Name: ________________________

PROBLEM 1: Evaluating Implementations (20 points)

This semester, you have learned to think through how the choices you make in your implementations can influence efficiency, storage, and side effects. The problems below show two different implementations that seek to achieve the same result. For each, please compare them on efficiency (computational complexity) and how much secondary storage they use, and indicate whether there are any side effects.

PART 1: Consider the two following implementations of splice, labeled A and B.

Splice Implementation A:

```
1  public class DynamicArray {
2      char[] arr;
3  
4      /**
5       * Splices the given array into this DynamicArray at the specified index.
6       *
7       * @param spliceIn The array to splice into the current array.
8       * @param atIndex The index at which to insert the new elements.
9       * @return A new array containing the combined elements.
10      */
11     public char[] splice(char[] spliceIn, int atIndex){
12         int newLen = this.arr.length + spliceIn.length;
13         char[] newArr = new char[newLen];
14 
15         for(int i = 0; i < atIndex; i++){
16             newArr[i] = this.arr[i];
17         }
18 
19         for(int i = 0; i < spliceIn.length; i++){
20             newArr[atIndex + i] = spliceIn[i];
21         }
22 
23         for(int i = atIndex + spliceIn.length; i < newLen; i++){
24             newArr[i] = this.arr[i - spliceIn.length];
25         }
26 
27         return newArr;
28     }
29 }
30 
```

Name: ________________________

Splice Implementation B:

```
1  public class DynamicArray {
2      char[] arr;
3
4      /**
5       * Splices the given array into this DynamicArray at the specified index.
6       *
7       * @param spliceIn The array to splice into the current array.
8       * @param atIndex The index at which to insert the new elements.
9       */
10     public void splice(char[] spliceIn, int atIndex){
11         int newLen = this.arr.length + spliceIn.length;
12         char[] newArr = new char[newLen];
13
14         for(int i = 0; i < atIndex; i++){
15             newArr[i] = this.arr[i];
16         }
17
18         for(int i = 0; i < spliceIn.length; i++){
19             newArr[atIndex + i] = spliceIn[i];
20         }
21
22         for(int i = atIndex + spliceIn.length; i < newLen; i++){
23             newArr[i] = this.arr[i - spliceIn.length];
24         }
25
26         this.arr = newArr;
27     }
28 }
29
```

Questions for Comparing Splice Implementations:

What is the **major difference** between implementation A and implementation B? For full credit, try to use terminology we used in class.

B mutates the original array in place
A makes, edits & returns a copy

Which of the following statements is true about the **computational complexity** of these two splice implementations? Please **circle** your choice.

I. Implementation A is more efficient than implementation B

II. Implementation B is more efficient than implementation A

III. These two implementations have similar computational complexities (circled)

Name: ________________________

Which of the following statements is true about the **secondary storage** used in each of these two splice implementations? Please **circle** your choice.

I. Implementation A uses secondary storage

II. Implementation B uses secondary storage

III. Both implementations use secondary storage

IV. Neither implementation uses secondary storage

Which of the following statements is true about the **side effects** of these two splice implementations (e.g., effects on the internal state of the object)? Please **circle** your choice.

I. Implementation A has side effects

II. Implementation B has side effects

III. The implementations both have side effects

IV. Neither implementation has side effects.

Name: ________________________

PART 2: Consider the three following implementations of append, then answer the questions below.

Append Implementation A:

```
1  public class AppendClass {
2      char[] arr;
3
4      public void append(char[] toAppend){
5          int newLen = this.arr.length + toAppend.length;
6          char[] newArr = new char[newLen];
7
8          for(int i = 0; i < this.arr.length; i++){
9              newArr[i] = this.arr[i];
10         }
11
12         for(int i = 0; i < toAppend.length; i++){
13             newArr[i + arr.length] = toAppend[i];
14         }
15
16         this.arr = newArr;
17
18     }
19 }
20
```

