UC Berkeley – Computer Science
CS61B: Data Structures

Final, Fall 2020

This test has 9 questions across 18 pages worth a total of 1600 points and is to be completed in 170 minutes. The exam is closed book, except that you are allowed to use two double sided written cheat sheets (front and back). No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. **Write the statement out below in the blank provided and sign your name. You may do this before the exam begins.**

"I have neither given nor received any assistance in the taking of this exam."

____

____

Signature: ___________________________

#	Points	#	Points
1	180	6	80
2	240	7	160
3	220	8	220
4	140	9	210
5	150	**TOTAL**	1600

Name: ______________________________

SID: _______________________________

GitHub Account #: fa20-s_____

Tips:

- There may be partial credit for incomplete answers. Write as much of the solution as you can but bear in mind that we may deduct points if your answers are much more complicated than necessary.
- There are a lot of problems on this exam. **Work through the ones you are comfortable with first. Do not get overly captivated by interesting design issues or complex corner cases you're not sure about.**
- Not all information provided in a problem may be useful, and **you may not need all lines.**
- Unless otherwise stated, all given code on this exam should compile. All code has been compiled and executed before printing, but in the unlikely event that we do happen to catch any bugs in the exam, we'll announce a fix. **Unless we specifically give you the option, the correct answer is not 'does not compile.'**
- ○ indicates that only one circle should be filled in.
- ☐ indicates that more than one box may be filled in.
- For answers which involve filling in a ○ or ☐, please fill in the shape completely.

Optional. Mark along the line to show your feelings on the spectrum between ☹ and ☺.

Before exam: [☹________________☺].
After exam: [☹________________☺].

2. Basic Sorts. Check the box corresponding to the first swap that happens when the given sorting algorithm is run on the array below. If the algorithm does not perform swaps when sorting, check "no swaps". Note that swapping 1-2 is the same thing as swapping 2-1. As an example, if we run selection sort on the array {5, 4, 3, 2, 1}, the first swap would be 5-1 (or equivalently 1-5). By a swap, we mean the exchange of two elements within the same array:

```
void swap(int[] x, int a, int b) {
        int temp = x[a];
        x[a] = x[b];
        x[b] = temp;
}
```

4	5	1	2	3	7

a) Selection sort **(30 points)**.

○ 4-5 ○ 4-1 ○ 4-2 ○ 4-3 ○ 4-7 ○ 5-1 ○ 5-2 ○ 5-3 ○ 5-7 ○ 1-2
○ 1-3 ○ 1-7 ○ 2-3 ○ 2-7 ○ 3-7 ○ No swaps

b) Insertion sort **(30 points)**.

○ 4-5 ○ 4-1 ○ 4-2 ○ 4-3 ○ 4-7 ○ 5-1 ○ 5-2 ○ 5-3 ○ 5-7 ○ 1-2
○ 1-3 ○ 1-7 ○ 2-3 ○ 2-7 ○ 3-7 ○ No swaps

c) Quick sort **(30 points)**.
What is the first swap made by Quicksort-LTH as described in lecture, i.e., using left most item as pivot and using Tony Hoare partitioning, all with no shuffling.

○ 4-5 ○ 4-1 ○ 4-2 ○ 4-3 ○ 4-7 ○ 5-1 ○ 5-2 ○ 5-3 ○ 5-7 ○ 1-2
○ 1-3 ○ 1-7 ○ 2-3 ○ 2-7 ○ 3-7 ○ No swaps

d) Merge sort **(30 points)**.
What is the first swap made by Merge Sort as described in lecture? If we are splitting an odd length array, assume the left half is larger.

○ 4-5 ○ 4-1 ○ 4-2 ○ 4-3 ○ 4-7 ○ 5-1 ○ 5-2 ○ 5-3 ○ 5-7 ○ 1-2
○ 1-3 ○ 1-7 ○ 2-3 ○ 2-7 ○ 3-7 ○ No swaps

e) Heapsort **(30 points)**.
What is the first swap made by in-place Heapsort as described in lecture?

