Practice Questions for COMP285 Final

The following are questions meant to help you practice, and cannot be submitted for a grade.

Important Notes

- *It is meant to give you a chance to do some practice questions after having reviewed the slides, quizzes, in-class exercises, homeworks, etc.*

- *It should give a rough sense of some ways questions might be posed, though there's no guarantee that the actual final will have the exact same format (at a minimum, one difference is that the actual final will show the point values associated with the questions).*

- *It should give a rough idea of the level of mastery expected generally, though more/less mastery may be expected for any given topic.*

Thanks for reading the notes above - the big picture thing is that I want to be sure you use this resource appropriately, while at the same time **do not neglect the many other more comprehensive resources**!

Measuring Performance

1. What is the worst-case time complexity and space complexity of the below? Remember to provide a tight upper bound with Big-O, and justify your answer.

```
algorithm findSumOfUniqueOccurrences
  input: vector<int> nums
  output: sum of each element in nums that appears exactly once

  numOccurrences = new unordered_map of int to int
  for each element n in nums
    if numOccurrences.contains(n)
      numOccurrences[n] += 1
    else
      numOccurrences[n] = 1

  answer = 0
  for each key-value pair k, v in numOccurrences
    if v == 1
      answer += k
  return answer
```

2. What is the worst case **time complexity** and **space complexity** of the below? Remember to provide a tighter upper bound with Big-O, and justify your answer.

```
void doSomething(const std::vector<int>& nums) {
  for (int i = 0; i < nums.size(); i++) {
    for (int j = 0; j < 10; j++) {
      std::cout << "hi" << std::endl;
    }
  }

  for (int i = 0; i < 10; i++) {
    for (int j = 0; j < nums.size(); j++) {
      for (int k = j; k < nums.size(); k++) {
        std::cout << "hello" << std::endl;
      }
    }
  }
}
```

3. Bob has analyzed the best-case time complexity for binary search as O(1) because there could be an array with one element in it containing exactly the desired target. If Bob is correct, explain. If not, what is the correct best-case time complexity and what input produces it?

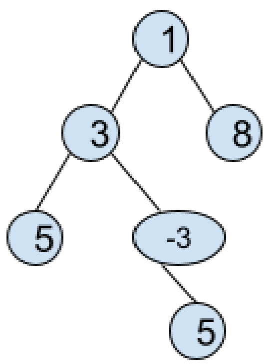
Trees

4. Order the following data structures from most general to most specific:
   Trees, Binary Trees, Graphs, Binary Search Trees

For questions 5 - 8, refer to the information below.

We want to write findSumOfTreeLeaves, which finds the sum of the values of all leaf nodes given the root of a binary tree.

5. What should findSumOfBinaryTreeLeaves return when called on the tree below?



6. What would an in-order traversal print out for the tree above?

7. Complete `findSumOfBinaryTreeLeaves` below **with recursion**. (Do not worry about the distinction between `->` and `.` for accessing variables / functions in the pseudocode. Also, you can access fields as follows):
   - root.isLeaf - returns a boolean for whether or not the node root is a leaf
   - root.value - returns the value of node root.
   - root.left - returns the left child.
   - root.right - returns the right child.

```
algorithm findSumOfBinaryTreeLeaves
  input: TreeNode root which represents the root of the tree
  output: the sum of the value of all tree leaves

if root == nullptr
  return 0
```

8. Assume the function above is run on a balanced **Binary Search Tree.** What is the **time complexity** and **space complexity** of this recursive algorithm? Provide a tight upper-bound with Big-O in terms of n (number of nodes in the tree) and **justify your answer**.

9. Explain whether it is more time efficient to be running “search” on a balanced BST or unbalanced BST. Provide a tight upper-bound with Big-O in terms of n (number of nodes in the tree) for each case.

10. Provide a tight upper-bound with Big-O in terms of n (number of nodes in the tree) for the time complexity of the **best-case** for search on a BST. Describe an input for which this happens.

Graphs

11. Which of the below are true? Select all that apply.
    - Bellman-Ford can be used even if there are negative-weight edges.
    - Dijkstra's can be used even if there are negative-weight edges.
    - When both can apply, we should prefer Dijkstra's over Bellman-Ford for time efficiency.
    - Dijkstra's is an example of a Greedy Algorithm.

For questions 12 - 15, refer to the below.

