,d), UNION(a,c), UNION(e,f), UNION(g,h), UNION(f,g), UNION(f,k), UNION(f,b)

(b) Suppose we then call FIND-SET(d). List in order the elements that are examined as we traverse from d to the root.

4. For this question, we will run Kruskal's algorithm on the following graph. Below the graph is a sorted list of the edges to use.



`(D,B), (E,D), (E,B), (H,G), (A,E), (A,D), (A,B), (C,F), (C,E), (E,F), (B,C), (G,C)`

(a) Trace the execution of Kruskal's algorithm using trees with path compression to implement the disjoint set ADT. When you call UNION(x,y), make the root of the tree containing x become the root of the new tree.

(b) Start again and trace the execution of Kruskal's algorithm using trees with path compression and union by rank to implement the disjoint set ADT. When you call UNION(x,y), **and the ranks are equal**, make the root of the tree containing x become the root of the new tree.

CSC263 Worksheet: Week 11: Disjoint Set Implementations

1. Draw your own pictures to represent the sets that would result from tracing the following sequence of operations. MAKE-SET(4), MAKE-SET(2), MAKE-SET(3), MAKE-SET(8), MAKE-SET(1), MAKE-SET(5), MAKE-SET(7), MAKE-SET(9), UNION(8,3), UNION(5,7), UNION(9,3), MAKE-SET(6), UNION(6,2), UNION(7,6)

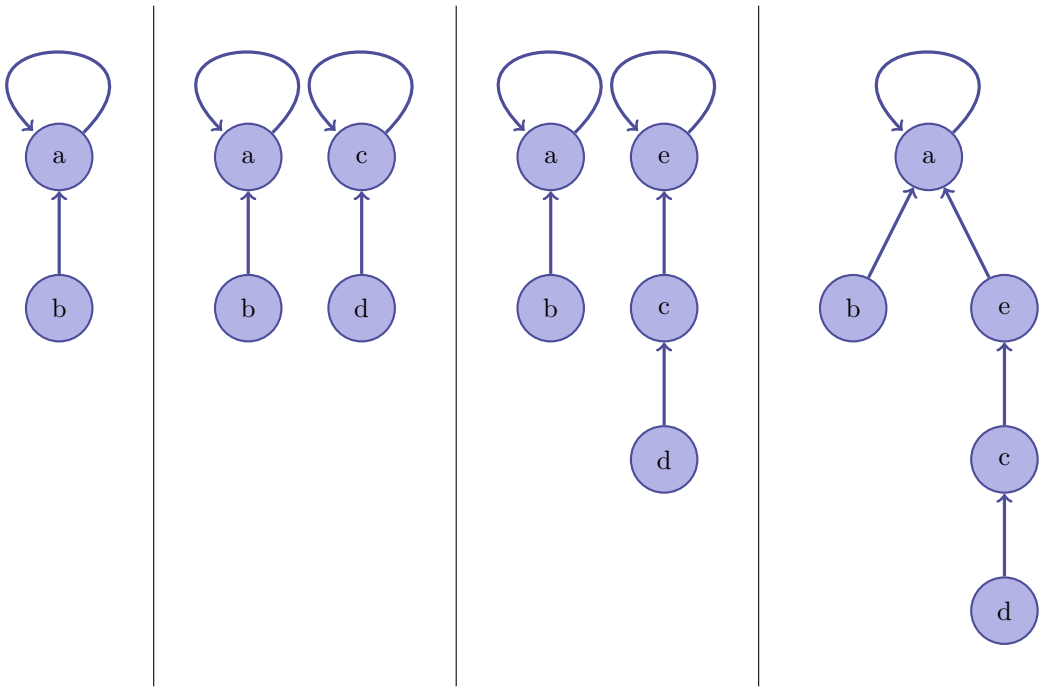
   (a) How many sets are there after the 14 operations have executed? SOLUTION: 4

   (b) What is the value of the expression FIND-SET(8) == FIND-SET(9)? SOLUTION: True

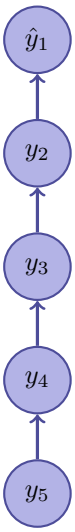
   (c) Does the order of the MAKE-SET operations matter? SOLUTION: Not at all. After this, we will usually put them all first.

2. Using Implementation 4 from our lecture answer the following.

   (a) Trace the following sequence of operations drawing the data-structure. MAKE-SET(a),MAKE-SET(b),MAKE-SET(c),MAKE-SET(d),MAKE-SET(e), UNION(a,b), UNION(c,d), UNION(e,d), UNION(b,d)
   SOLUTION:

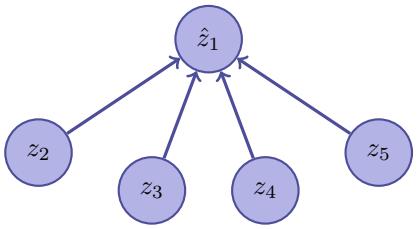


   (b) Give a sequence of operations that would result in this data-structure.



SOLUTION: One possible sequence is: MAKE-SET(1), MAKE-SET(2),MAKE-SET(3),MAKE-SET(4),MAKE-SET(5), UNION(4,5), UNION(3,5), UNION(2,3), UNION(1,5).

(c) Give a sequence of operations that would result in this data-structure.

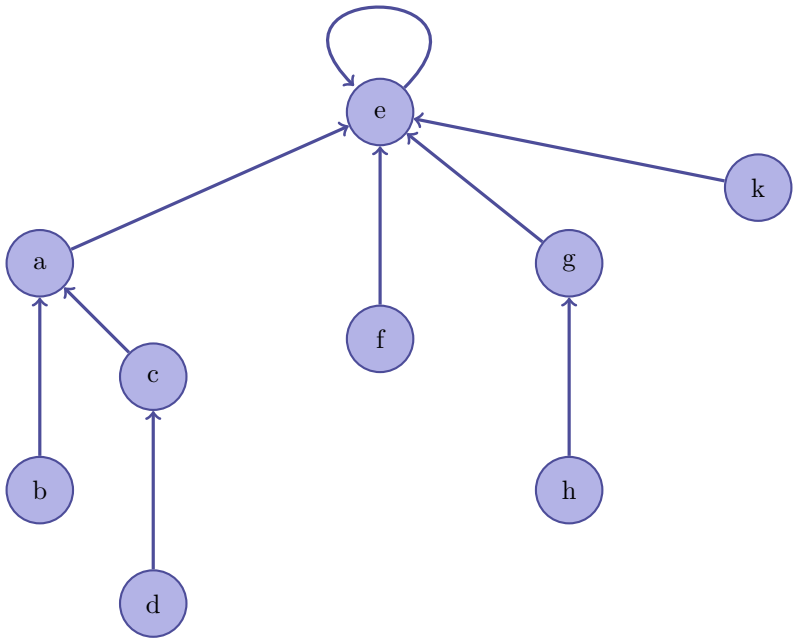


SOLUTION: One possible sequence is: MAKE-SET(1), MAKE-SET(2),MAKE-SET(3),MAKE-SET(4),MAKE-SET(5), UNION(1,5), UNION(1,3), UNION(1,2), UNION(1,4).