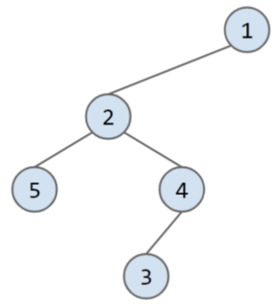
○ 4-5 ○ 4-1 ○ 4-2 ○ 4-3 ○ 4-7 ○ 5-1 ○ 5-2 ○ 5-3 ○ 5-7 ○ 1-2
○ 1-3 ○ 1-7 ○ 2-3 ○ 2-7 ○ 3-7 ○ No swaps

f) Counting sort **(30 points)**.
What is the first swap made by Counting Sort as described in lecture?

○ 4-5 ○ 4-1 ○ 4-2 ○ 4-3 ○ 4-7 ○ 5-1 ○ 5-2 ○ 5-3 ○ 5-7 ○ 1-2
○ 1-3 ○ 1-7 ○ 2-3 ○ 2-7 ○ 3-7 ○ No swaps

3. New Traversals.

a) **Preorder (30 points).** Give the DFS-preorder for the graph below starting at **node 4**. Suppose we **break ties by going to the smaller node first**. Give your answer as a comma separated list, e.g. "1, 2, 3".



DFS-preorder from node 4: ____

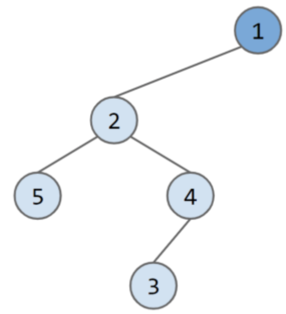
b) **Postorder (40 points).** Now give the DFS-postorder for the graph starting from **node 2**, again breaking ties by going to the smaller node first. Note that we are now starting from node 2!

DFS-postorder from node 2: ____

In class, we discussed pre, in, and post order traversals for binary trees. In this problem, consider a new traversal called the **pre/in order traversal**. This new traversal is given by the pseudocode below, assuming that the action taken upon visitation is to print the vertex's item. Note that in this new traversal, each vertex is visited (e.g., printed) **exactly twice**.

```
pre_in(Node n):
        if n is null, return
        print(n.item)
        pre_in(n.left)
        print(n.item)
        pre_in(n.right)
```

c) **A New Order (50 points).** Give the pre/in order traversal of the tree below, which is the graph above recopied as **a binary tree rooted 1**. Note that this is not supposed to be a binary search tree, it is just a binary tree. Assume we start at 1. Give your answer as a comma separated list.



Pre/in traversal: ____

d) **Tree Inference (60 points).** Now suppose we have the pre/in traversal below starting from the root: 9, 3, 3, 6, 5, 1, 1, 5, 6, 9

Answer the following questions about the binary tree. Recall that the height of a tree is the number of links between the root and the deepest leaf, e.g. the height of the tree from part c is 3. For the last problem, give the level order traversal of the tree as a comma separated list. Note that there is only one tree with the pre/in traversal given above. If an answer does not apply, write "None".

Number of nodes: ____
Height of the tree: ____
The left child of 6: ____
The parent of 1: ____
Level order traversal of the tree: ____

e) **Harder Tree Inference (60 points).** Now suppose we have the pre/in traversal below starting from the root: 2, 2, 2, 2, 2, 2, 3, 3

How many distinct binary trees have this same pre/in traversal? We say that two trees are distinct if they are different in any way. Note that the order of the 2s does not matter.

Number of trees: ____

4. Java Syntax and Data Structure Usage

a) **Prime Factor Iterator (Java Syntax) (80 points).** The `PrimeFactorIterator` class allows you to iterate over the prime factors of a number in increasing order. This problem assumes nothing about your familiarity with prime numbers. Here are some helpful definitions:

- A factor is a number that divides a number exactly, e.g., 15 is a factor of 45.
- A prime is a number whose only factors are 1 and itself.

The code below would print out 3, then 3, then 5, because these are the 3 prime factors of 45. Another way of thinking about this is that 45/3 is 15, and 15/3 is 5, and 5/5 is 1.

```
int x = 45;
/* The loop below iterates 3 times. The first iteration prints 3, then the second
iteration prints 3, then the third iteration prints 5. */
for (int f : new PrimeFactorIterator(x)) {
    System.out.println(f);
}
```

Your implementation of the PrimeFactorIterator should provide the functionality above. Fill in the `PrimeFactorIterator` class below. Note that your code **must fit in the skeleton provided**, i.e., you cannot add any lines. The reference sheet may be helpful. You may assume `x` is a positive integer.

```
public class PrimeFactorIterator implements ________________, ________________ {
      private int x;

      public PrimeFactorIterator(int givenX) {

            ____________________________________________;
      }

      public boolean hasNext() {

            ____________________________________________;
      }

      public int next() {
            int i = 2;

            while (____________________________) {

                  ________________________________________________;
            }

            x = ________________________________________;
            return i;
      }

      public Iterator<Integer> iterator() {

            return ________________________________________;
      }
}
```

b) **Unique Factor Count (Data Structure Selection) (60 points).** The `uniquePrimeFactorCount` of a number is the number of unique prime factors for that number. Examples:

- 98 has a `uniquePrimeFactorCount` of 2, since its prime factors are 2 x 7 x 7. The two unique factors are 2 and 7.
- 223650 has a `uniquePrimeFactorCount` of 5, because its prime factors are 2 x 3 x 3 x 5 x 5 x 7 x 71. The five unique factors are 2, 3, 5, 7, and 71.
- 7 has a `uniquePrimeFactorCount` of 1, since it has just 1 unique prime factor, i.e. itself.

Fill in the `uniquePrimeFactorCount` function below.

You may use PrimeFactorIterator. Even if you did not complete part a correctly, you may assume that the implementation of `PrimeFactorIterator` works correctly. Your implementation **must fit in the skeleton provided**. **You may not add additional lines**. You may assume that any useful Java data structures you'd like to use have been imported. The reference sheet may be helpful.

```
private int uniquePrimeFactorCount(int x) {

      _________________________________________________;

      for (____________________________________________) {


            ____________________________________________;

      }

      return ___________________________________________;
}
```

c) **Most Unique Factors (Data Structure Usage) (80 points).** You are given the `Comparator` below which compares two numbers based on their `uniquePrimeFactorCount`:

```
public class UniqueFactorCountComparator implements Comparator<Integer> {
    public int compare(Integer a, Integer b) {
        return uniquePrimeFactorCount (a) - uniquePrimeFactorCount (b);
    }
}
```

Implement `mostUniquePrimeFactors` below to print the k numbers between 1 and n, inclusive, that have the highest `uniquePrimeFactorCount`. Assume k < n. If multiple numbers have the same `uniquePrimeFactorCount`, you may print any of them. Your code must use O(k) memory and must run in O(n log(k)) time. You may use previous parts. You can add a **maximum of 12 lines** of code. [The reference sheet](#) may be helpful. For simplicity, assume the runtime of `uniquePrimeFactorCount` is constant.

```
public void mostUniquePrimeFactors(int n, int k) {

      Comparator<Integer> c = ____________________________________________;

      MinPQ<Integer> pq = new MinPQ<>(____________________________________);


}
```

Side node to number theory enjoying students (This paragraph is not part of the exam! You do not need to read it): Of course, you can trivially find the number with the most unique prime factors in the given range by just multiplying 2 x 3 x 5 x ... until you get to n. However, you can make this number theory experiment a little more interesting by looking at only a subset of the numbers between 1 and n. For example, when writing this problem, I originally looked at only palindromic numbers between 1 and n. P.S. the first palindromic number with 7 prime factors is 20522502.

5. Sorting Variations. For this problem, we will be sorting Strings and will consider different algorithms to do so. **Assume that all strings are unique.**

Mergesort can be written in pseudocode:

```
- If we have zero or one items, return.
- Split the items into two halves.
- Mergesort the left half.
- Mergesort the right half.
- Use merge to combine the two halves.
```

Quicksort can be written as:

```
- If we have zero or one items, return.
- Select the leftmost item as pivot.
- Partition the array around the pivot. The pivot is now in place.
- Quicksort everything to the left of the pivot.
- Quicksort everything to the right of the pivot.
```

Selection Sort can be written as:

```
- If we have zero or one items, return.
- Find the smallest item.
- Swap the smallest item into the leftmost position.
- Selection Sort the remaining items.
```

Note that above, any time we need to Sort the remaining items, we used the sort itself, e.g. with Quicksort, we Quicksort everything to the left of the pivot, and Quicksort everything to the right of the pivot.

a) **P1sort (40 points).** However, other choices are possible, for example, consider the algorithm below, which I call P1sort. Consider P1Sort below:

```
- If we have zero or one items, return.
- Select the leftmost item as pivot.
- Partition the array around the pivot. The pivot is now in place.
- Selection Sort everything to the left of the pivot.
- Quicksort everything to the right of the pivot.
```

Note that P1Sort does not call itself recursively.

1. What is the best-case runtime for P1sort? Assume that all strings are unique and have constant length.
   ○ $\Theta(1)$ ○ $\Theta(\log(N))$ ○ $\Theta(N)$ ○ $\Theta(N \log(N))$ ○ $\Theta(N^2)$ ○ $\Theta(N^2 \log(N))$
   ○ $\Theta(N^3)$ ○ $\Theta(N^4)$ ○ Worse than $\Theta(N^4)$ ○ Never terminates (infinite loop)

2. What is the worst-case runtime for P1sort? Assume that all strings are unique and have constant length.
   ○ $\Theta(1)$ ○ $\Theta(\log(N))$ ○ $\Theta(N)$ ○ $\Theta(N \log(N))$ ○ $\Theta(N^2)$ ○ $\Theta(N^2 \log(N))$
   ○ $\Theta(N^3)$ ○ $\Theta(N^4)$ ○ Worse than $\Theta(N^4)$ ○ Never terminates (infinite loop)

b) **P2sort (40 points).** Consider P2sort below:

```
- If we have zero or one items, return.
- Split the items into two halves.
- Heapsort the left half.
- Pick a random sorting algorithm from {Selection, Merge, Quick, LSD sort, P2sort}
and use it to sort the right half.
- Use merge to combine the two halves.
```

1. What is the best-case runtime for P2sort? Assume that all strings are unique and have constant length.
   ○ $\Theta(1)$ ○ $\Theta(\log(N))$ ○ $\Theta(N)$ ○ $\Theta(N \log(N))$ ○ $\Theta(N^2)$ ○ $\Theta(N^2 \log(N))$
   ○ $\Theta(N^3)$ ○$\Theta(N^4)$ ○Worse than $\Theta(N^4)$ ○Never terminates (infinite loop)

2. What is the worst-case runtime for P2sort? Assume that all strings are unique and have constant length.
   ○ $\Theta(1)$ ○ $\Theta(\log(N))$ ○ $\Theta(N)$ ○ $\Theta(N \log(N))$ ○ $\Theta(N^2)$ ○ $\Theta(N^2 \log(N))$
   ○ $\Theta(N^3)$ ○$\Theta(N^4)$ ○Worse than $\Theta(N^4)$ ○Never terminates (infinite loop)

c) **P3sort (60 points).** Consider P3sort below:

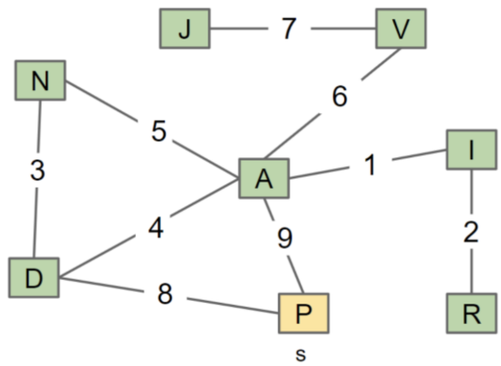
```
- If we have zero or one items, return.
- Split the items into two halves.
- Pick a random sorting algorithm from {Selection, Merge, Quick, LSD sort, P3sort}
and use it to sort the left half.
- Pick a random sorting algorithm from {Selection, Merge, Quick, LSD sort, P3sort}
and use it to sort the right half.
- Use merge to combine the two halves.
```

1. What is the best-case runtime for P3sort? Assume that all strings are unique and have constant length.
   ○ $\Theta(1)$ ○ $\Theta(\log(N))$ ○ $\Theta(N)$ ○ $\Theta(N \log(N))$ ○ $\Theta(N^2)$ ○ $\Theta(N^2 \log(N))$
   ○ $\Theta(N^3)$ ○$\Theta(N^4)$ ○Worse than $\Theta(N^4)$ ○Never terminates (infinite loop)

2. What is the worst-case runtime for P3sort? Assume that all strings are unique and have constant length.
   ○ $\Theta(1)$ ○ $\Theta(\log(N))$ ○ $\Theta(N)$ ○ $\Theta(N \log(N))$ ○ $\Theta(N^2)$ ○ $\Theta(N^2 \log(N))$
   ○ $\Theta(N^3)$ ○$\Theta(N^4)$ ○Worse than $\Theta(N^4)$ ○Never terminates (infinite loop)

3. What is the runtime for P3sort in the case where it happens to pick P3sort for every random choice? Assume that all strings are unique and have constant length.
   ○ $\Theta(1)$ ○ $\Theta(\log(N))$ ○ $\Theta(N)$ ○ $\Theta(N \log(N))$ ○ $\Theta(N^2)$ ○ $\Theta(N^2 \log(N))$
   ○ $\Theta(N^3)$ ○$\Theta(N^4)$ ○Worse than $\Theta(N^4)$ ○Never terminates (infinite loop)

6. Minimum Spanning Trees. Suppose we have the graph below and run Prim's starting from **P**. Break ties alphabetically.



a) **Prim's Nodes (40 points).** What is the order that the nodes are visited? Format your answer as a comma separated list including start vertex P, e.g., "P, A, B, C".

Order: ____

b) **Prim's Edges (30 points).** What edges are included in the MST?

☐ AV ☐ IR ☐ AI ☐ AN ☐ JV ☐ DN ☐ DP ☐ AD ☐ AP

c) **Multiple MSTs (80 points).** For the next part, we will be exploring the idea of having multiple MSTs in a graph. Recall a graph can have multiple MSTs if there are multiple spanning trees of minimum weight.

Looking at the graph below, which has been recopied from above, you might've realized that there is only one MST. For each edge, provide a new **edge weight** which would result in the graph having multiple MSTs **if you changed only that edge**.

If there is no possible value, put "None". If there are multiple possible values, pick any valid value. Note that each answer box is independent of all of the other answer boxes, i.e., if you put a value under AD, that doesn't have any effect on your answer for AP.

AD: ______________________________

AP: ______________________________

DN: ______________________________

AI: ______________________________

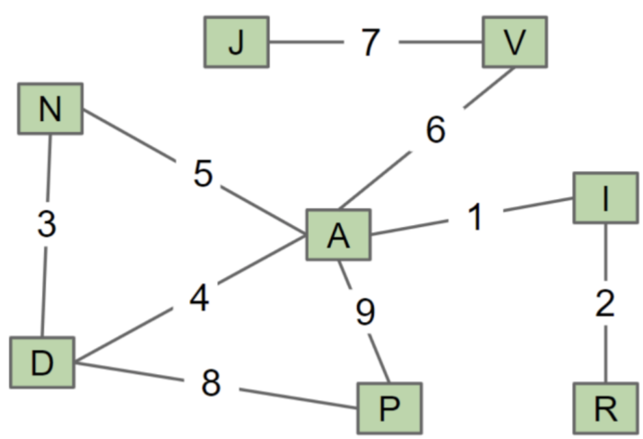
AV: ______________________________

IR: ______________________________

AN: ______________________________

DP: ______________________________

JV: ______________________________



7. Hashing Bears (80 points).

For this problem, assume the `HashMap` works as described in class and is implemented with an array of length 4. You may assume the `HashMap` never resizes. Suppose the Bear class is defined as follows.

```
public class Bear {
    public String name;

    public boolean equals(Object o) {
        Bear bear = (Bear) o;
        return name.equals(bear.name);
    }

    public int hashCode() {
        return name.length();
    }

    ...
}
```

Suppose we run the following code.

```
HashMap<Bear, Integer> map = new HashMap<>();
Bear a = new Bear("sohum");
Bear b = new Bear("arjun");

map.put(a, 1);
map.put(b, 2);

a.name += a.name;
map.put(a, 3);
map.put(b, 4);

b.name += b.name;
map.put(b, 5);
```

Answer the following. Assume the code above has been run. Assume that get returns null if the key is not in the hash table, and that the buckets are zero indexed. Lists of values should be given as a comma separated list of integers, e.g., "3, 5". If a list of values is empty, write "None".

`map.size():` ____

`map.get(new Bear("sohum")):` ____

`map.get(new Bear("sohumsohum")):` ____

Length of list in bucket 1: ____

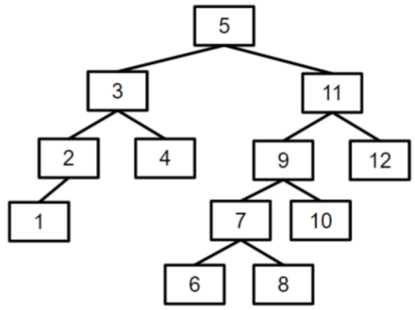
Values in bucket 1 as comma separated list: ____

Length of list in bucket 2: ____

Values in bucket 2 as comma separated list: ____

8. BST Rotations.

a) **Rotation (30 points).** Consider the following BST.



Suppose we call `rotateLeft(p5)`, where p5 is the node containing the 5. Give the following quantities after the rotation is complete. If a node has no parent, write -1.

Root: ______________	3's parent: ____________	5's parent: ____________
9's parent: ____________	11's parent: ____________	12's parent: ____________

b) **Rotation Sequence (50 points).** Considering the same BST, we want the new root of this BST to be the `BSTNode` containing the integer 10. This can be done with 3 rotation operations on the BST. Fill in the following with the type of rotation and the item in the `BSTNode` that is being rotated.

1.	○ `rotateRight` ○ `rotateLeft`	Value to rotate: ___________________
2.	○ `rotateRight` ○ `rotateLeft`	Value to rotate: ___________________
3.	○ `rotateRight` ○ `rotateLeft`	Value to rotate: ___________________

c) **Rotation Code (80 points).** Suppose we have an implementation of a BST class as follows.

```
public class BST {

    private BSTNode root;

    private class BSTNode {
        private int value;
        private BSTNode left;
        private BSTNode right;
    }

    /* Rotates node to the left */
    private void rotateLeft(BSTNode node) {// implementation not shown}

    /* Rotates node to the right */
    private void rotateRight(BSTNode node) {// implementation not shown}
    ...
}
```

Suppose we want to add the method `rotateUp(int value)` that rotates the `BSTNode` containing the given `value` to become the new root of the tree. In part b, you performed `rotateUp(10)`. For simplicity, you may assume the given `value` is in the tree.

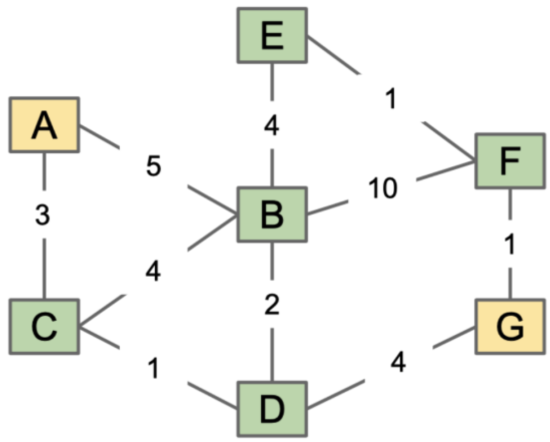
Fill in the implementation for `rotateUp` below. You may only fill in the code provided. You may not add additional lines.

```
public class BST {
      …
      public void rotateUp(int value) {


            ______________________________________________;

      }

      public void rotateUpHelper(int value, BSTNode curr) {


            if (__________________________________________) {

                  ______________________________________________;

            }
            if (__________________________________________) {

                  ______________________________________________;

                  ______________________________________________;

            } else {

                  ______________________________________________;

                  ______________________________________________;

            }
      }
}
```

d) **PNH (0 points).** The patriarch of the Tenenbaum family falsely claimed on his gravestone to have died tragically rescuing the family from what?

Answer: ____

9. Shortest Paths (220 points).
Consider the graph below. We recommend you draw the graph on paper before starting. For all algorithms, suppose we break ties alphabetically.



a) **BFS (30 points).** Suppose we run BFS from **vertex G** on the graph above.
Give the first **four** vertices visited by BFS, including the start **vertex G**. Format your answer as a comma separated list, e.g. "G, X, Y, Z".

Order: ____

b) **Dijsktras (140 points).** Suppose we run Dijsktra's from **vertex A** on the graph above. Note that we do not ask for distTo(A) because it's trivially zero since A is the start.

1. (25 points). Give the distance values computed by Dijkstra's algorithm for each vertex, if we run Dijkstra's starting at **vertex A**.

   `distTo(B):` ____________ `distTo(C):` ____________ `distTo(D):` ____________

   `distTo(E):` ____________ `distTo(F):` ____________ `distTo(G):` ____________

2. (25 points). What is the order that the vertices are visited by Dijkstra's algorithm starting from vertex A? Format your answer as a comma separated list, including vertex A.

   Order: ____

3. (25 points). What edges are included in the shortest paths tree (SPT) for vertex A?

   ☐ AC ☐ AB ☐ BC ☐ BD ☐ BF ☐ CD ☐ DG ☐ EF ☐ FG

4. (65 points). **Subtract an integer k** from **one edge** such that Dijkstra's fails to find the shortest path from **A to G**. For instance, if we wanted to change BE to 1, **k** would be 3. What is the **minimum k**, and what single edge should be changed? If there are multiple correct answers check all that apply (e.g. if subtracting k from AC by itself or subtracting k from AB by itself would result in the wrong shortest path from **A to G**, check the boxes for both AC and AB).

To reemphasize: We're only concerned with the correctness of the shortest path from **A to G**.

☐ AC ☐ AB ☐ BC ☐ BD ☐ BF ☐ CD ☐ DG ☐ EF ☐ FG

k: ________________________________

c) **A* (65 points).** Suppose we run A* from **A** to **G** on the original graph with the heuristic function below. Note that we are trying to find the shortest path from **A to G**, *but* the heuristic of a vertex **v** is the weight of the shortest path from **v to E.**

```
h(v) = total weight of the shortest path from v to E
```

For example, `h(E) = 0, h(F) = 1, h(G) = 2, h(B) = 4`, etc.

1. What will be the priority of C when it is added to the fringe?

   Priority of C: ________________________________

2. In what order will the vertices be visited? Give your answer as a comma separated list:

   A* visit order: ____

3. Will A* compute the correct shortest total distance from **A** to **G** using this heuristic?

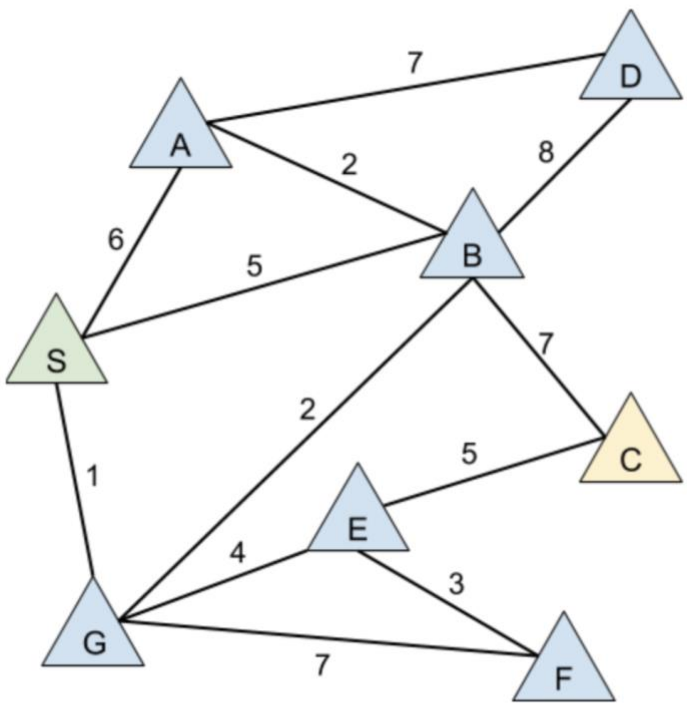
   ◯ Yes ◯ Depends on how ties are broken by the priority queue ◯ No

10. Optimal Peak.

Warning: This problem is particularly challenging. The description is very verbose and somewhat repetitive to make sure you don't miss any details, so it'll take some time to read. For this problem, we are given a graph of `Peaks` (vertices) and `Trails` (edges), where all the `Trails` have positive weights. Sohum wants to plan a hiking **trip** which he starts from a `startPeak`, hikes out to a `goalPeak`, and hikes back to the `startPeak`! Let's define a **valid trip** by the following criteria:

1. Sohum wants his trip to be **balanced** — the length of the path to the `goalPeak` must be the **exact same** as the length of the return trip.
2. Sohum wants his trip to be **efficient** — the path to the `goalPeak` and from the `goalPeak` back must both be **shortest paths**, i.e. there cannot be a shorter path to the `goalPeak`, or from the goalPeak back.
3. Sohum wants to **leave a different way than he came** to the `goalPeak` — the `Trail` Sohum uses to enter the `goalPeak` must **differ** from the Trail Sohum uses to leave the `goalPeak`.

Now, let's define the **longest valid trip** as the **valid trip** of **maximum total distance**, and let's define the `goalPeak` of the longest valid trip as the **`optimalPeak`**.



For instance, in the graph above, if S is the `startPeak`, there are two valid trips:
S → G → **F** → E → G → S and S → G → B → **C** → E → G → S. These two trips have goal peaks F and C, respectively. These are the only valid goal peaks starting from S.

To verify that **F** is a valid goal peak / S → G → **F** → E → G → S is a valid trip:

1. **Balanced**: The length of S → G → **F** is equal to the length of **F** → E → G → S
2. **Efficient**: S → G → **F** is the shortest path from S to **F** and **F** → E → G → S is the shortest path from **F** to S
3. **Different Way**: In this trip, Sohum takes G → **F** to get to **F** and **F** → E to leave **F**

To verify that **C** is a valid goal peak / S → G → B → **C** → E → G → S is a valid trip:

1. **Balanced**: The length of S → G → B → **C** is equal to the length of **C** → E → G → S
2. **Efficient**: S → G → B → **C** is the shortest path from S to **C** and S → G → E → **C** is the shortest path from **C** to S
3. **Different Way**: In this trip, Sohum takes B → **C** to get to **C** and **C** → E to leave **C**

All other peaks are invalid. For example, D is not a valid `goalPeak` despite there existence of a trip that is **balanced** and **different way**, i.e. (S → A → D) & (D → B → S). This is because (S → A → D) is not **efficient**, i.e., a shorter path from S to D exists (S → G → B → D).

Among these two choices of valid goal peaks, **C is the `optimalPeak`** because the trip to C and back is longer than the trip to F and back, i.e., they have total weight 20 and 16 respectively.

a) **Specific Inputs (100 points).** For each `startPeak` below, find the corresponding `optimalPeak` and **select the trails traversed on the trip**. In the event that a trip traverses the same trail multiple times, check it once. For instance, in the example above with S as the `startPeak`, the `optimalPeak` is **C**, and we would select SG, GB, BC, EC, and GE. If there is no valid trip with the given `startPeak`, put the None for both options.

1. (50 points). C is the `startPeak`. Trails Traversed:

   ☐ AB ☐ AD ☐ AS ☐ BC ☐ BD ☐ BG ☐ BS ☐ CE ☐ EF

   ☐ EG ☐ FG ☐ GS ☐ None

   Optimal Peak (note, no credit will be awarded unless your answer to trails traversed is correct or almost correct):

   ○ A ○ B ○ C ○ D ○ E ○ F ○ G ○ S ○ None

2. (50 points). A is the `startPeak`. Trails Traversed:

   ☐ AB ☐ AD ☐ AS ☐ BC ☐ BD ☐ BG ☐ BS ☐ CE ☐ EF

   ☐ EG ☐ FG ☐ GS ☐ None

   Optimal Peak (note, no credit will be awarded unless your answer to trails traversed is correct or almost correct):

   ○ A ○ B ○ C ○ D ○ E ○ F ○ G ○ S ○ None

b) **General Algorithm (110 points).** Give an algorithm to find the `optimalPeak` given a `startPeak` and a graph. Your algorithm should run in O(E log(V)), where E is the number of edges (trails) and V is the number of vertices (Peaks). You will receive **no partial credit** if your solution does not meet the runtime bound. Assume E > V. Note **you do not need to find the path to and from the `optimalPeak`, only the `optimalPeak`**. Please be detailed, precise, and concise in your explanation.

As an example of how you might write an algorithm in a mix of pseudocode and reasoning, here's our attempt at writing Dijkstra's:

```
Given a source vertex s:
    Set distTo(v) = infinity for all vertices except for the source which has
distTo(s) = 0
    Set edgeTo(v) = null for all vertices

    Until all vertices have been visited:
        Let A be the vertex with smallest distTo that has not yet been visited
        visit(A)


define visit(A):
    For each edge e from A to B with weight w:
        if distTo(A) + w < distTo(B):
            distTo(B) = distTo(A) + w
            edgeTo(B) = A
```

To dissuade random guessing, we will provide 10% credit (11 points) for those that select the option below.

◯ I will accept 10% credit and acknowledge my response will not be considered.
◯ I will attempt the problem and acknowledge I will be graded as such.

Give your algorithm here. Note it should NOT be in code. Instead, we recommend using a mix of pseudocode and reasoning.