There are no questions to answer on this page

Name: ________________________

Append Implementation B:

```
1  public class AppendClass {
2      SLL<Character> list;
3
4      public void append(SLL<Character> toAppend){
5          SLL<Character> newList = new SLL<>();
6
7          // Copy old list, starting with head node
8          NodeSL<Character> currentRef = list.head;
9          NodeSL<Character> currentNew = new NodeSL<>(currentRef.getData(), next:null);
10         newList.head = currentNew;
11
12         while(currentRef.getNext() != null){
13             currentRef = currentRef.getNext();
14             currentNew.setNext(new NodeSL<>(currentRef.getData(), next:null));
15             currentNew = currentNew.getNext();
16         }
17
18         // Append new data
19         currentRef = toAppend.head;
20         while(currentRef != null){
21             currentNew.setNext(new NodeSL<>(currentRef.getData(), next:null));
22             currentNew = currentNew.getNext();
23             currentRef = currentRef.getNext();
24         }
25
26         this.list = newList;
27     }
28 }
29
```

There are no questions to answer on this page

Name: ________________________

Append Implementation C:

```
1  public class AppendClass {
2      SLL<Character> list;
3
4      public void append(SLL<Character> toAppend){
5
6          // Find tail of this.list
7          NodeSL<Character> currentRef = list.head;
8          while(currentRef.getNext() != null){
9              currentRef = currentRef.getNext();
10         }
11
12         // Append new data
13         currentRef.setNext(toAppend.head);
14
15     }
16 }
17
```

Append implementation D:

```
1  public class AppendClass {
2      SLL<Character> list;
3
4      public SLL<Character> append(SLL<Character> toAppend){
5          SLL<Character> newList = new SLL<>();
6          newList.head = list.head;
7
8          // Find tail of list
9          NodeSL<Character> currentRef = list.head;
10         while(currentRef.getNext() != null){
11             currentRef = currentRef.getNext();
12         }
13
14         // Append new data
15         currentRef.setNext(toAppend.head);
16
17         return newList;
18     }
19 }
20
```

Name: ________________________

Questions for Comparing Append Implementation A vs B:

What is the **major difference** between Append Implementation A and B? For full credit, please use terminology we used in class.

A uses an array implementation & B uses a linked list implementation

Which of the following statements is true about the **computational complexity** of these two append implementations? Please **circle** your choice.

I. Implementation A is more efficient than implementation B

II. Implementation B is more efficient than implementation A

III. These two implementations have similar computational complexities

Which of the following statements is true about the **secondary storage** used in each of these two splice implementations? Please **circle** your choice.

V. Implementation A uses secondary storage

VI. Implementation B uses secondary storage

VII. Both implementations use secondary storage

VIII. Neither implementation uses secondary storage

Which of the following statements is true about the **side effects** of these two append implementations (e.g., effects on the internal state of the object)? Please **circle** your choice.

I. Implementation A has side effects

II. Implementation B has side effects

III. The implementations both have side effects

IV. Neither implementation has side effects.

Questions for Comparing Append Implementation B vs C:

What is the **major difference** between implementations B and C? For full credit, please use the terminology we used in class.

Name: ________________________

B makes a copy + then appends ✓copy style
C appends in place ← transfer style

Which of the following statements is true about the **computational complexity** of these two implementations? Please **circle** your choice.

I. Implementation B is more efficient than implementation C
II. Implementation C is more efficient than implementation B
III. These two implementations have similar computational complexities

you can argue either of these

Which of the following statements is true about the **secondary storage** used in each of these two splice implementations? Please **circle** your choice.

I. Implementation B uses secondary storage
II. Implementation C uses secondary storage
III. Both implementations use secondary storage
IV. Neither implementation uses secondary storage

Which of the following statements is true about **the side effects** of these two implementations (e.g., effects on the internal state of the object)? Please **circle** your choice.

I. Implementation B has side effects
II. Implementation C has side effects
III. The implementations both have side effects
IV. Neither implementation has side effects.

Questions for Comparing Append Implementation C vs D:

What is the **major difference** between implementations C and D? For full credit, please use the terminology we used in class.