(d) Suppose that after the operations in part (a), we call FIND-SET(x) where x is chosen uniformly at random from a,b,c,d or e. What is the average number of elements that have to be examined?
SOLUTION: $\frac{1+2(2)+3+4}{5} = \frac{12}{5} = 2.4$

3. Using Implementation 5 from our lecture answer the following.

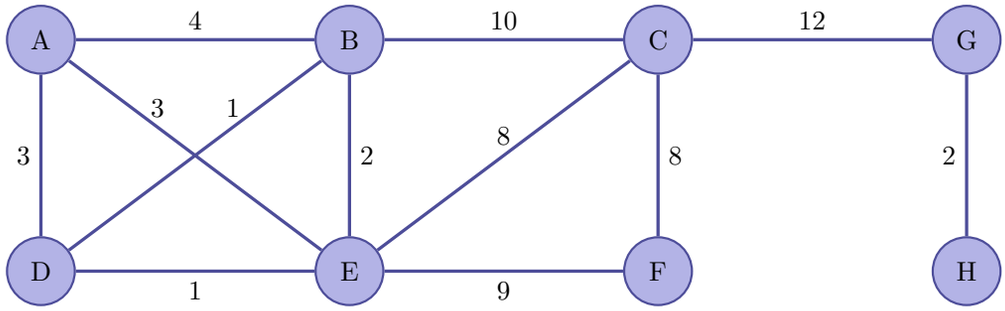
(a) Trace the following sequence of operations drawing the data-structure. MAKE-SET(a),MAKE-SET(b),MAKE-SET(c),MAKE-SET(d),MAKE-SET(e), MAKE-SET(f), MAKE-SET(g), MAKE-SET(h), MAKE-SET(k), UNION(a,b), UNION(c,d), UNION(a,c), UNION(e,f), UNION(g,h), UNION(f,g), UNION(f,k), UNION(f,b)
SOLUTION:



(b) Suppose we then call FIND-SET(d). List in order the elements that are examined as we traverse from d to the root.
SOLUTION: d, c, a, e

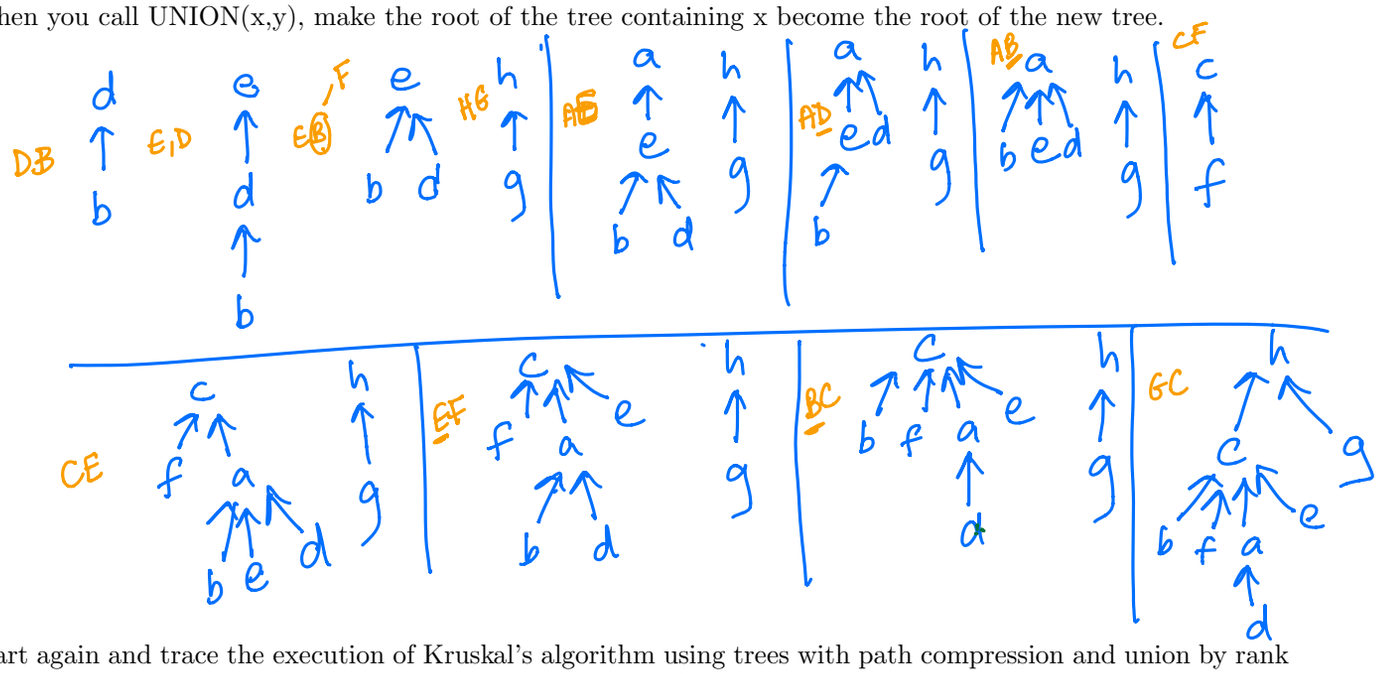
CSC263 Worksheet: Week 12 Disjoint Set Implementations Continued

For this worksheet, we will run Kruskal's algorithm on the following graph. Below the graph is a sorted list of the edges to use.



`(D,B), (E,D), (E,B), (H,G), (A,E), (A,D), (A,B), (C,F), (C,E), (E,F), (B,C), (G,C)`

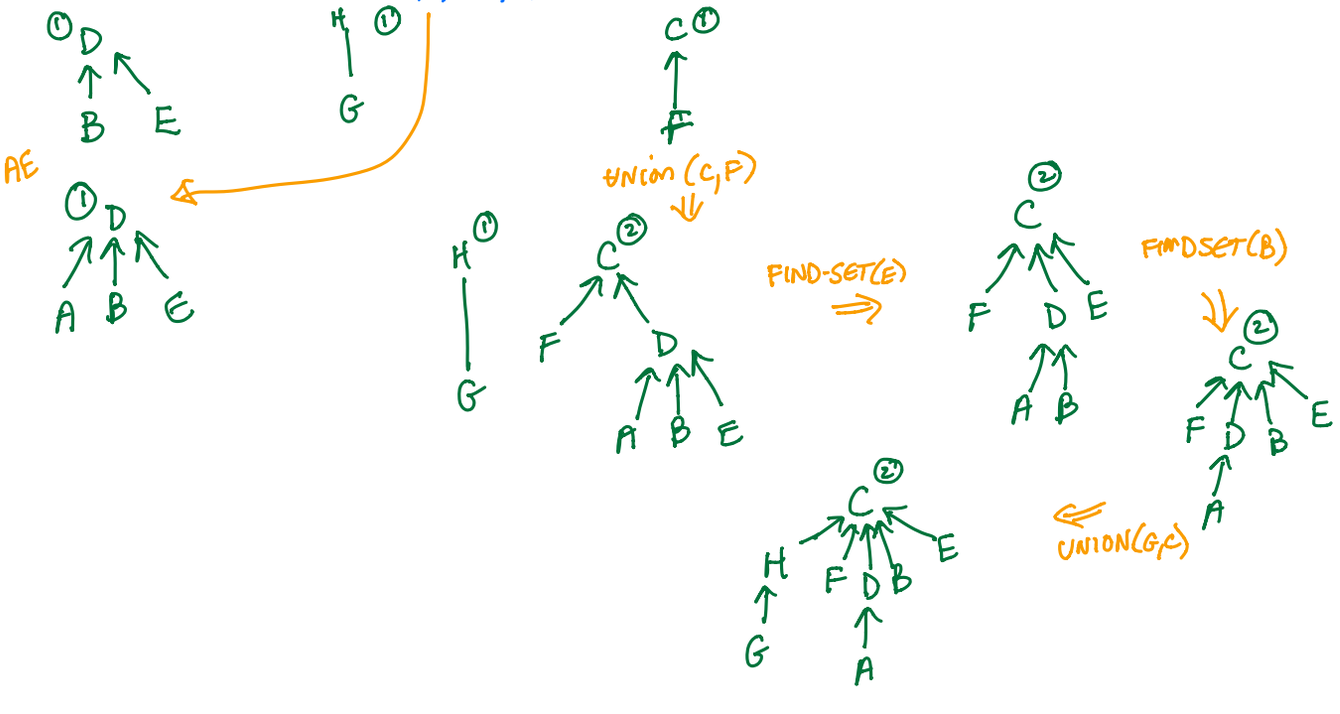
1. Trace the execution of Kruskal's algorithm using trees with path compression to implement the disjoint set ADT. When you call UNION(x,y), make the root of the tree containing x become the root of the new tree.