There are n cities, some of which are connected by roads. If there is a road from a to b, and from b to c, then we say that a and c are connected indirectly. A province is a group of directly or indirectly connected cities, and no other cities outside the group are reachable.

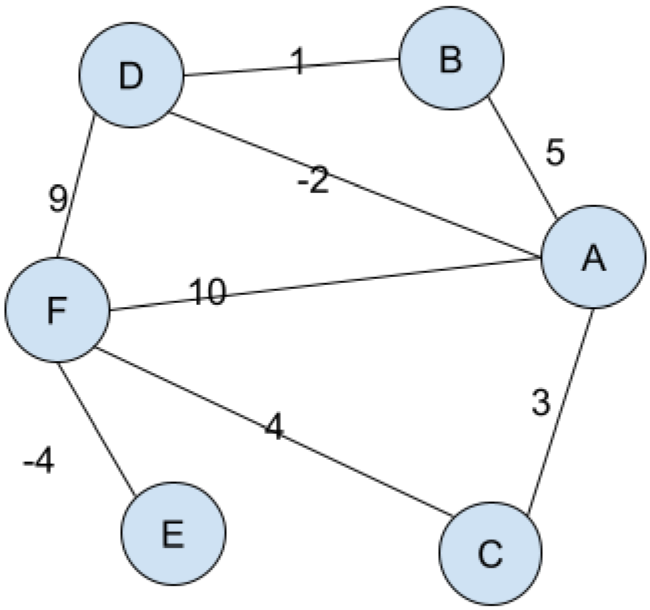
12. In order to solve this problem, we can represent this as a graph. What are the nodes and edges?

13. We want to find the total number of provinces. How would you solve this problem leveraging graph algorithms we've covered? Explain which algorithm(s) you would use in words AND how you would use it.

14. Now suppose the roads have associated distances, and we want to find the cost of the shortest path between a city m and n, if it exists. Which algorithm(s) that we've covered would be most appropriate and efficient to solve this problem? Explain which algorithm(s) you would use in words AND how you would use it.

15. One province leader is interested in building an underground tunnel between the 2 closest cities. Which algorithm(s) that we've covered would be most appropriate and efficient to solve this problem? Explain which algorithm(s) you would use in words AND how you would use it.

MSTs



16. Given the graph above, which of the following are valid *Spanning Trees*? Note that we are not looking for *Minimum* Spanning Trees. Select all that apply.
    - A-D, A-F, B-C, C-F
    - A-B, A-D, A-F, C-F, E-F
    - A-C, B-D, C-F, D-F, E-F
    - A-D, A-F, C-F, D-F, E-F
    - A-C, A-F, B-D, C-F, D-F, E-F

17. Given the above graph, what's the weight of the **minimum** spanning tree?

For the following questions, assume we run the following version of Prim's on the graph above.

```
algorithm Prims
  Input: Weighted, Undirected, connected Graph G=(V,E) with edge weights
wₑ
  Output: A Tree T=(V,E'), with E'⊆E that minimizes the edge weight sum

  for all u ∈ V :
    cost(u) = ∞
    prev(u) = nil
```

```
Pick any initial node u₀
cost(u₀) = 0

unvisited = makequeue(V) (priority queue, using cost-values as keys)
while unvisited is not empty:
  v = extractmin(unvisited)
  for each {v, z} ∈ E:
    if cost(z) > w(v, z) and unvisited.contains(Z):
      cost(z) = w(v, z)
      prev(z) = v
      decreasekey(unvisited, z)
```

18. Which of the following will be the correct cost array after two iterations of the while loop? **Assume we start at node D**. Select one.
    - cost(A) = -2, cost(B) = 1, cost(C) = inf, cost(D) = 0, cost(E) = inf, cost(F) = 9
    - cost(A) = -2, cost(B) = 5, cost(C) = 3, cost(D) = 0, cost(E) = inf, cost(F) = 9
    - cost(A) = -2, cost(B) = 1, cost(C) = 3, cost(D) = 0, cost(E) = inf, cost(F) = 8
    - cost(A) = -2, cost(B) = 1, cost(C) = 3, cost(D) = 0, cost(E) = inf, cost(F) = 9

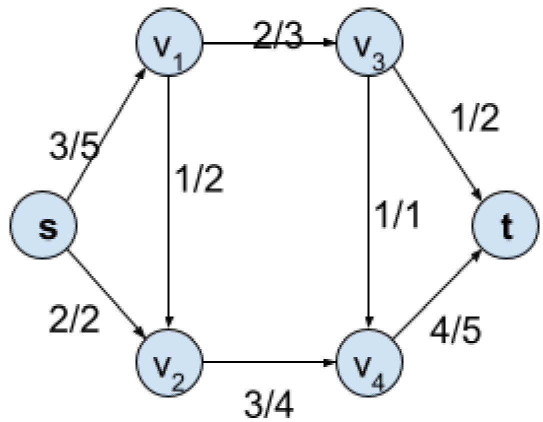
19. If you run Prim's algorithm **starting at node A**, list the sequence of nodes you'll remove from the priority queue in order.

20. What's the order of edges visited by Kruskal's? Include edges that are skipped (put "skipped" next to the edge if applicable), and assume we stop after picking |V|-1 edges.

21. In general, which of the following is true about Kruskal's and Prim's? Select all that apply.
    - Kruskal's incorrectly outputs on graphs with negative edges
    - Kruskal's correctly outputs on graphs with negative cycles
    - Prim's will produce a correct result on graphs with negative cycles
    - Prim's and Kruskal's will output different MSTs on the same graph even if they follow the same tie-breaking convention (smallest labeled nodes/edges first)

Network Flow

Consider the flow network below.

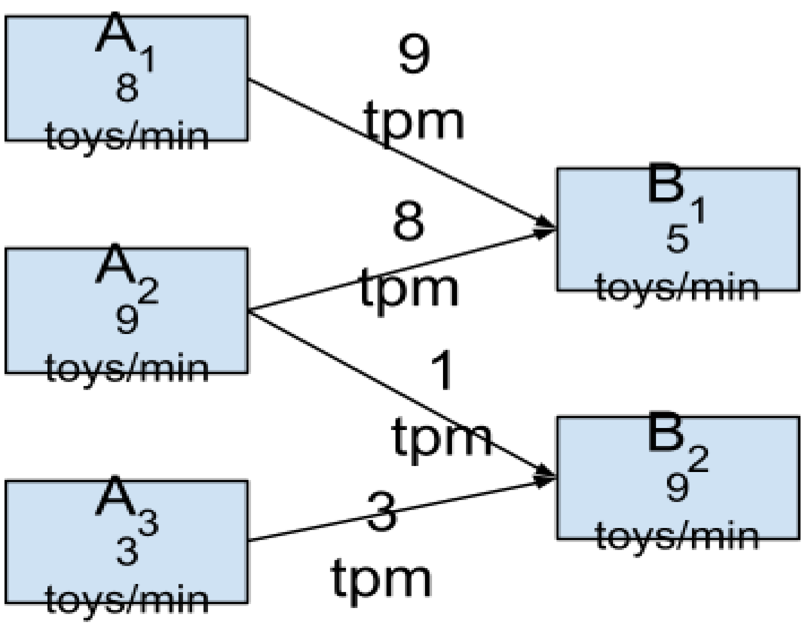


22. Draw the edges and weights of the residual graph for the flow network below.

23. Identify an augmenting path in the residual graph if there is one. Note: we are looking for a sequence of vertex labels here that starts with s and ends at t.

24. Update the flows on the original network according to this new improved flow.

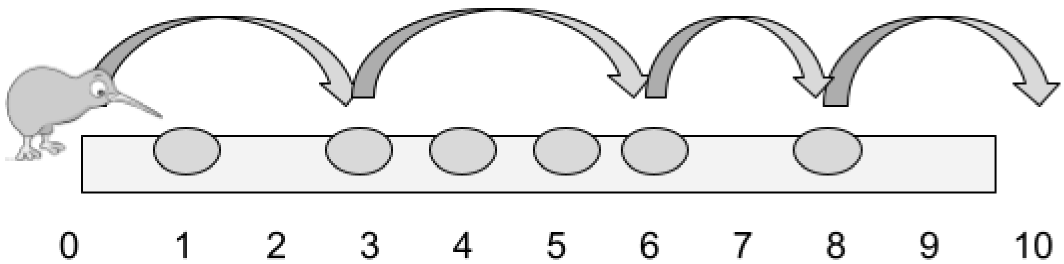
25. Suppose we are optimizing a toy manufacturing facility. For a toy to be successfully created, it must go through an A-machine which creates the toy and then a B-machine which packages it. We also have conveyor belts that can move toys from A-machines to B-machines. Each conveyor belt has a max speed given in toy(s) per minute (tpm). Each machine also has a max tpm that they produce. Draw a transformed flow network for the toy factory below for which the max flow will tell us how many toys we can produce when running optimally. You do not need to label the vertices, but there must be a source s and a sink t, and all edge capacities must be specified.



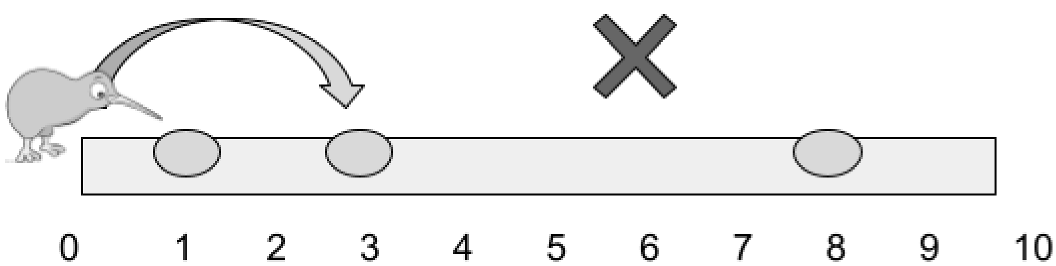
26. What does the max flow on the transformed graph represent?

Greedy, Recursion

A kiwi bird is trying to cross a river by hopping across rocks in the stream. The kiwi can hop at most k feet at a time, and rocks are placed at various parts of the stream. For example, if our kiwi can hop at most 3 feet, then it'll take 4 hops to make it across this river on the following set of rocks:



If the rocks were positioned differently, our kiwi wouldn't be able to make it across since it wouldn't be able to jump far enough to safely make it to the next rock (assuming k = 3).



We want to write an algorithm named `findMinBirdHops`, which takes in as input the `k` and vector<int> `rocks` that describe the rock positions and the distance to the far shore, and computes the minimum number of hops needed to cross the river. It should return -1 if it isn't possible for our bird to safely make it to the other side of the river. Here are some examples:

```
// should output 4 (start, rock 3, rock 6, rock 8, end)
findMinBirdHops(3, {0, 1, 3, 4, 5, 6, 8, 10})

// should output 3 (start, rock 4, rock 8, end)
findMinBirdHops(4, {0, 1, 3, 4, 5, 6, 8, 10})

// should output -1 (after rock 3, no rocks near enough to continue)
findMinBirdHops(3, {0, 1, 3, 7, 8, 10})
```

Note: The kiwi can always hop less than k feet. The first element in the array is always 0 (the start). The last element in the array should always be the distance to the far side of the river (the end).

27. What should `findMinBirdHops` output in this case?

    ```
    findMinBirdHops(3, {0, 1, 3, 4, 7, 8, 11, 12})
    ```

28. What should birdHops output in this case?

    ```
    findMinBirdHops(5, {0, 4})
    ```

29. What should birdHops output in this case?

    ```
    findMinBirdHops(3, {0, 4})
    ```

30. Describe a greedy algorithm that solves this problem. You do not need to write code, but we will expect you to be detailed enough so that we could design an algorithm from your response. Please include how you would update and return the minimum # of hops.

31. Now let's try to write a brute-force recursive version. Given the following code, fill in the blanks so that it will correctly compute the minimum hops.

```
algorithm findBirdHops
```

```
  input: int k, vector<int> of N rocks
  output: optimal # of bird hops
return birdHopsHelper(______, ________, _______)

algorithm findBirdHopsHelper
  input: int k, vector<int> of N rocks, int index
  output: <left unspecified for this question>
if _________
  return 0
minHops = MAX_INT // predefined constant
for i = index + 1, index + 2, … N - 1
  if rocks[index] + k >= rocks[i]
    result = _____________
  if result != -1
    minHops = min(minHops, 1 + result)
if minHops == MAX_INT
  return ______
return minHops
```

Greedy, Dynamic Programming

Consider the following problem:

> Imagine you had a vending machine that dispensed `n` different types of snacks. Snack `i` has `calories[i]` calories and costs `price[i]` dollars. Assume the vending machine will never run out of any snack (i.e. you can buy as many of one type of snack as you would like). Given `X` dollars, what's the most amount of calories you can buy from the machine?

For example, given the following array `calories` and array `price`, if X = 10, then the total calories we could buy is 48 (buy the `i=0` snack once and the `i=3` snack twice).

```
calories = {30, 14, 16, 9}
price =    {6, 3, 4, 2}
```

32. Consider the following greedy algorithm:

    Pick the snack that has the best calorie to price ratio and buy as many of them as possible until you no longer have enough money to do so. Then buy the snack with the next best ratio as much as possible. Each time, add the total calories gained from the purchases to a `totalCalories` variable. Repeat this process until you don't have enough money to buy any snack. Return `totalCalories`. The calorie to price ratio of snack `i` is `calories[i]/price[i]`.

    Given `calorie` and `price` arrays above, what would this algorithm output when `X = 10`?

33. Now consider another greedy algorithm:

    Pick the snack that has the most calories and buy as many of them as possible until you no longer have enough money to do so. Then buy the snack with the next most calories as much as possible. Each time, add the total calories gained from the purchases to a `totalCalories` variable. Repeat this process until you don't have enough money to buy any snack. Return `totalCalories`.

    Given `calorie` and `price` arrays above, what would this algorithm output when `X = 10`?

34. Using the answers from above, describe why both greedy approaches won't work in this case.

Now let's examine a Dynamic Program algorithm to solve this problem.

35. Let's say we initialize an array of zeros called `C` of length `X+1`. How should we define `C[j]`? We are looking for a sentence which describes what C[j] represents in words.
   - `C[j]` is the most amount of calories you can get by buying only snack `j`
   - `C[j]` is the most amount of calories you can get with `j` dollars
   - `C[j]` is the calories earned if you add the calories of all snacks from 0 to `j`
   - `C[j]` is the total price of buying one of each snack from 0 to `j`
   - None of the above

36. What should our base case(s) be?

37. Describe the recursive subproblem for this problem in the blank below.

```
C[j] = ___________________________________
        for all i from 0 to n-1
```

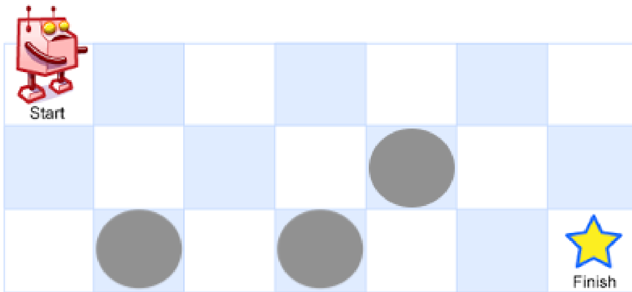
(Recall that n is the # of types of snacks, aka price.size())

38. What special conditions, if any, do we need to watch out for? What order should we solve the subproblems in, and what will we return as our final answer?

39. What is the time complexity of this dynamic programming approach?

40. We can also solve dynamic programming questions top-down with memoization. In your own words, explain what memoization is and when it is useful.

Dynamic Programming



Consider the following problem:

> A robot is located at the top-left corner of a m x n grid (marked 'Start' in the diagram below). The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below). But, there are some squares which are magnet traps (gray circles) that the robot cannot step on. For inputs, we are given the dimensions m and n, as well as a vector<pair<int, int>> representing the coordinates of all magnet traps. How many possible unique paths are there?

41. Let's say we initialize a 2D table of size m by n. How should we define M[i][j] in words?

42. What should our base case(s) be?

43. Describe the recursive subproblem for this problem.

    `M[i][j] =` ________________________________

44. What special conditions, if any, do we need to watch out for? What order should we solve the subproblems in and what should we return as our final answer?

45. What is the time complexity of this dynamic programming approach?

P, NP, and More

46. Given a problem that does not have a polynomial time verifier, which complexity class must this problem belong to? Select all that apply.
    - P
    - NP
    - NP-Complete
    - None of the Above

47. If a problem is in NP, which of the following COULD be true? Select all that apply.
    - The solution to this problem could be verifiable in polynomial time
    - The problem could be solved in polynomial time
    - The problem can only be verified in exponential time

48. If P = NP, which of the following statements are true? Select all that apply.
    - The best algorithms for NP problems will be exponential.
    - All problems in NP will be solvable in polynomial time.
    - Password encryption would be broken.