Both are mutating, but C does the append operation in place, while D (confusingly) returns a shallow copy of the orig list

Name: ________________________

Which of the following statements is true about the **computational complexity** of these two implementations? Please **circle** your choice.

IV. Implementation C is faster than implementation D

V. Implementation D is faster than implementation C

VI. These two implementations have similar computational complexities

Which of the following statements is true about the **secondary storage** used in each of these two splice implementations? Please **circle** your choice.

V. Implementation C uses secondary storage

VI. Implementation D uses secondary storage ← can also argue this b/c it makes a new list object

VII. Both implementations use secondary storage

VIII. Neither implementation uses secondary storage

Which of the following statements is true about **the side effects** of these two implementations (e.g., effects on the internal state of the object)? Please **circle** your choice.

V. Implementation C has side effects

VI. Implementation D has side effects

VII. The implementations both have side effects

VIII. Neither implementation has side effects.

Both change the original object but D has much worse side effects

Name: ________________________

PROBLEM 2: Selecting a Data Structure (18 points)

Imagine you are developing a program to help you manage large amounts of data. For each problem below, you are asked to indicate which data structure would be best suited to your implementation.

For each question, you are provided a few different options. Indicate your *first choice* and *last choice* data structures and explain why. If you have two valid options, please make sure to address computational complexity.

ANSWER THREE OF THE FOUR PROBLEMS BELOW. Write SKIP on the one you are skipping.

1. You want to be able to search for integers efficiently. (For this one, assume the integers are already sorted).

   Options: Array Linked List Binary Tree

   A. First choice data structure: Binary Tree or Array

   Why? Either lets you use binary search faster to add/remove elements with Binary tree

   B. Last choice data structure: Linked list

   Why? Linear search is the only option for a linked list

Name: ________________________

2. You want to implement a stack from scratch. You want to maximize the efficiency of push and pop operations.

Options: Array Linked List Hash Set

A. First choice data structure: Linked List (or array can work if you know max size)

Why? Fast add/remove at head, no risk of overflow. If you can mitigate overflow issues, array has similar complexity

B. Last choice data structure: Hash Set

Why? unordered

3. At your ITS summer internship, you are tasked with creating a tool to track the 99 numbers and names of all Smith students.

Options: Hash Set Hash Map Lookup Table

A. First choice data structure: Hash map

Why? Efficient use of space, fast lookup (not always exactly O(1) if you have collisions but << O(n) if you use an appropriate hash function

B. Last choice data structure: Hash set

Why? No way to associate key-value pairs

Name: ________________________

4. You want to keep integers stored in sorted order at all times as new ones are added. Efficient insertion matters.

   Options: Array Linked List Binary Tree

   A. First choice data structure: Binary Tree

   Why? this will allow us to lookup & insert quickly O(log(n))

   B. Last choice data structure: Array

   Why? O(n) insertions even though it allows efficient lookup O(log(n))

Name: ________________________

PROBLEM 3: Revisiting Recursion (32 points)

This semester, we worked on a classic problem:

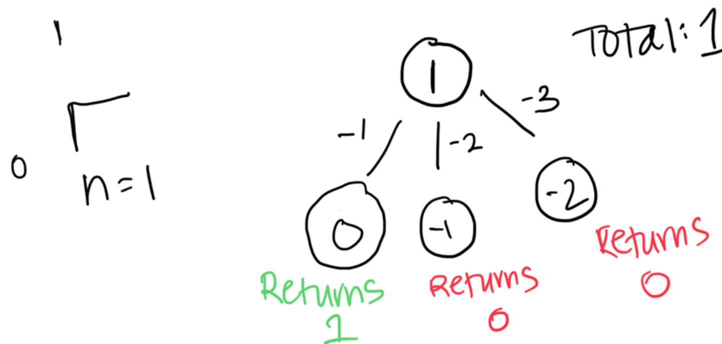
A child is running up a staircase with n steps. She can hop either 1 step, 2 steps, or 3 steps at a time. Assume she must start on the first step and finish at the last step. Describe, in pseudocode, a potential recursive approach that would let you count the number of possible ways that the child can climb the stairs.

