CSE 332 Spring 2023 Final Exam

Name: ____**Sample Solution**_____

Email address (UWNetID): _____

Instructions:

- The allotted time is 1 hour and 50 minutes.
- Please do not turn the page until the staff says to do so.
- This is a closed-book and closed-notes exam. You are NOT permitted to access electronic devices including calculators.
- Read the directions carefully, especially for problems that require you to show work or provide an explanation.
- When provided, write your answers in the box or on the line provided.
- Unless otherwise noted, every time we ask for an O, Ω, or Θ bound, it must be simplified and tight.
- For answers that involve bubbling in a ◯ or ☐, make sure to fill in the shape completely.
- If you run out of room on a page, indicate where the answer continues. Try to avoid writing on the very edges of the pages: we scan your exams and edges often get cropped off.
- A formula sheet has been included at the end of the exam.

Advice

- If you feel like you're stuck on a problem, you may want to skip it and come back at the end if you have time.
- Look at the question titles on the cover page to see if you want to start somewhere other than problem 1.
- **Relax and take a few deep breaths. You've got this! :-).**

Total: 97 points

Q1: Short-answer questions (10 pts)

- For questions asking you about runtime, give a simplified, tight Big-O bound. This means that, for example, $O(5n^2 + 7n + 3)$ (not simplified) or $O(2^{n!})$ (not tight enough) are unlikely to get points. Unless otherwise specified, all logs are base 2.
- For questions with a mathematical answer, you may leave you answer as an unsimplified formula (e.g. $7 \cdot 10^3$).

We will only grade what is in the provided answer box.

a. **True** or **False**: The **worst-case** runtime of delete in a Binary Search Tree containing N elements is $\Omega(log\ N)$.

● True ○ False

b. What is the maximum number of nodes in a binary max heap with height h?

$2^{h+1} - 1$

c. i) For a Binary Min Heap, in Floyd's build heap algorithm, you (pick one) .

○ percolate elements **up** ● percolate elements **down**

ii) For a Binary Min Heap, in Floyd's build heap algorithm, you go from (pick one):

● the **bottom** (where the leaf nodes are) up to the top of the heap

○ the **top** (where the root node is) down to the bottom of the heap

d. Give the **worst-case** runtime of an increaseKey operation on a Binary Min Heap containing N values.

O(log(N))

e. What is the minimum number of nodes in an AVL tree of height 3?

7

Q2: More Short-answer questions (10 pts)

(Same instructions as for Q1)

a.What is the worst-case runtime for Kruskal's algorithm as described in lecture. (Give your answer in terms of V, E and/or d)

O(ElogE (or ElogV))

b.What is the worst-case runtime of finding all **incoming edges of a vertex** using an adjacency list representation of a graph. Use the adjacency list as described in lecture - with a linked list node for each edge, NOT what you used for P3. (Give your answer in terms of V, E and/or d)

O(V+E)

c. What is the worst-case runtime of a find operation on a hash table containing N elements, using separate chaining, where each bucket is a sorted linked list. The load factor of this table is log N.

O(N)

d. Assuming the implementations described in lecture, give the name of a **sequential** sort that is **not** in-place but is stable.

Merge Sort, Bucket Sort, or Radix Sort

e. If 1/4 of your program must be run sequentially (and the remaining 75% can be parallelized) what is the maximum speedup you would expect to get with 5 processors?

2.5

Q3: Hashing (10 pts)

For question a), insert **82, 4, 57, 13, 24, 17, 32** into the tables below. For each table, TableSize = **10**, and you should use the primary hash function h(k) = k%10. If an item cannot be inserted into the table, please indicate this and continue inserting the remaining values.

a) [3 pts] **Double Hashing Hashtable**. You should use the primary hash function: **h(k) = k % 10** and a secondary hash function: **g(k) = 2 + (k % 3)**.
If any values cannot be inserted, write them here:_____**NONE**_____

0	**32**
1	**17**
2	**82**
3	**13**
4	**4**
5	
6	**24**
7	**57**
8	
9	

b) [1 pt] What is the load factor for the table in part a)? _____**0.7**_____

c) [5 pts] You are designing a hash table to store your album collection. Each album will be represented by an `Album` object:

```
public class Album {
      public static final String UUID = "086340820472047927402740274a";
      private String title;
      private String artist;
      private List<Song> songs;

      public Album (String t, String a, List<Song> s) {
            title = t;
            artist = a;
            songs = s;
      }

      public int hashCode() {
            ...
      }
}
```

We define two `Album`s as being equal if they have the same `title`, `artist`, and `songs`. Just like in the real world, no two albums in a hashmap can have the same title or same set of songs.