2. Start again and trace the execution of Kruskal's algorithm using trees with path compression and union by rank to implement the disjoint set ADT. When you call UNION(x,y), **and the ranks are equal**, make the root of the tree containing x become the root of the new tree.

`(D,B), (E,D), (E,B), (H,G), (A,E), (A,D), (A,B), (C,F), (C,E), (E,F), (B,C), (G,C)`

no effect · no effect · no effect · no change



Lower bounds for Sorting and Course Wrap Up

Chalk Talk

How fast can we sort?

mergesort

- Existence of algorithms that run in worst-case time O(nlogn) confirm that sorting can be done in time O(nlogn), but does not rule out existance of other algorithm that do better
- We know how to analyze WC complexity of algorithm. WC complexity of _____ involves extra work.
- For problem P, C(P) – best worst-case running time of any algorithm that solves P
  - Upper bound on C(P): give an algorithm an analyze its time
  - Lower bound on C(P): ??? every algo requires a certain amount of time
- In practice, prove lower bounds on classes of algorithms.

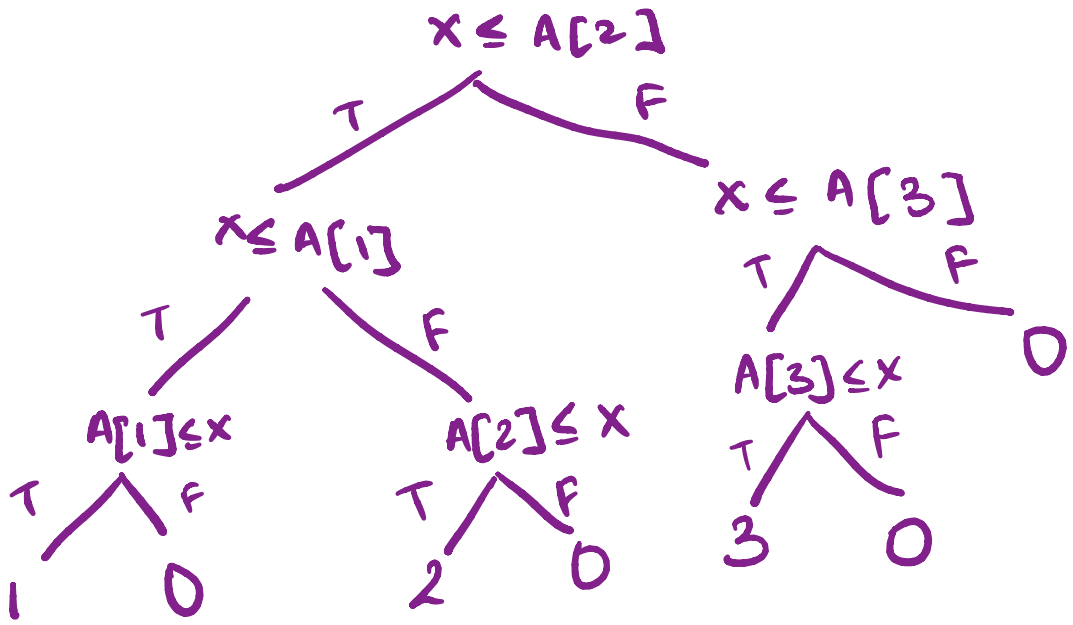
WC bubble-sort: n^2
WC merge-sort: nlogn
WC selection-sort: n^2

} best is merge-sort

Chalk Talk

Comparison Trees

- Represent algorithms that work by pair-wise comparison of element
- Example (binary search in A[1..3]), returning index of x (or 0)



Active Learning

Exercise Break

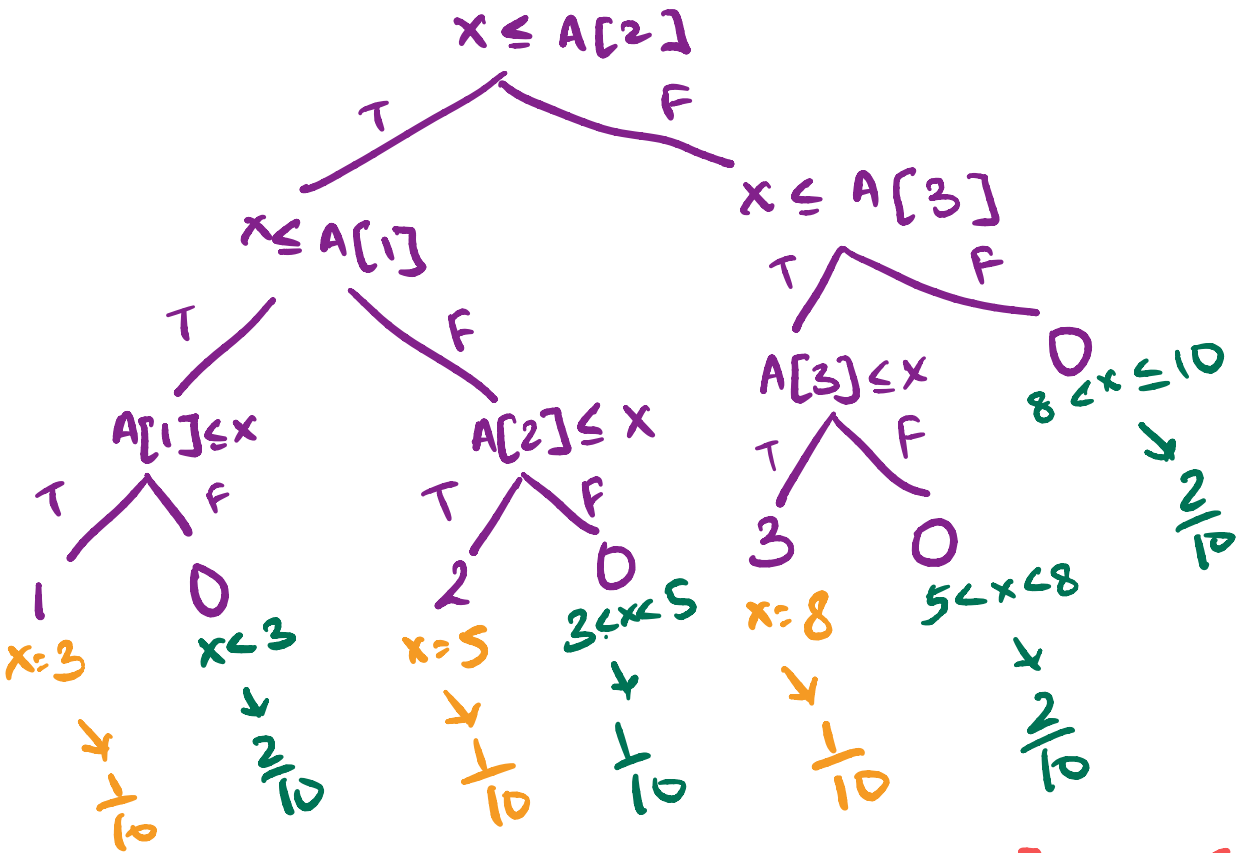
Consider the comparison tree that we just created and suppose A = [3, 5, 8]