Here, we will revisit this problem using concepts from the second half of the semester.

A. Here is solution in Java to count the number of ways to climb *n* stairs. Inside the **boxes to the left** of the blocks of code, indicate whether each block represents a **base case (BC)** or one or more **recursive calls (RC)**.

```
3      public static int climb(int n){
4
5  [BC]    if(n == 0){
6              return 1;
7          }
8
9  [BC]    else if(n < 0){
10             return 0;
11         }
12
13 [RC]    else{
14             return climb(n-1) + climb(n-2) + climb(n-3);
15         }
16     }
```

B. Pure recursive problem solving, like what is shown above, can be depicted as implementing depth-first search over a decision tree. You can imagine representing the solution to n=1 as:

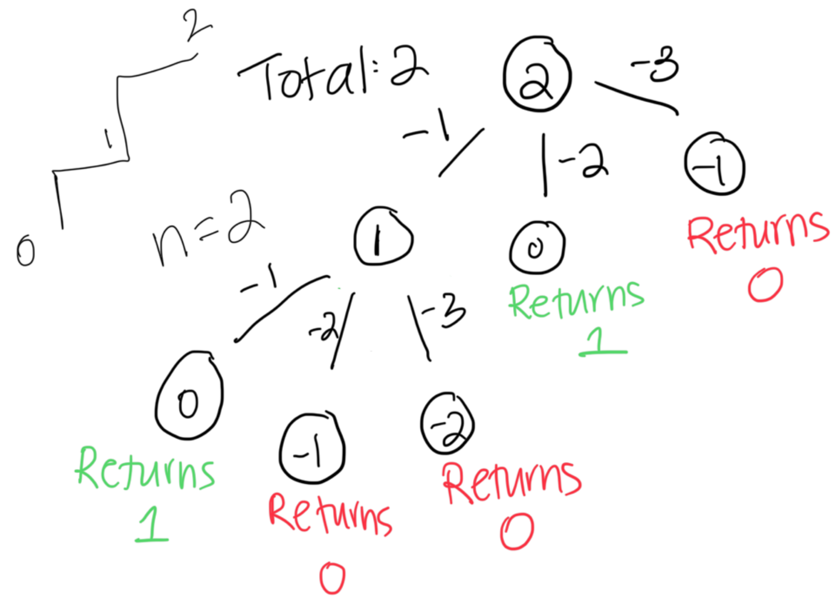


Here, each node represents a call to our climb() method. The number indicated inside the node is the number passed to the function, e.g., the root node represents climb(1). When leaf nodes are reached, the value returned is indicated.

Name: ________________________

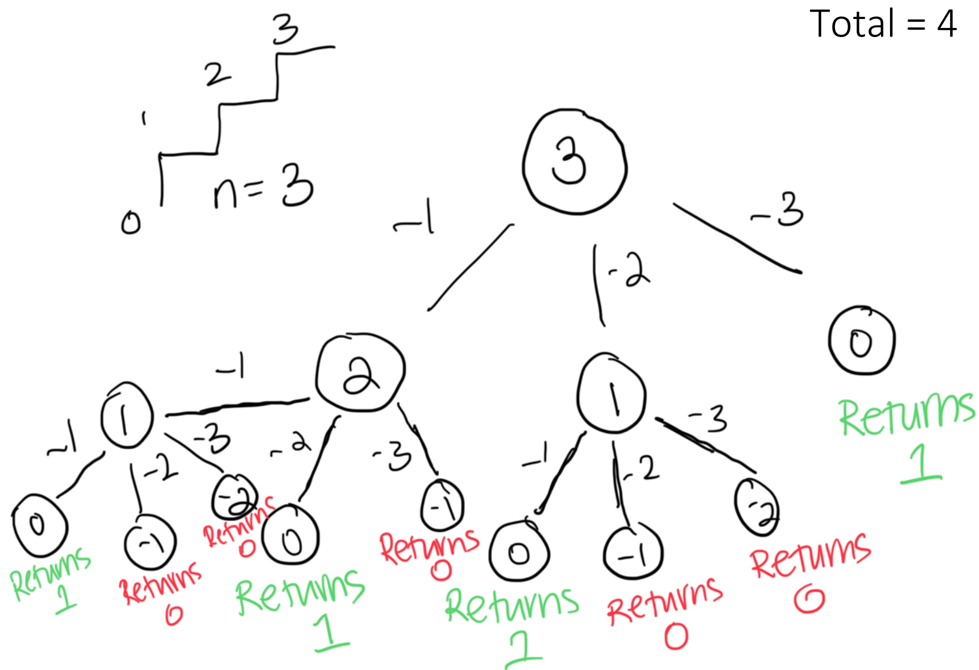
In this case (n=1), you can see there is one valid solution (green return statements).