For each of the `hashCode()` implementations below, fill in the appropriate bubble to indicate whether the implementation is valid or invalid.

- A hashcode is "**valid**" if the methods of a hash table implementation (e.g. HashSet) always return the correct results.
- A hashcode is "**invalid**" if the code either doesn't compile, or some methods of a hash table implementation (e.g. HashSet) would not work as expected.

In the code below, you should assume that the calls to `hashCode()` on objects other than `Albums` are implemented correctly and efficiently.

`hashCode` implementation	**valid**	**invalid**
i) `return title.hashCode() + artist.hashCode();`	●	○
ii) `return UUID;`	○	●
iii) `return UUID.hashCode();`	●	○
iv) `Random newRand = new Random();` `return newRand.nextInt();`	○	●

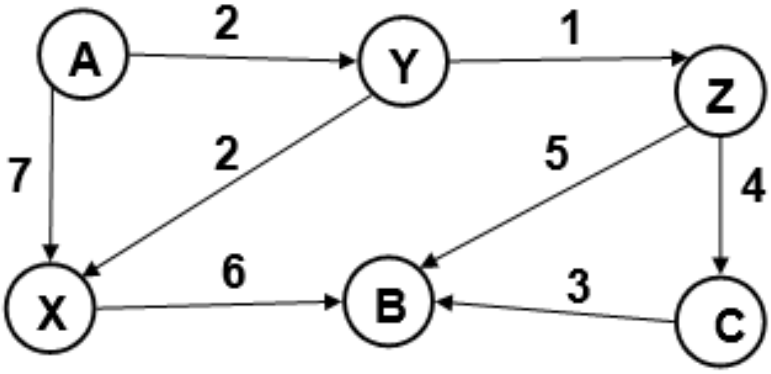
v) `int result = 0;`
`for (Song s: songs) {`
`  result += s.hashCode();`
`}`
`return result;`
	●	○

d) [1 pt] Which of the **valid** **`hashCode`** implementations in c) would you expect to have the **most collisions** (pick ONE):

○ i) ○ ii) ● iii) ○ iv) ○ v)

Q4: Graphs (19 pts)

Use the following graph for the problems on this page:



a) [2 pts] List a valid **topological ordering** of the nodes in the graph above (if there are no valid orderings, state why not).

___A____, ___Y____, ___Z____, ___X____, ___C____, ___B____

___A____, ___Y____, ___X____, ___Z____, ___C____, ___B____

___A____, ___Y____, ___Z____, ___C____, ___X____, ___B____

b) [4 pts] Step through Dijkstra's Algorithm to calculate the **single source shortest path from A** to every other vertex. Break ties by choosing the lexicographically smallest letter first; ex. if B and C were tied, you would explore B first. *Note that the next question asks you to recall what order vertices were declared known*. Make sure the final distance and predecessor are clear in the table below.

Vertex	Known	Distance	Predecessor
A	T	0	------
B	T	8	Z
C	T	7	Z
X	T	4	Y
Y	T	2	A
Z	T	3	Y

c) [1 pt] In what order would Dijkstra's algorithm mark each node as *known*?

___A____, ___Y____, ___Z____, ___X____, ___C____, ___B____

d) [1 pt] List the **shortest path** from A to B. (Give the actual path **NOT** the cost.)

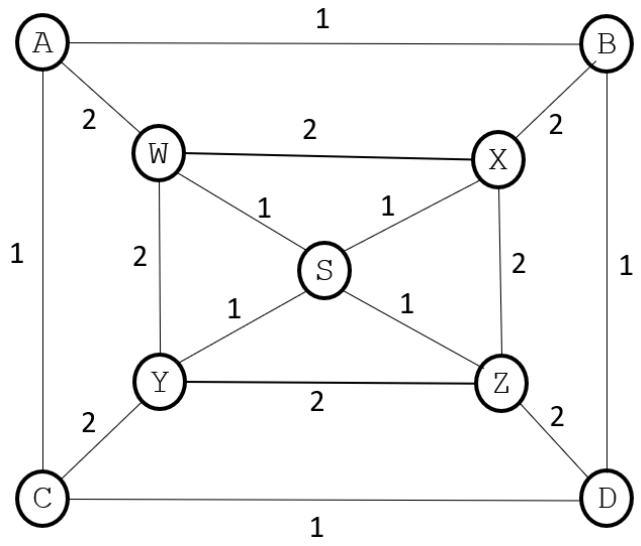
A, Y, Z, B

e) [2 pts] Is this graph **complete**? **Explain your answer in 1-2 sentences for any credit.**

○ Yes ● No

The graph is not complete because a directed edge does not exist between every pair of nodes. For example, the edge from B to Y does not exist. There is also no edge from C to Z.