- What is the worst-case number of comparisons?
- What is the expected number of comparisons if we select x uniformly from {1, …, 10} ?
- What if we select x uniformly from {1, …, 100}?

Active Learning

Exercise Break

A = [3, 5, 8] select x from {1, …, 10} select x from {1, …, 100}



$$= 3\left[\frac{1}{10} + \frac{2}{10} + \frac{1}{10} + \frac{1}{10} + \frac{1}{10} + \frac{2}{10}\right] + 2\left[\frac{2}{10}\right]$$

$$= 3\left(\frac{8}{10}\right) + \frac{2}{10}(2) = 2.8$$

Chalk Talk

Information Theory lower bounds

- Every binary tree with height h has $\leq 2^h$ leaves
  - every binary tree with L leaves has height $\geq \lceil \log_2 L \rceil$
- Every comparison tree that solves a problem P has at least one leaf for each possible output
  - every comparison tree for P has height $\geq \lceil \log_2 m \rceil$ where m is the number of possible outputs

Chalk Talk

Applying this to sorting

- Input: A[1..n]
- Output: permutation of [1, … , n] indicating position of each element

rank

Example: Input is [6, 100, -1, 4] Output is [3, 4, 1, 2]

Example: Input is [5, 4, 10, 9] Output? [2, 1, 4, 3]

Why do it this way?

How many possible outputs? n!

Active Learning

Exercise Break

Draw the comparison tree for sorting a 3-element array.

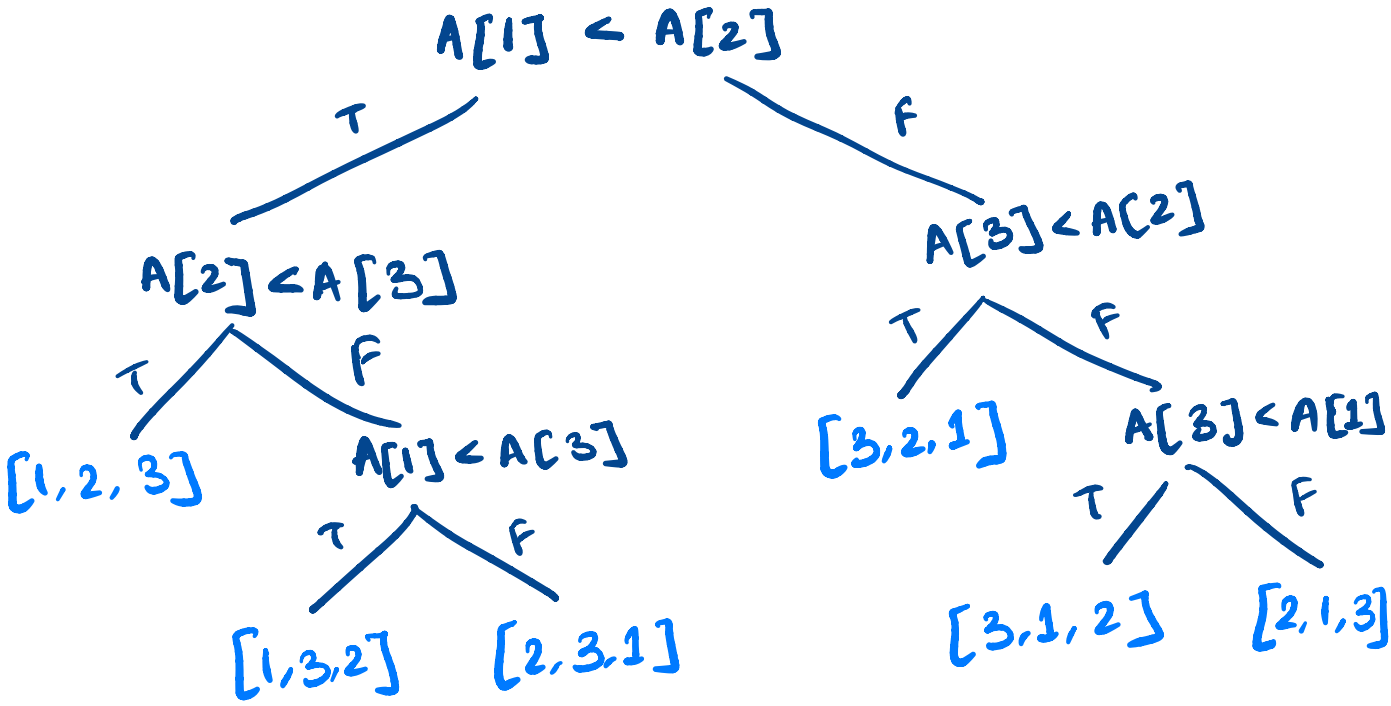
- Input: A[1], A[2], A[3]
- Output: permutation of [1, … , n] indicating position of each element

Now suppose A = [5, 3, 8] → [2, 1, 3]

- What is the correct output?
- Trace through the tree to reach that output.

Active Learning

Comparison Tree for Sorting (n=3)



Chalk Talk

Information theory lower bound

- Number of outputs = $n!$
- Every comparison tree has height $\geq \log_2(n!)$
- Every algorithm that uses only comparisons requires at least $\log_2(n!)$ comparisons.
- Fact $\log_2(n!) = \Theta(n \log n)$

MEMORISE THIS →

$\log(n!) = \log(n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1)$
$= \log n + \log(n-1) + \cdots \log 2 + \log 1$
$\leq \log n + \log n + \cdots \log n$
$\leq O(n \log n)$

look at first ½ all $> \log \frac{n}{2}$
$\log(n!) \geq \frac{n}{2} \log(\frac{n}{2})$
$\Omega(\frac{n}{2} \log \frac{n}{2})$
$\Rightarrow \Theta(n \log n)$

- Therefore, every algorithm that uses only comparisons between pairs of elements to sort requires $\Theta(n \log n)$

Chalk Talk

What about other sorts?