For n=2, our tree would look like:



This time, we found 2 valid solutions (green return statements).

For n=3, the decision tree would look like:



Corresponding to 4 valid solutions (green return statements).

Name: ________________________

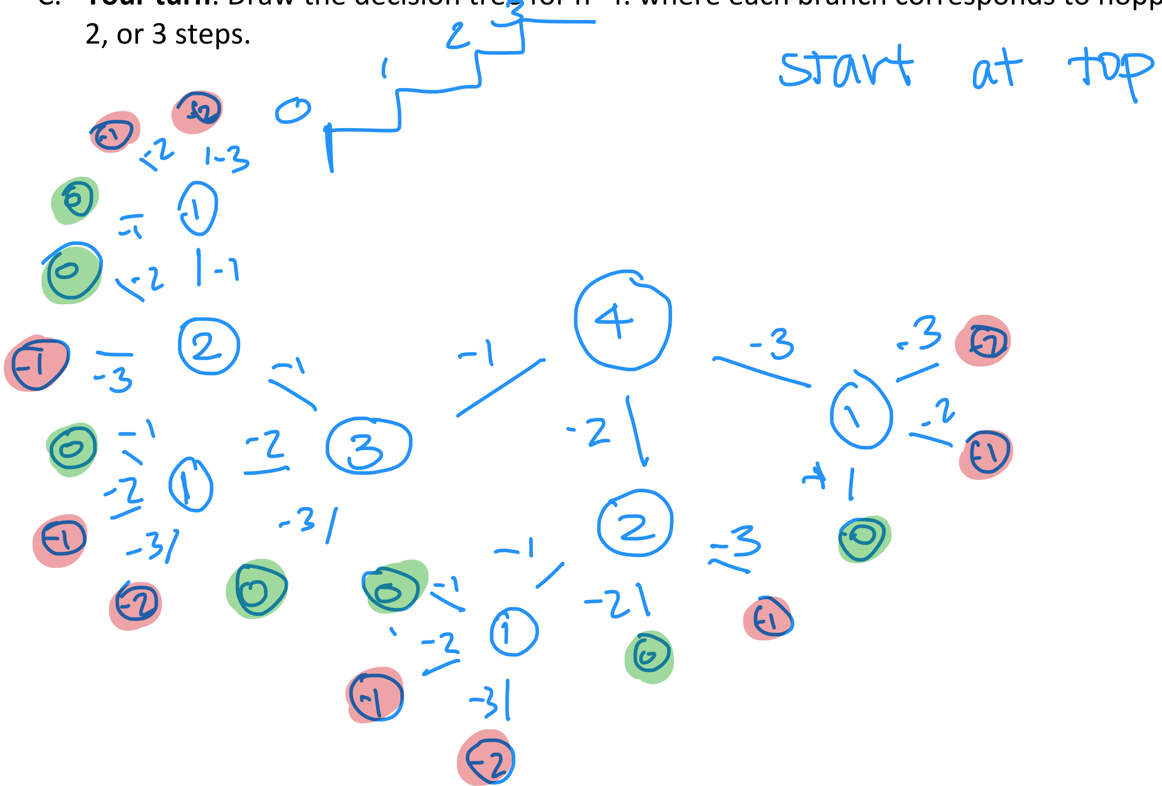
Questions about the diagrams above:

Why does each branch node have three children? 3 options (1, 2 or 3 hops) = 3 recursive calls

What does each leaf represent? Base case reached

Name: ________________________

C. **Your turn**: Draw the decision tree for n=4. where each branch corresponds to hopping 1, 2, or 3 steps.

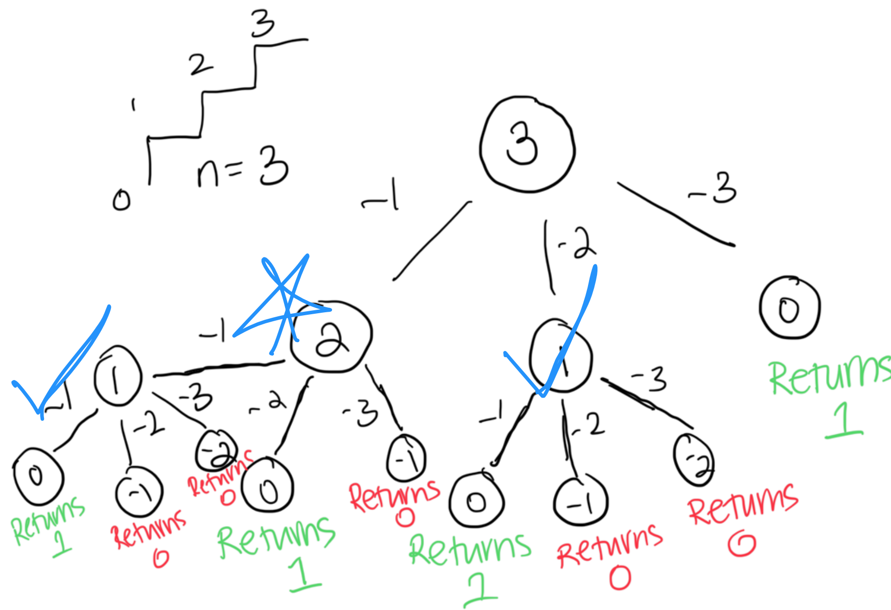


How many valid ways did you find to hop to the top when n=4?

Answer: 7

Name: ________________________

D. Here is the solution for n=3 again. In this diagram, can you find subtrees that look like what we showed for n=2 and n=1?
   - Draw a STAR next to the root note of each subtree identical to the n=2 subproblem.
   - Draw a CHECK MARK next to the root node of each subtree identical to then n=1 subproblem.



E. This semester we briefly discussed **dynamic programming**, where you choose to use more memory to save on the number of operations you have to perform. Dynamic programming is very useful for problems like this one.

   Now that you know many solutions there are for n=1, n=2, n=3, and n=4: 1, 3, 4, and the number you calculated in part (B). What is the best data structure you can think of to store this data for easy lookup based on a given *n*? Indicate **which data structure** you would use to store this information, and then **draw what it would look like** with the data for n = 1...4 stored.

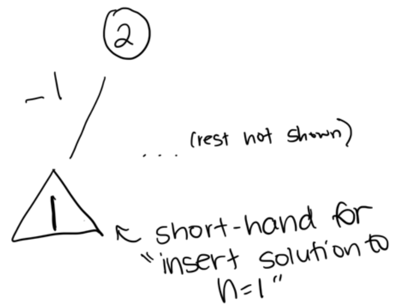
   Data structure selected: Hash map / lookup table