Use the following graph for the problems on this page:



f) [2 pts] If we run Kruskal's algorithm, which of the following edges could be the last edge added to the Minimum Spanning Tree? **Bubble in the box for all that apply.** Assume any ties are broken randomly.

☐ SW	☐ SX	☐ SY	☐ SZ	☐ WX	☐ WY	☐ XZ	☐ YZ
■ AW	■ BX	■ CY	■ DZ	☐ AB	☐ BD	☐ AC	☐ CD

g) [2 points] If we run Prim's algorithm from vertex S, which of the following edges could be the last edge added to the Minimum Spanning Tree? **Bubble in the box for all that apply.** Assume any ties are broken randomly.

☐ SW	☐ SX	☐ SY	☐ SZ	☐ WX	☐ WY	☐ XZ	☐ YZ
☐ AW	☐ BX	☐ CY	☐ DZ	■ AB	■ BD	■ AC	■ CD

h) [1 pt] What is the **total cost** of a minimum spanning tree in the graph above? 9

i) [2 pts] ASSUMING the edges above are **unweighted**, give a valid **breadth first search** of this graph, **starting at vertex S**, using the algorithm described in lecture. **When adding elements to the data structure, you should break ties by choosing the lexicographically smallest letter first**; ex. if A and B were tied, you would add A to the data structure first. You only need to show the final breadth first search.

_S_, _W_, _X_, _Y_, _Z_, _A_, _B_, _C_, _D_

j) [2 pts] ASSUMING the edges above are **unweighted**, give a valid **depth first search** of this graph, **starting at vertex S**, using the non-recursive algorithm described in lecture. **When adding elements to the data structure, you should break ties by choosing the lexicographically smallest letter first**; ex. if A and B were tied, you would add A to the data structure first. You only need to show the final depth first search.

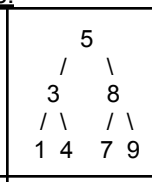
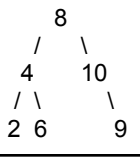
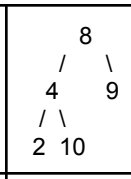
_S_, _Z_, _D_, _C_, _A_, _B_, _Y_, _X_, _W_

Q5: ForkJoin (14 pts)

In Java using the ForkJoin Framework, write code to solve the following problem:

- **Input**: A root `Node` of a valid (i.e. you may assume correct fields) binary tree of **unique** integers.
- **Output**: True iff the binary tree is a valid **Binary Search Tree (BST)**. False otherwise.

Examples:

Inputs			
Output	True	False	False
Why	All numbers are in correct ordering	10 occurs before 9	10 is greater than the root node

Notes:

- Use the given `CUTOFF` field to employ a sequential cutoff. You may use the given sequential method that determines whether a binary tree is a BST (true) or not (false).
- Make sure your code has $O(\log n)$ span and $O(n)$ work, where n is the number of nodes.
  - ***To achieve this span, you may assume the given binary tree is balanced* (similar to an AVL tree).**
- You may **NOT** use any **global data structures** or **synchronization primitives (locks).**
- Do **NOT** worry about off-by-one errors.

Fill in the underlines in the function `validateBST` below.

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;
import java.util.concurrent.RecursiveAction;

public class Main {
    public static final ForkJoinPool fjPool = new ForkJoinPool();

    private static class Node { // You should use this class
        int key;
        int height;
        Node left, right;
    }

    public static Boolean validateBST(Node root) {

        return fjPool.invoke(new Task(root, Integer.MIN_VALUE, Integer.MAX_VALUE));
    }

    /**
     * @param root   root of a valid binary tree
     * @param min    maximum value root.key can be
     * @param max    minimum value root.key can be
     * @return true  if the root is the root of a valid BST (where
     *               min <= root.key <= max) and false otherwise.
     * Note: Every node in the tree must also also have a key between min and
     *       max and be a valid BST.
     * Runs in O(n) time, where n is the number of nodes in the Binary Tree
     **/
    private static boolean sequential(Node root, int min, int max) {
        // some good code here that follows the spec :)
    }