- Radix sort or counting sort?
- Can be done in less than O(n log n) comparisons because
  - Not based on pairwise comparisons
  - restrict the problem down
- Be careful when lower bounds apply only to algorithms of a particular type.
- Be careful when we are using a restricted problem
  - Sorting n distinct numbers from 1 to n → 1 to n
  - Sorting n-1 distinct numbers from 1 to n

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

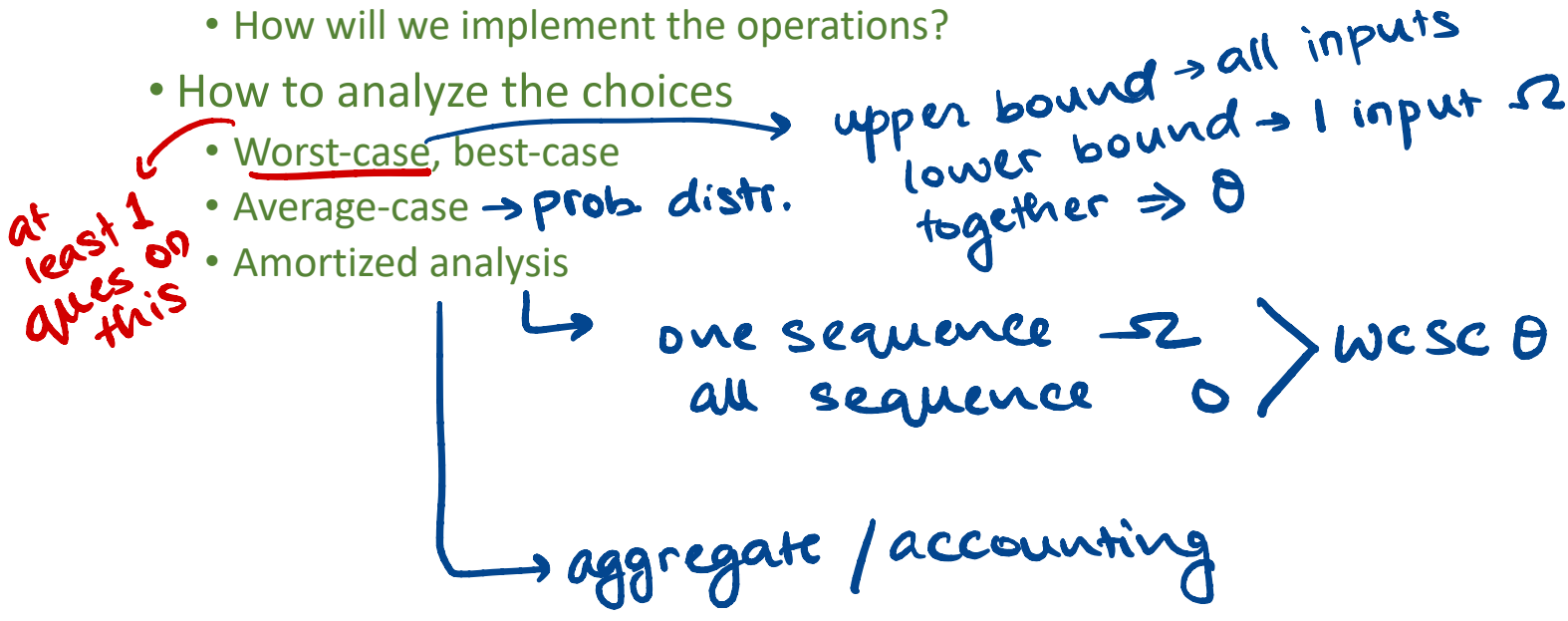
↖ memorise

Chalk Talk

If we have time: B trees and B+ trees

What have we learned?

- Given an ADT, we have choices about how to design the data structure
- How to plan our our implementation
  - How will we store the data?
  - How will we implement the operations?
- How to analyze the choices
  - Worst-case, best-case
  - Average-case → prob distr.
  - Amortized analysis



What have we learned?

- Heaps
  - Used for priority queues
  - Implementation using array
  - Operations including algorithm for building if we have list
- Balanced trees (AVL trees)
  - Balance factors, operations including rotations
- Hash tables
  - Implementations
  - Usefulness also limitations
- Augmenting and combining data structures

What have we learned?

- One new sort: randomized quick sort
- Graphs adjacency list + adjacency matrix
  - Representations:
  - Algorithms: BFS, DFS, MST – Kruskal & Prim

- Disjoint Set ADT implementations
- Theoretical limits on comparison-based algorithms

At least 2 questions to trace the algorithm on BFS/DFS/AVL/MST/...

Advice

The Final Exam

3 hours

9 ”questions” for 72 marks ~2.5 minutes per mark

- In runtime analysis question:

“Simplify your answer until you have (one or more) summations of the following form:

$\sum_{i=1}^{b} a^i$ $\sum_{i=1}^{b} i a^i$ $\sum_{i=1}^{b} i = \frac{b(b+1)}{2}$

Memorise this!

Do not expand the summations or do any further simplifications beyond getting to these forms.”

- In worst-case analysis don’t forget the ~~lower bound~~ upper bound!

know what its Big-Oh is!

Advice

The Final Exam

- Be sure you can do the trace-this-algorithm questions, but don't expect many of these.
  - Maybe do them first to boost your confidence
- Expect a "design a data structure to implement this ADT"
  - Part marks if your design meets some but not all constraints
  - Be honest if your design doesn't meet the complexity constraints
- Explain what you are doing, but don't ramble

Advice

Studying

- Make sure you know where you made mistakes on the midterm or on problem sets
- Practice on old exams from the library repository
- Make yourself summary notes of the course as if you were preparing an aid sheet
- Come to pre-exam office hours (watch Quercus for times)

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science
DECEMBER 2019 EXAMINATIONS
CSC 263 H1F
Duration: 3 hours
Aids Allowed: None

PLEASE HAND IN

Student Number: |_|_|_|_|_|_|_|_|_|_|_|

Last (Family) Name(s): _____

First (Given) Name(s): _____

---

*Do **not** turn this page until you have received the signal to start.*
In the meantime, please read *carefully* every reminder on this page.

- Fill out your name and student number above—do it now, don't wait!
- Turn off and place all cell phones, smart watches, electronic devices, and unauthorized study materials in your bag under your desk. If it is left in your pocket, it may be an academic offence.
- When you are done your exam, raise your hand for someone to come and collect your exam. Do not collect your bag and jacket before your exam is handed in.
- If you are feeling ill and unable to finish your exam, please bring it to the attention of an Exam Facilitator so it can be recorded before leaving the exam hall.
- In the event of a fire alarm, do not check your cell phone when escorted outside.
- This final examination consists of 6 questions on 18 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy of the examination is complete.*
- Answer each question directly on the examination paper, in the space provided, and use a "blank" page for rough work. If you need more space for one of your solutions, use one of the "blank" pages and *indicate clearly the part of your work that should be marked.*
- *Remember that, in order to pass the course, you must achieve a grade of at least **40%** on this final examination.*
- As a student, you help create a fair and inclusive writing environment. If you possess an unauthorized aid during an exam, you may be charged with an academic offence.
- State any assumptions you make.

Marking Guide

Nº 1:	_____/ 50
Nº 2:	_____/ 10
Nº 3:	_____/ 20
Nº 4:	_____/ 10
Nº 5:	_____/ 20
Nº 6:	_____/ 10
TOTAL:	_____/120

STUDENTS MUST HAND IN ALL EXAMINATION MATERIALS AT THE END

Good Luck!

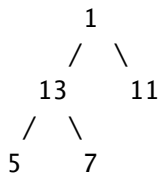
OVER...

Question 1. Multiple Choice using Bubble Sheet. [50 marks]

Instructions. Indicate your answers on the **bubble sheet** at the end of this booklet. Fill *exactly one* bubble per question. Use a pen. If you change your mind, *cross out* (X) the filled bubble, then fill in another. Your bubble sheet will constitute your complete response to these questions.

Failure to use the bubble sheet will incur a **10 point penalty.**

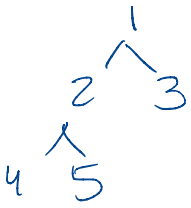
1. Which property of a max-heap is being violated by the structure below?



   (a) Shape
   (b) Order
   (c) Completeness
   (d) Height balance
   (e) None of the above

2. What is the minimum number of elements in a non-empty heap of height h? (Convention: a heap with only one element has height 0.)
   (a) 2^h
   (b) $2^h + 1$
   (c) $2^h - 1$
   (d) $h + 2$
   (e) None of the above

3. What is the index of the right child of a node i in a min-heap? (Convention: the root node of a tree has index 1.)
   (a) $2i$
   (b) $2i - 1$
   (c) $2i + 1$
   (d) $\frac{i}{2}$
   (e) None of the above

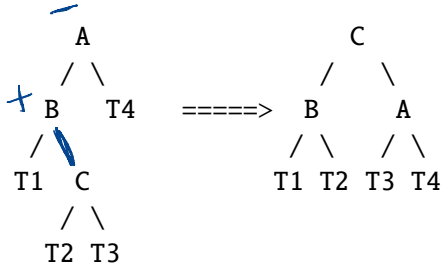


4. Which operations are supported by a min-priority queue?
   (a) Push, Pop
   (b) Enqueue, Dequeue
   (c) Insert, Search
   (d) Insert, Extract-Min
   (e) None of the above

5. Consider a Dictionary D containing n nodes. Let k be a key and x be a (pointer to a) dictionary node. If the Dictionary is implemented as _____, then the operation _____ has worst case complexity _____.
   (a) an unsorted doubly linked list, DELETE(D,x), $O(n)$
   (b) a sorted array, DELETE(D,x), $O(1)$
   (c) a binary search tree, SEARCH(D,k), $O(\log n)$
   (d) a sorted array, SEARCH(D,k), $O(\log n)$
   (e) None of the above.

6. Which of the following statements about AVL trees is wrong:
   (a) An AVL tree augments a binary search tree with a balance factor (BF) at every node x.
   (b) The subtree rooted at (every) node x is either balanced, right heavy, or left heavy.
   (c) A rotation at x during INSERT may necessitate updating BFs for all ancestors of x.
   (d) A rotation at x during DELETE may necessitate updating BFs for all ancestors of x.
   (e) None of the above.

7. Consider the following binary tree, where A, B, and C are nodes and T1, T2, T3, and T4 are subtrees. What rotations have been applied on the tree on the left in order to get the tree on the right?



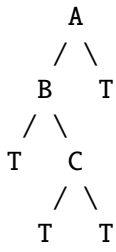
left-right rotation
B
A

   (a) Perform a right rotation on A, then a left rotation on C.
   (b) Perform a right rotation on A, then a right rotation on C.
   (c) Perform a left rotation on B, then a right rotation on C.
   (d) Perform a left rotation on B, then a right rotation on A.
   (e) None of the above.

8. Let f be an attribute that augments an AVL tree T of n nodes. Suppose that the value $(x.f)$ of f at each node x can be computed in $O(1)$ time from any information stored at x, $x.left$, and $x.right$. It is always possible to modify the operations INSERT and DELETE so that each operation correctly maintains the values of f throughout the tree, and the worst case complexity of the revised operations INSERT and DELETE is
   (a) $\Theta(1)$
   (b) $\Theta(\log n)$
   (c) $\Theta(n)$
   (d) $\Theta(n \log n)$
   (e) None of the above

?

9. In the diagram, T is a *full* tree, and A, B, and C are nodes. The subtrees rooted at A, B, and C are not balanced, _____ balanced, and _____ balanced, respectively.



(a) height, height, height
(b) height, not, ideally
(c) not, ideally, height
(d) not, height, ideally
(e) None of the above.

10. Recall that an *Interval tree* augments an AVL tree to store a collection of intervals. Each node x is augmented to store the interval (x.int). Each node is keyed (x.key) on the low endpoint of the interval. Each node x is also augmented to store the maximum value (x.max) of any interval endpoint stored in the subtree rooted at x. Recall also that INTERVAL-SEARCH(T, i) finds a tree node whose interval overlaps the given interval i.

```
Buggy-Interval-Search(T,i)
1: x = T.root
2: while x not= T.nil and i overlaps x.int
3:     if x.left not= T.nil and x.left.max >= i.low
4:         x = x.left
5:     else x = x.right
6: return x
```

This algorithm contains a bug in

(a) line 2
(b) line 3
(c) lines 4 and 5
(d) line 6
(e) None of the above.

11. *Path compression* is a modification to the _____ operation in disjoint-set forests.

(a) MAKESET
(b) FINDSET
(c) LINK
(d) Both B and C.
(e) None of the above.

12. Consider a linked-list implementation of disjoint sets. In our implementation, every node carries a back-pointer to the representative. Presently, the data structure already contains n elements. If the next operation is UNION(x,y) / MAKESET(x) / FINDSET(x) what is the *worst case complexity* of a single operation?

(a) O(1), O(1), O(1)
(b) O(1), O(1), O(n)
(c) O(n), O(1), O(1)
(d) O(n), O(1), O(n)
(e) None of the above

remove back pointers for y

13. Consider a linked-list implementation of disjoint sets. In our implementation, every node carries a back-pointer to the representative. We use the union-by-weight heuristic. Consider a sequence of n operations comprised of MAKESET(x), FINDSET(x), and/or UNION(x,y). What is the *worst case sequence complexity*?

(a) $O(1)$
(b) $O(n)$
(c) $O(n \log n)$
(d) $O(n \log^* n)$
(e) None of the above

14. A hash table of size 10 uses open addressing with the hash function $h(k) = k \mod 10$, and linear probing. After inserting 6 values into an empty hash table, the table is as shown below.

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

Which one of the following choices gives a possible order in which the key values could have been inserted in the table?

(a) 46, 42, 34, 52, 23, 33 → 52 before 23
(b) 34, 42, 23, 52, 33, 46 → 33 before 46
(c) 46, 34, 42, 23, 52, 33 ✓
(d) 42, 46, 33, 23, 34, 52 ✗ → 33 can't be before 23
(e) None of the above

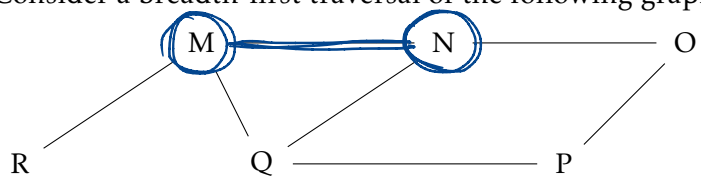
15. Universal hashing selects a hash function _____ in a way that is _____ of the keys that are actually going to be stored. The missing words are:
    (a) randomly, independent
    (b) randomly, dependent
    (c) deterministically, independent
    (d) deterministically, dependent
    (e) None of the above

16. Consider a hash table. Which of the following is not a technique for *resolving* a collision once it has occured?
    (a) Chaining
    (b) Uniform hashing
    (c) Linear probing
    (d) Double hashing
    (e) None of the above

17. What is the worst case time complexity of breadth first search on a graph $G = (V, E)$, represented by an adjacency list, in terms of $n = |V|$ and $m = |E|$?
    (a) $O(n)$
    (b) $O(mn)$
    (c) $O(m+n)$
    (d) $O(m^n)$
    (e) None of the above

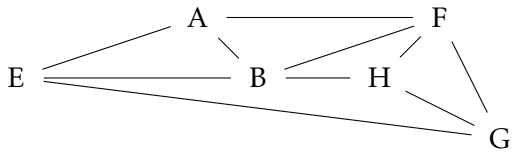
18. What is the worst case time complexity of breadth first search for the complete graph K_n, represented by an adjacency list? Recall that K_n has n vertices, and every pair of vertices is connected by an edge.
    (a) $O(\log n)$
    (b) $O(n)$
    (c) $O(n \log n)$
    (d) $O(n^2)$
    (e) None of the above

19. Consider a breadth-first traversal of the following graph:



What is a valid order for the discovery (transition from white to gray) of vertices?
    (a) M N O P Q R
    (b) N Q M P O R
    (c) Q M N P R O
    (d) Q M N P O R
    (e) None of the above

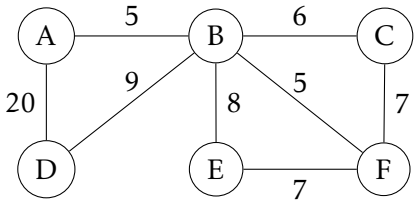
20. Consider a depth-first traversal of the following graph:



Among the following:
I) A B E G H F
II) A B F E H G
III) A B F H G E
IV) A F G H B E
Which are valid orderings for the vertices, if they are ordered in terms of increasing discovery time?

(a) I, II, and IV only
(b) I and IV only
(c) II, III and IV only
(d) I, III, and IV only
(e) None of the above

21. Apply Prim's algorithm, starting at node A. In what order are the edges visited?



(a) AB - BF - CF - EF - BD
(b) AB - BF - EF - BC - BD
(c) AB - BF - BC - EF - BD
(d) AB - BF - EF - CF - BD
(e) None of the above.

22. Imagine $G = (V, E)$ is a simple, connected, undirected, and weighted graph, where V is the set of vertices and E is the set of edges. We are trying to find a minimum spanning tree (MST) for this graph. Select the true statement:

(a) If all the edges in E have distinct weights, the MST is unique.
(b) Prim's and Kruskal's algorithms always produce the same MST.
(c) A MST has exactly $|V|$ edges. → $|V|-1$ edges
(d) Prim's algorithm uses disjoint sets; Kruskal's algorithm uses a priority queue.
(e) None of the above.

opposite
Kruskals → Disjoint set
Prims → Priority Queue

23. Kruskal's algorithm employs a _____ data structure.
    (a) disjoint set
    (b) dictionary
    (c) priority queue
    (d) minimal spanning queue
    (e) None of the above.

24. UNKNOWN is an algorithm that operates on disjoint set forests. Select the most appropriate description for UNKNOWN:

```
UNKNOWN( x )
1 if x not= x.p
2    x.p = UNKNOWN(x.p)
3 return x.p
```

    (a) UNION without weighting heuristic
    (b) UNION with weighting heuristic
    (c) FIND-SET without path compression
    (d) FIND-SET with path compression
    (e) None of the above.

25. Consider these statements:
    i) $f(n) \in \Theta(n^2) \rightarrow f(n) \in O(n^2)$
    ii) $f(n) \in O(n^2) \rightarrow f(n) \in \Theta(n^2)$
    iii) $f(n) \in \Omega(n^2) \rightarrow f(n) \in \Theta(n^2)$
    iv) $f(n) \in \Theta(n^2) \rightarrow f(n) \in \Omega(n^2)$

    Which of the above statements are true?
    (a) (i) and (ii)
    (b) (ii) and (iii)
    (c) (iii) and (iv)
    (d) (iv) and (i)
    (e) None of the above.

Reminder: Failure to use the bubble sheet will incur a *10 point penalty.*

Question 2. Randomized Quicksort [10 MARKS]

20% automatic grade for writing *"I don't know how to answer this"* and **crossing out the entire page**.

Part (a) [4 MARKS]

Let $A = [4, 5, 2, 6, 7, 12, 9, 13]$ be the **result** of running the first iteration of the Randomized-Quicksort algorithm. What are the possible pivots that the algorithm has chosen in this iteration?

6, 7, 13

Part (b) [4 MARKS]

Again, consider running the first iteration of the Randomized-Quicksort algorithm on an input array of distinct elements of length n. Choosing which elements as the pivot can lead us to the worst-case scenario?

2 or 13

Part (c) [2 MARKS]

What is the probability of choosing the elements that you have provided in the last section as pivot?

$\frac{2}{n} = \frac{2}{8} = \frac{1}{4} = 0.25$

Question 3. Amortized Analysis [20 marks]

20% automatic grade for writing *"I don't know how to answer this"* and **crossing out the entire page**.

Part (a) Binary Counter [10 marks]

Consider initializing a k-bit binary counter to the value zero, and then applying a sequence of n Increment operations.

Suppose that in our *cost model*, flipping a bit costs *one unit*. Then the *worst case cost* of a single Increment is $\Theta(k)$; yet, we saw in class that the *amortized cost* of each Increment is $O(1)$.

Now consider a different cost model, in which flipping the i-th bit costs i units. (By i-th bit, we mean the bit representing 2^i.) Show that the amortized time of Increment is still $O(1)$. *Hint:* You may use this fact: for any $0 \leq x < 1$, we have $\sum_{i=0}^{\infty} ix^i = \frac{1}{(1-x)^2} - \frac{1}{1-x}$.

Part (b) Two-Stack Queue, Part A [3 MARKS]

20% automatic grade for writing *"I don't know how to answer this"* and **crossing out the entire page**.

A *queue* is an abstract data structure, somewhat similar to *stacks*. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (ENQUEUE) and the other is used to remove data (DEQUEUE). Queue follows *first in*, *first out* data flow: elements are dequeued in the order that they are enqueued.

A queue Q can be implemented using two stacks, the "in" stack S_i and the "out" stack S_o. We ENQUEUE by pushing onto the "in" stack, and we DEQUEUE by popping out of the "out" stack. It can happen that S_o is empty, and S_i is non-empty. In this case, how do we dequeue? In this case, we first repeatedly pop from S_i and push onto S_o, until S_i is empty. In this way we transfer all the stacked up inputs to the output stack, setting them up in the appropriate order for dequeueing. (You may wish to draw some pictures to elucidate how this queue operates.)

To be precise:

```
Q.Enqueue(x)
   Q.Si.push(x)

Q.Dequeue(x)
   if Q.So.isEmpty() then
      while Q.Si.isEmpty() not= true
          Q.So.push( Q.si.pop() )
   return Q.So.pop()
```

Our *cost model* is that each stack PUSH or POP costs *one unit*. Starting with an empty queue, consider the following sequence of operations, and write down the cost of the operation:

```
                   Cost of operation

Q.Enqueue(9)              1

x = Q.Dequeue()           3

Q.Enqueue(5)              1

Q.Enqueue(4)        ________________

Q.Enqueue(1)        ________________

x = Q.Dequeue()     ________________

x = Q.Dequeue()     ________________
```

Part (c) Two-Stack Queue, Part B [7 MARKS]

20% automatic grade for writing *"I don't know how to answer this"* and **crossing out the entire page**.

Use the *accounting method* to *prove* that in any sequence of n queue operations, the amortized cost of ENQUEUE is at most *three units*, and the amortized cost of DEQUEUE is at most *one unit*. Your solution must clearly state the *credit invariant*, prove the invariant, and provide clear reasoning for the final amortized cost.

Question 4. Column Data Structure [10 MARKS]

20% automatic grade for writing *"I don't know how to answer this"* and **crossing out the entire page**.

Consider the abstract data type COL, which describes the pixels on one column of a screen of height M. An object S of COL is a subset of $\{1,\ldots,M\}$, which denotes those pixels that are illuminated. A *line* is a maximal[1] set of consecutive integers in S. For example, If $S = \{3,4,5,11,12,13,14,17\}$, then the column contains three lines: the line $\{3,4,5\}$ whose endpoints are 3 and 5, the line $\{11,12,13,14\}$ whose endpoints are 11 and 14, and the line $\{17\}$ whose endpoints are both 17. COL supports two operations:

- ON(x) illuminates pixel $x \in \{1,\ldots,M\}$, i.e. it adds x to S, if it is not already in S.
- ENDPOINTS(x) returns the endpoints of the line containing x. If $x \notin S$, it returns (0,0).

Describe, in English and/or pseudocode, an implementation of COL such that the *worst case sequence complexity* of m operations (ON, ENDPOINTS) is $O(m \log^* n)$, where n is the size of S, and $\log^*$ is the iterated logarithm[2]. Justify why your implementation is correct and fulfills the specified complexity.

[1] Consider a set $Q \subseteq S$ of consecutive integers. If there exists another set $Q' \subseteq S$ of consecutive integers such that $Q \subset Q'$, then Q is *not* maximal. On the other hand, if such a Q' does not exist, then Q is maximal.

[2] The iterated logarithm $\log^*$ is the number of times the logarithm function must be iteratively applied before the result is less than or equal to 1. The iterated logarithm is a *very* slowly growing function. For all values of n relevant to counting the running times of algorithms implemented in practice (n = number of atoms in the known universe), $\log^* n \leq 5$.

Question 5. DFS/BFS [20 marks]

20% automatic grade for writing *"I don't know how to answer this"* and **crossing out the entire page**.

A simple graph $G = (V, E)$ is called *bipartite* if its vertex set can be partitioned into two disjoint subsets $V = V_1 \cup V_2$, such that every edge has the form $e = (a, b)$ where $a \in V_1$ and $b \in V_2$. In the following questions, you will use two methods to check if a graph is bipartite.

Part (a) [10 marks]

A simple graph is *2-colourable* if each vertex can be assigned a color (black or white) such that *every edge has endpoints of opposite color*. A graph is bipartite if and only if the graph is 2-colourable. In particular, V_1 and V_2 are the sets of black and white vertices, respectively.

Design an algorithm that uses *depth first search* (DFS) to determine whether a graph is 2-colourable. The input to your algorithm is a simple graph $G = (V, E)$, and the boolean output should be True if and only if the graph is 2-colourable. Provide:

1. pseudocode,
2. an explaination of your algorithm in English, and
3. an analysis of the time complexity.

Part (b) [10 MARKS]

20% automatic grade for writing *"I don't know how to answer this"* and **crossing out the entire page**.

Let's consider another approach. A graph is bipartite if and only if the graph has *no* odd length cycles. Design a algorithm based on *breadth first search* (BFS) that finds an odd length cycle in a simple graph, if one exists. The input to your algorithm is a simple graph $G = (V, E)$. Your algorithm should return a sequence S of edges that form an odd length cycle, or an empty sequence if no odd length cycle is found. If the graph has multiple odd length cycles, return just one of those cycles. The operation S.APPEND(e) adds edge e to the end of the sequence S; this operation costs $O(1)$. Provide:

1. pseudocode,
2. an explaination of your algorithm in English, and
3. an analysis of the time complexity.

Question 6. Modeling with Graphs [10 marks]

20% automatic grade for writing *"I don't know how to answer this"* and **crossing out the entire page**.

Our village is comprised of n geographically scattered buildings. We also have m roads. Each road connects exactly two buildings. We have such a great set of roads, that there is always at least one route between any pair of buildings. Our town is great. Except... we pollute.

To "go green and eco-friendly," we will be establishing bike lanes along some (not necessarily all) of the existing roads. This set of lanes must reach all the town's buildings. To avoid "cycle madness" our set of bike lanes will not have any cycles (it's true... cyclists in neighboring towns can get stuck going around in loops forever!).

Help us plan our bike lanes. We'll be honest... we are *cheap*. Give us a plan that keeps the *total length* of the bike lanes to a *minimum*, please!

Part (a) [4 marks]

Model our town with a graph $G = (V, E)$. (1) what are the vertices and edges? (2) Is the graph directed or undirected? (3) Weighted or unweighed? If weighted, what is the weight of each edge?

Part (b) [3 marks]

Translate the bike lane planning problem into a graph problem. Give (1) the name of the graph problem, (2) the name of one algorithm that solves the problem, and (3) the time complexity for solving the problem using the named algorithm.

Part (c) [3 marks]

Whoops! We changed our mind. It turns out it's not a Go-Green campaign, but a Be-Healthy and Burn Calories campaign. Instead of minimizing the total length, let's *maximize the total length*. As before, we need to avoid cycle madness. Translate this revised planning problem into a graph problem. Describe (1) any changes to the graph setup, (2) the name of the graph problem, (3) the name of one algorithm that solves the problem, and (4) the time complexity for solving the problem using the named algorithm.

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.