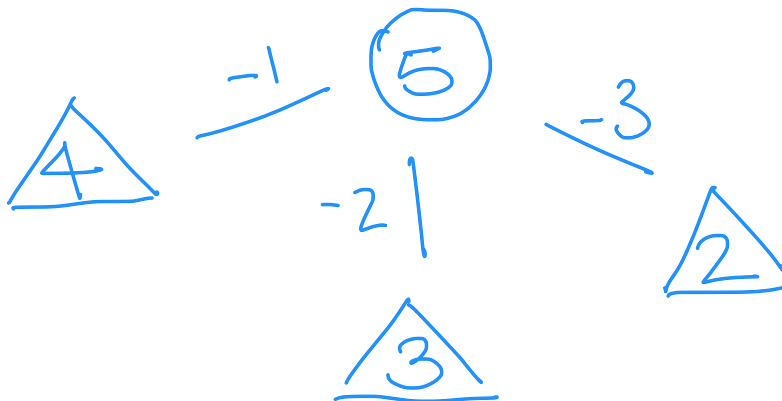
   What it would look like with our data:

N	solutions
0	1
1	1
2	2
3	4
4	7

Name: ______________________

F. Based on your answer to (D), how could you solve n=5? Start by drawing the decision tree, but this time, don't solve any problem that you already found the answer for above. Instead, mark these solved subproblems with a triangle, as shown below.





solution (4) + solution (3) + sol (2)

7 + 4 + 2

G. Based on the diagram you drew above, what is the solution to climb(5)? Put your final answer in the box.

13

Name: ________________________

H. What updates would you make to the code shown in part A (pg 13) to integrate this dynamic programming approach? You don't need to write Java (pseudocode or plan words work just as well), but make sure to indicate a) which lines you would change, and b) what you would do at this line. Your suggestion can be very brief, but just make sure you indicate what, specifically, you would write code to do.

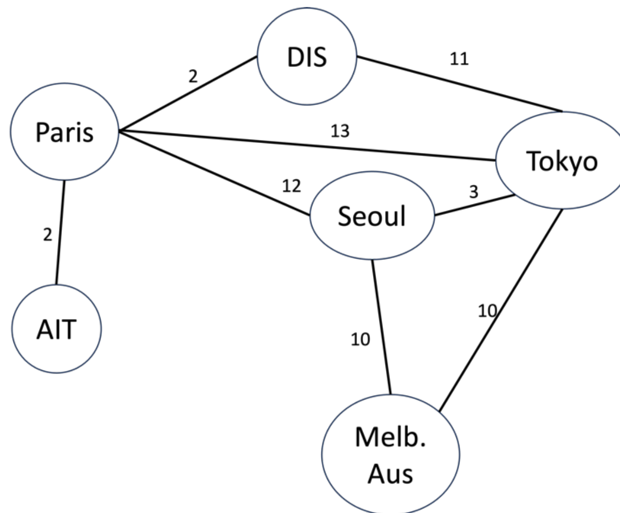
For example (not related to this problem): "At line 242, insert a for loop to iterate over each element of the array, adding one to each element."

Add a step to check if we have a solution to this problem in our table – would go in base case block or even replace existing BC lines 5-11 or line 12

Name: ________________________

PROBLEM 4: Graph Traversal (20 points)

It's fall 2025 and Ellie is studying abroad in Paris (true story). At some point, she wants to visit all of her friends from CSC 210 who are also studying abroad. Below is a graph showing the different cities/programs she'd like to visit, as well as the flight time between locations. (Melb. stands for Melbourne, DIS is in Copenhagen, and AIT is in Budapest).



a.) Ellie decides to see what would happen if she tried to visit everyone in one trip using **depth-first** traversal. What would be her next step in each of these possible itineraries? Fill in the boxes with the next stop. Some boxes have more than one possible answer. Note: Route 1 and Route 2 should show different possible paths.