    // Your class goes here (write it on the next page)
}
```

Write your class here:

```
public static class Task extends RecursiveTask<Boolean> {
    public static final int CUTOFF = 3;
    // Fields go here
    private final Node node;
    private final int min;
    private final int max;

    public Task(Node node, int min, int max) {
        this.node = node;
        this.min = min;
        this.max = max;
    }


    public Boolean compute() {
        if (node == null) {
            return true;
        }

        if (node.height <= CUTOFF) { // or < CUTOFF
            return sequential(node, min, max);
        }

        if (node.key < min || node.key > max) {
            // or !(min <= node.key && node.key <= max)
            // i.e. not within range
            return false;
        }

        Task l = new Task(node.left, min, node.key - 1);
        Task r = new Task(node.right, node.key + 1, max);

        l.fork();
        boolean rRes = r.compute();
        boolean lRes = l.join();
        return lRes && rRes;
    }
```

Q6: Concurrency (10 pts)

The UWBookstore class manages the bookstore's stock and balance by selling books and restocking books. Multiple threads (students) might be accessing the same `UWBookstore` object. Assume Map objects ARE THREAD-SAFE, have enough space, and operations on them will not throw an exception.

```
1  public class UWBookstore {
2      private Map<String, Integer> books;
3      private Map<String, Object> bookLocks;
4      private int balance;
5
6      // The bookstore will ONLY carry the books in bookList and you can assume that
7      // methods in the class will only be called on these books.
8      public UWBookstore(List<String> bookList) {
9          this.books = new HashMap<>();
10         this.bookLocks = new HashMap<>();
11         for (String book : bookList) {
12              this.books.put(book, 0);
13              this.bookLocks.put(book, new Object());
14          }
15          this.balance = 100;
16      }
17
18      public void sell(String book, int price) {
19
20          synchronized(bookLocks.get(book)) { // assume book exists
21
22              if (books.get(book) > 0) {
23
24                  books.put(book, books.get(book) - 1); // remove the book from stock
25                  synchronized(this) {
26                  balance += price; // The store gets $ for each book it sells.
27                  }
28              }
29
30          }
31
32      }
33
34
35      public void restock(String book, int quantity, int costEach) {
36
37          int totalCost = quantity * costEach;
38
39          synchronized(bookLocks.get(book)) { // assume book exists
40              synchronized(this) {
41              if (balance >= totalCost) {
42
43                  books.put(book, books.get(book) + quantity); // add books to stock
44
45                  balance -= totalCost; // The store must pay for these books!
46
47              }
48              }
49          }
50      }
51
52  }
```

a) [3 pts]Does the **`UWBookstore`** class above have (bubble in all that apply):

■ a race condition ☐ potential for deadlock ■ a data race ☐ none of these

If there are any problems, describe each problem selected in 1-2 sentences.

There is a data race on the `balance` instance variable with possible write-write or read-write conflict between lines 26 and/or 45 and possible read-write conflict with line 41 and either of lines 26 or 45. A data race is a race condition.

b) [3 pts] You decide to add one more method to the **`UWBookstore`** class:

```
58  public void giveawayBundle(String book1, String book2) {
59      // Removes one copy of book1 and book2 from stock
60      String firstBook = book1;
61      String secondBook = book2;
62
63      if (book1.compareTo(book2) > 0) {
64
65          firstBook = book2;
66          secondBook = book1;
67
68      }
69
70      synchronized(bookLocks.get(firstBook)) { // assume firstBook exists
71
72          synchronized(bookLocks.get(secondBook)) { // assume secondBook exists
73
74              if (books.get(book1) > 0 && books.get(book2) > 0) {
75
76                  books.put(book1, books.get(book1) - 1);
77                  books.put(book2, books.get(book2) - 1);
78
79              }
80
81          }
82
83      }
84
85  }
```

Does adding this method to the **`UWBookstore`** class **cause any new** (bubble in all that apply):

☐ a race condition ☐ potential for deadlock ☐ a data race ■ none of these

If there are any new problems, describe each problem selected in 1-2 sentences.

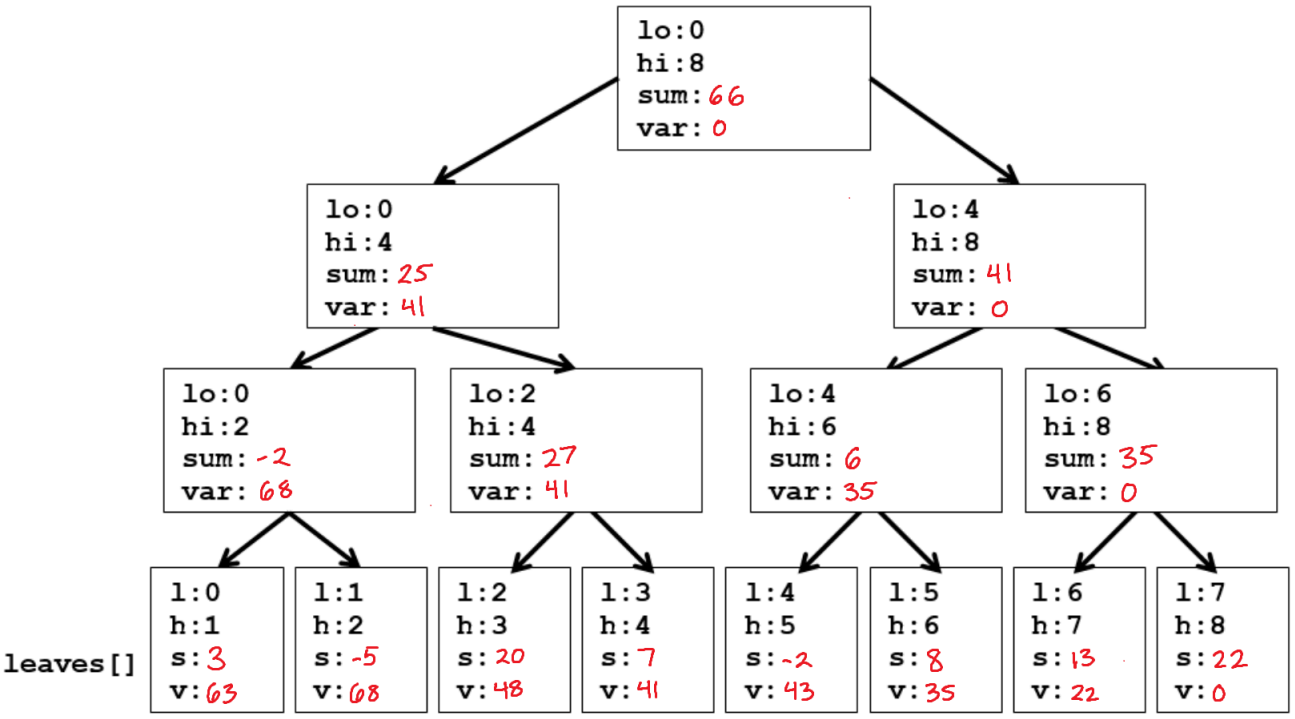
c) [4 pts] Modify the **code above in part b) and on the previous page** (draw arrows if needed) to ***allow the most concurrent access*** and to avoid all of the potential problems listed above. **For full credit you must allow the most concurrent access possible without introducing any errors or extra locks.** You should use `synchronized` appropriately to provide locking.

Q7: Parallel Suffix Sum - like Prefix, but from the Right instead! (9 pts)

Given the following array as input, perform a parallel **suffix** algorithm to fill the output array with the **sum of values contained in all of the cells to the right** (including the value contained in that cell) in the input array. The first pass of the algorithm is similar to the first pass of the parallel prefix code you have seen before. Do not use a sequential cutoff. For example, for
input: {3, 14, -1, 4, 5, 5, 6, 1},
output should be: {37, 34, 20, 21, 17, 12, 7, 1}.

Note: This question continues on the next page.

a) [5 pts] Fill in the values for **`sum`**, **`var`**, and the **`output`** array in the picture below. Note that problems b-e, on the next page, ask you to give the formulas you used in your calculation.



Index	0	1	2	3	4	5	6	7
input	3	-5	20	7	-2	8	13	22
output	66	63	68	48	41	43	35	22

Give formulas for the following values where `p` is a reference to a non-leaf tree node and `leaves[i]` refers to the leaf node in the tree visible just above the corresponding location in the `input` and `output` arrays in the picture on the previous page.

b) [1 pt] Give pseudocode for how you assigned a value to `leaves[i].sum`

```
leaves[i].sum = input[i]
```

c) [1 pt] Give code for assigning `p.left.var`.

```
p.left.var = p.right.sum + p.var
```

d) [1 pt] Give code for assigning `p.right.var`.

```
p.right.var = p.var
```

e) [1 pt] How is `output[i]` computed? Give exact code assuming `leaves[i]` refers to the leaf node in the tree visible just above the corresponding location in the `input` and `output` arrays in the picture above.

```
output[i] = leaves[i].sum + leaves[i].var
```
(also accepted: `input[i] + leaves[i].var`**)**

Q8: Sorting (9 pts)

.

(a-d) [4 pts] You are given an initial array: [22, 10, 14, 37, 14, 4, 3].

For each of the arrays shown below, indicate whether they could represent **one or more** of the given sorting algorithms at any point (after an iteration has completed) during the sorting algorithm's execution. If none of the sorting algorithms could have the given state, select **None**.

Note: For Radix sort, if you have following items in buckets 0-9:
`0:[a], 1:[b, c], 2:[d], 3:[], 4:[], 5:[e, f], 6:[], 7:[g], 8:[], 9:[]`
we would write the resulting array as: `[a, b, c, d, e, f, g]`

a) [3, 4, 14, 37, 14, 10, 22]

☐ Insertion sort ■ Selection sort ☐ Radix sort ☐ None

b) [10, 22, 3, 14, 14, 37, 4]

☐ Insertion sort ☐ Selection sort ☐ Radix sort ■ None

c) [10, 22, 3, 14, 14, 4, 37]

☐ Insertion sort ☐ Selection sort ■ Radix sort ☐ None

d) [10, 14, 14, 22, 37,4, 3]

■ Insertion sort ☐ Selection sort ☐ Radix sort ☐ None

---

e) [1 pt] True or False: Heap sort is a **stable** sort.

○ True ● False

Explain why or why not in 1-2 sentences.

If two objects with equal keys appear in the same order in sorted output as they appear in the input array.

f) [2 pts] Give the *recurrence* for PARALLEL Merge Sort (parallel sort & sequential merge) - **best case**

T(n) = T(n/2) + O(n) (or + n, or + c1*n), (can add + c2)

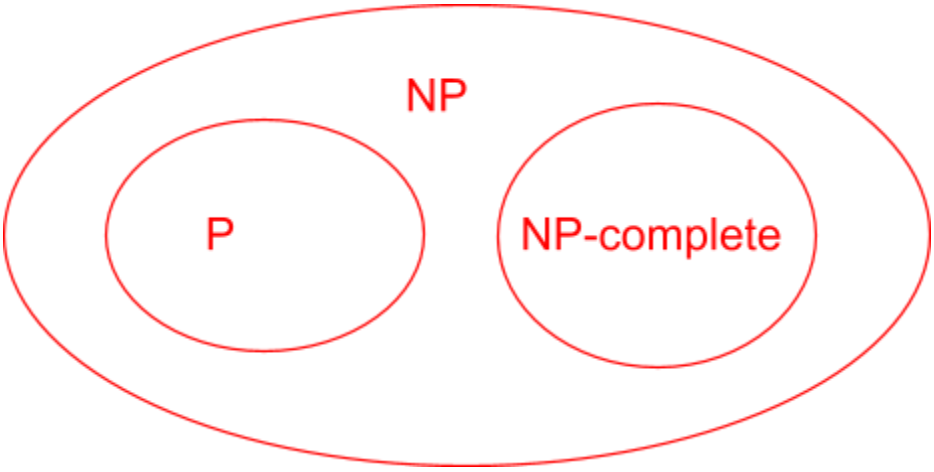
g) [2 pts] Give the *recurrence* for SEQUENTIAL Quicksort - **best case**.

T(n) = 2T(n/2) + O(n) (or + n, or + c1*n), (can add + c2)

Q9: P/NP (6 pts)

a) [1 pt] "NP" stands for ____**Nondeterministic Polynomial**_____

b) [2 pt] Draw a diagram demonstrating how (we are pretty sure) the sets P, NP, and NP-Complete, overlap / don't overlap with each other.



For the following problems, bubble in **ALL** the sets each problem belongs to:

c) [1 pt] Given a weighted, undirected graph, determine if a minimum spanning tree of cost *k* exists.

☐ NP-complete ■ NP ■ P ☐ none of these

d) [1 pt] Given a weighted, directed graph that has an edge between each vertex, determine whether there is a path of total cost *k* that begins and ends at the same vertex and goes through each **vertex** exactly once.

■ NP-complete ■ NP ☐ P ☐ none of these

e) [1 pt] Given an undirected graph, determine whether there is a path that begins and ends at the same vertex and goes through each **edge** exactly once.

☐ NP-complete ■ NP ■ P ☐ none of these

This is a blank page! Enjoy!