Depth-First Route 1:

Paris → AIT → DIS → [Tokyo] → Melb. → [Seoul]

Depth-First Route 2:

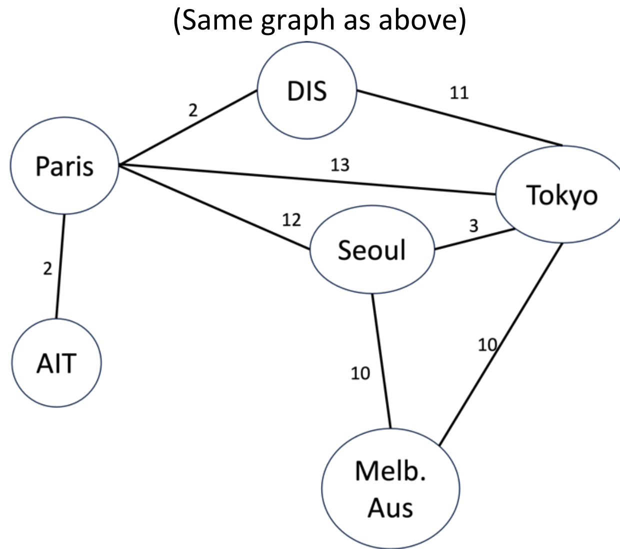
Paris → AIT → DIS → [Tokyo] → [Seoul] → [Melb]

b.) If Ellie starts by visiting Tokyo, what city will she always visit last using depth-first traversal? (You can fill in possible "next location"s if you want, but it's not required).

Paris → Tokyo → [next location] → [next location] → [next location] → [AIT]

Name: ________________________

(Same graph as above)



c.) Now Ellie wants to evaluate her options to visit everyone using **breadth-first** traversal. Just as above, fill in the next city on each possible route. Some boxes have more than one possible answer. 1st step Queue: ~~Paris~~ AIT DIS Seoul Tokyo

Breadth-First Route 1:

Paris → [AIT] → DIS → Tokyo → Seoul → [Melb]

Breadth-First Route 2:

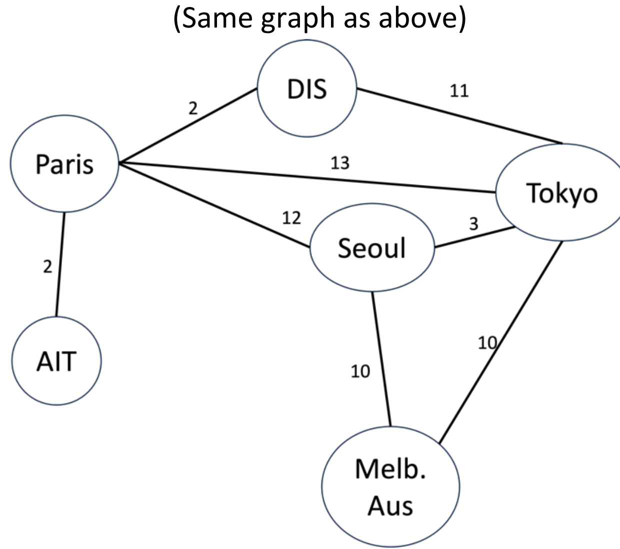
Paris → AIT → DIS → [Seoul] → [Tokyo] → [Melb]

d.) Which stop will always be visited last using Breadth-First Traversal? Circle the name of the node.

AIT DIS (Melbourne) Paris Seoul Tokyo

Name: ________________________

(Same graph as above)



For the remaining questions, use your understanding of Dijkstra's algorithm and the edge weights to circle the route you would recommend Ellie take to minimize her overall flight time. Explain your reasoning below.

e.) Paris to Seoul

Fly to Seoul direct from Paris OR Fly to Seoul via Tokyo

12 13+3=16

Justification: Faster to fly direct

f.) Paris to Tokyo

Fly to Tokyo direct from Paris OR Stop by DIS on the way to Tokyo from Paris

13 2+11

Justification: Her choice! Same overall time less travel vs croissants

g.) **Paris to Melbourne**

Fly to Melbourne via Tokyo OR Fly to Melbourne via Seoul

13+10=23 12+10=22

Justification: Faster to fly via Seoul

END OF QUESTIONS

Name: ________________________

EXTRA PAGE FOR SCRATCH WORK:

Name: ________________________

EXTRA PAGE FOR SCRATCH WORK: