Final Exam

Student Name

Search students by name or email...

Q1 Are You Better Informed than ChatGPT?

600 Points

I asked ChatGPT a series of questions about our course. In this problem, you'll critique ChatGPT's answers. In each, I've edited ChatGPT's answers slightly in order to make them more concise.

Q1.1 Hash Tables

150 Points

In response to the question "What are some advantages of balanced binary search trees over hash tables?", ChatGPT gave the following list of advantages. Select all that are **completely true**. Note: ChatGPT is referring to hash tables that work as described in lecture, i.e. each bucket is a linked list.

- Balanced binary search trees provide faster search, insertion, and deletion operations than hash tables in the average case, because they are able to take advantage of the ordering of the keys to quickly locate an element.
- [✓ Balanced binary search trees support some operations that are inefficient with hash tables. For example, you can efficiently find the minimum or maximum element in a binary search tree, or find all elements within a given range, whereas these operations are not efficiently supported by hash tables.]
- [✓ Balanced binary search trees are more predictable in terms of performance, because their worst-case time complexity is known and is better than the worst-case time complexity of hash tables.]
- None of the above

Explanation

- Balanced binary search trees do NOT provide faster operations in the average case. On average, balanced binary search trees take logarithmic time, compared to constant time for hash tables.
- The second option is a true statement - range searching and operations involving comparisons more naturally trend towards BSTs over hash tables.
- Lastly, balanced BSTs (like LLRBs) have $O(\log(N))$ time complexity compared to $O(N)$ for hash tables using external chaining (where the worst case scenario is no different than a linked list).

✔ Correct

Save Answer Last saved on **Feb 26 at 9:36 PM**

Q1.2 Insertion Sort and Randomized Quicksort
150 Points

In response to the question "What are some advantages of insertion sort over randomized quicksort?" Again, select all that are **completely true.**

[ ] One advantage is that insertion sort is generally more efficient for small data sets, because the time complexity of insertion sort is O(n) for a data set with n elements, while the time complexity of randomized quicksort is O(n log n) on average for a data set of n elements. This means that for very small data sets, insertion sort may be faster than randomized quicksort.

[✓] Another advantage of insertion sort is that it is a stable sorting algorithm, which means that it preserves the relative order of items with the same value in the sorted data set. This is not the case for randomized quicksort, which is not a stable sorting algorithm.

[ ] Insertion sort is an in-place sorting algorithm, which means that it does not require any additional memory space to sort a data set. This is not the case for randomized quicksort, which uses an auxiliary array to store the pivot element during the partitioning process.

[✓] Insertion sort is a good choice for sorting data sets that are already partially sorted, because it is able to take advantage of the existing order in the data set to sort the data more efficiently. Randomized quicksort does not have this advantage, because it does not use the existing order of the data set to guide the sorting process.

[ ] None of the above

Explanation

- The first statement is *almost* completely true. Although insertion sort is more efficient for small data sets, the time complexity is still $O(n^2)$, not $O(n)$.
- The second statement is true - insertion sort is stable because swaps are between two contiguous elements, so preventing a swap between two equal elements will preserve stability. Hoare partitioning, on the other hand, trades good space complexity for loss in a stable sorting algorithm, as the partitioning scheme may invert the order of equal elements.
- Third, as mentioned earlier, Hoare partitioning is done in place, which does not use any auxiliary array (another partitioning scheme, 3-scan, does conventionally use an auxiliary array).
- The last statement relies on the knowledge that insertion sort can actually be theta bound by $\theta(n + k)$, where k represents the number of inversions necessary to completely sort the array. As a result, if we decrease the number of inversions by inputting already partially sorted data, we can get closer to a $\theta(n)$ bound for insertion sort, as opposed to randomized quicksort, which runs in $n \log(n)$ time on average.

✔ Correct

Save Answer   Last saved on **Feb 26 at 9:23 PM**

Q1.3 Disjoint Sets
150 Points

In response to the question "What are some advantages of the Quick Find implementation of the Disjoint Sets ADT over the Weighted Quick Union implementation?" Again, select all that are

completely true.

- [ ] Quick Find is generally faster than Weighted Quick Union in the average case, because it uses a very simple algorithm that requires only constant time operations. This can be beneficial in applications where speed is important.
- [ ] Quick Find is also more space efficient than Weighted Quick Union, because it uses a flat array to represent the disjoint sets, whereas Weighted Quick Union uses a tree-based data structure. This means that Quick Find can be a better choice in applications where memory is limited.
- [ ] Quick Find does not require any special operations to maintain the disjoint sets. This can make Quick Find a better choice in situations where the data is highly dynamic, or where the disjoint sets are expected to change frequently.
- [x] None of the above

Explanation

- Quick find is fast in determining connectivity of two elements, but every single other operation is much slower in comparison to a WQU implementation.
- Next, both quick find and WQU can use a flat array (although the WQU you may have used in homework 3 visually represents it using a tree-based data structure).
- Third, we *especially* want WQU when the data is dynamic, since that implies that we will be making many connect()/union() calls, which is faster with WQU compared to quick find.

✓ Correct

Save Answer Last saved on **Feb 26 at 9:23 PM**

Q1.4 Selection sort vs. merge sort

150 Points

In response to the question "When should you use selection sort instead of merge sort?" Again, select all that are **completely true.**

Recall that "time complexity" just means the order of growth of the runtime.

- [x] Selection sort has a time complexity of O(n^2) compared to merge sort, which has a time complexity of O(n log n).
- [ ] In general, you should use merge sort over selection sort except in special cases, such as when you are working with very small lists or when you need a stable sort.
- [ ] None of the above.

Explanation

- The first statement is true since n^2 has a larger order of growth compared to $nlogn$. However, it's not a reason to use selection sort over merge sort, so we gave credit.
- The second statement is false since it asserts that selection sort is preferable to merge sort in two cases: small input sizes and when we need a stable sort. However, merge sort is stable, so this does not qualify as a situation where selection sort is preferable to merge sort.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:23 PM**

Q2 Sort Mechanisms
450 Points

Q2.1 Selection Sort
100 Points

Suppose we start with the array [5, 3, 1, 6, 4, 9, 7]. What is the output after the first swap if we use **selection sort**?

- [✓ 1, 3, 5, 6, 4, 9, 7]
- 1, 3, 4, 5, 6, 7, 9
- 1, 5, 3, 6, 4, 9, 7
- 3, 5, 1, 6, 4, 9, 7
- 6, 3, 1, 5, 4, 9, 7
- 9, 3, 1, 6, 4, 5, 7
- 5, 3, 9, 6, 4, 1, 7
- 5, 3, 1, 4, 6, 9, 7

Explanation
The first swap occurs when we scan the entire array to find the minimum element, and swap the element in the 0th index with the index containing the minimum element. Thus, 5 and 1 are swapped.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:23 PM**

Q2.2 Insertion Sort
100 Points

Suppose we start with the array [5, 3, 1, 6, 4, 9, 7]. What is the output after the first swap if we use **insertion sort**?

- 1, 3, 5, 6, 4, 9, 7
- 1, 3, 4, 5, 6, 7, 9
- 1, 5, 3, 6, 4, 9, 7
- [✓ 3, 5, 1, 6, 4, 9, 7]
- 6, 3, 1, 5, 4, 9, 7
- 9, 3, 1, 6, 4, 5, 7
- 5, 3, 9, 6, 4, 1, 7

○ 5, 3, 1, 4, 6, 9, 7

Explanation

For insertion sort, the first swap occurs when we compare 3 with 5 (since 5 cannot move backwards), and since 3 < 5, we swap 3 and 5.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:23 PM**

Q2.3 Quick Sort

100 Points

Suppose we start with the array [5, 3, 1, 6, 4, 9, 7]. What is the output after the first swap if we use **quick sort**, where we use the leftmost item as our pivot, we do not shuffle, and we use Tony Hoare style partitioning (with the L and G pointers).

○ 1, 3, 5, 6, 4, 9, 7
○ 1, 3, 4, 5, 6, 7, 9
○ 1, 5, 3, 6, 4, 9, 7
○ 3, 5, 1, 6, 4, 9, 7
○ 6, 3, 1, 5, 4, 9, 7
○ 9, 3, 1, 6, 4, 5, 7
○ 5, 3, 9, 6, 4, 1, 7
[✓ 5, 3, 1, 4, 6, 9, 7]

Explanation

Since 5 is our pivot, we initialize L to point at 3, and G to point at 7. Since 3 is less than 5, we move L to 1, which is also less than 5, so L moves to 6. Since 6 > 5, we then move to G. Since 7 > 5, we move G to 9, which is also larger than 5, so we move G to 4. Since 4 < 5, we have reached the conditions for L (6) and G (4) to swap.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:24 PM**

Q2.4 Heap Sort

100 Points

Suppose we start with the array [5, 3, 1, 6, 4, 9, 7]. What is the output after the first swap if we use **heap sort**.

○ 1, 3, 5, 6, 4, 9, 7
○ 1, 3, 4, 5, 6, 7, 9
○ 1, 5, 3, 6, 4, 9, 7
○ 3, 5, 1, 6, 4, 9, 7
○ 6, 3, 1, 5, 4, 9, 7

○ 9, 3, 1, 6, 4, 5, 7

[✓ 5, 3, 9, 6, 4, 1, 7]

○ 5, 3, 1, 4, 6, 9, 7

Explanation

Remember that the first step of heap sort is heapification! For this, draw out the array as a complete binary heap (5 at the root, with 3 as the left child and 1 as the right child, and 6 as the left child of 3, 4 as the right child of 3...) Then, we sink down in reverse level order. Since 7, 9, 4, and 6 are all leaf nodes, those cannot sink down. Then, 1 yields to 9 (since we're comparing between 1's children of 9 and 7 and we're working with a max heap in heap sort). This means that 1 and 9 are effectively swapped in the heap.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:24 PM**

Q2.5 Merge Sort

50 Points

For merge sort, I'm not going to ask you about what the array [5, 3, 1, 6, 4, 9, 7] looks like after the first swap. Why? Give the best answer. Assume we're talking about merge sort as described in lecture.

○ Merge sort is not deterministic.

○ The order of the swaps depends on the specific strategic used for merging.

○ Merge sort does not use swap operations on the given array.

[✓ Merge sort does not use swap operations on any array.]

○ None of the above.

Explanation

The crux of merge sort comes from the recursive calls which recursively split the input until hitting the base case, then zippering (merging) both input lists together until we have a fully sorted array. Merge sort performs no swaps at all.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:24 PM**

Q3 Asymptotic Runtime Analysis

850 Points

Q3.1 f1

100 Points

What is the runtime of the function below?

```
public void f1(int N) {
    for (int i = 1; i < N; i = i + 2) {
        for (int j = 1; j < N; j = j * 3) {
            System.out.println("*");
        }
    }
}
```

- ○ $\theta(1)$
- ○ $\theta(\log N)$
- ○ $\theta(N)$
- [✓ $\theta(N \log N)$]
- ○ $\theta(N^2)$
- ○ $\theta(N^2 \log N)$
- ○ $\theta(2^N)$
- ○ $\theta(2^N \log N)$
- ○ $\theta(3^N)$
- ○ $\theta(3^N \log N)$

Explanation

`i` will take on approximately $N/2$ values, and for each value of `i`, `j` will take on approximately $log_3(N)$ values. This means that we will be making a total of $(N/2) * log_3(N)$ print calls, and after dropping constant multiplicative factors, we're left with $\theta(N \log(N))$.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:24 PM**

Q3.2 f2

200 Points

Let C(N) be the number of print statements for the function `f2` below. What is C(N)? You may assume N is a power of 2.

```
public void f2(int N) {
    System.out.println("x");
    int half = N / 2;
    if (N > 1) {
        f2(half);
        f2(half);
        f2(half);
        f2(half);
    }
}
```

- ○ $C(N) = 1$
- ○ $C(N) = 1 + 2 + 3 + 4 + ... + N$
- ○ $C(N) = 1 + 4 + 8 + 16 + ... + N$
- ○ $C(N) = 1 + 4 + 8 + 16 + ... + N^2$

○ $C(N) = 1 + 4 + 8 + 16 + ... + 2^N$

○ $C(N) = 1 + 4 + 8 + 16 + ... + 4^N$

○ $C(N) = 1 + 4 + 16 + 64 + ... + N$

[✓ $C(N) = 1 + 4 + 16 + 64 + ... + N^2$]

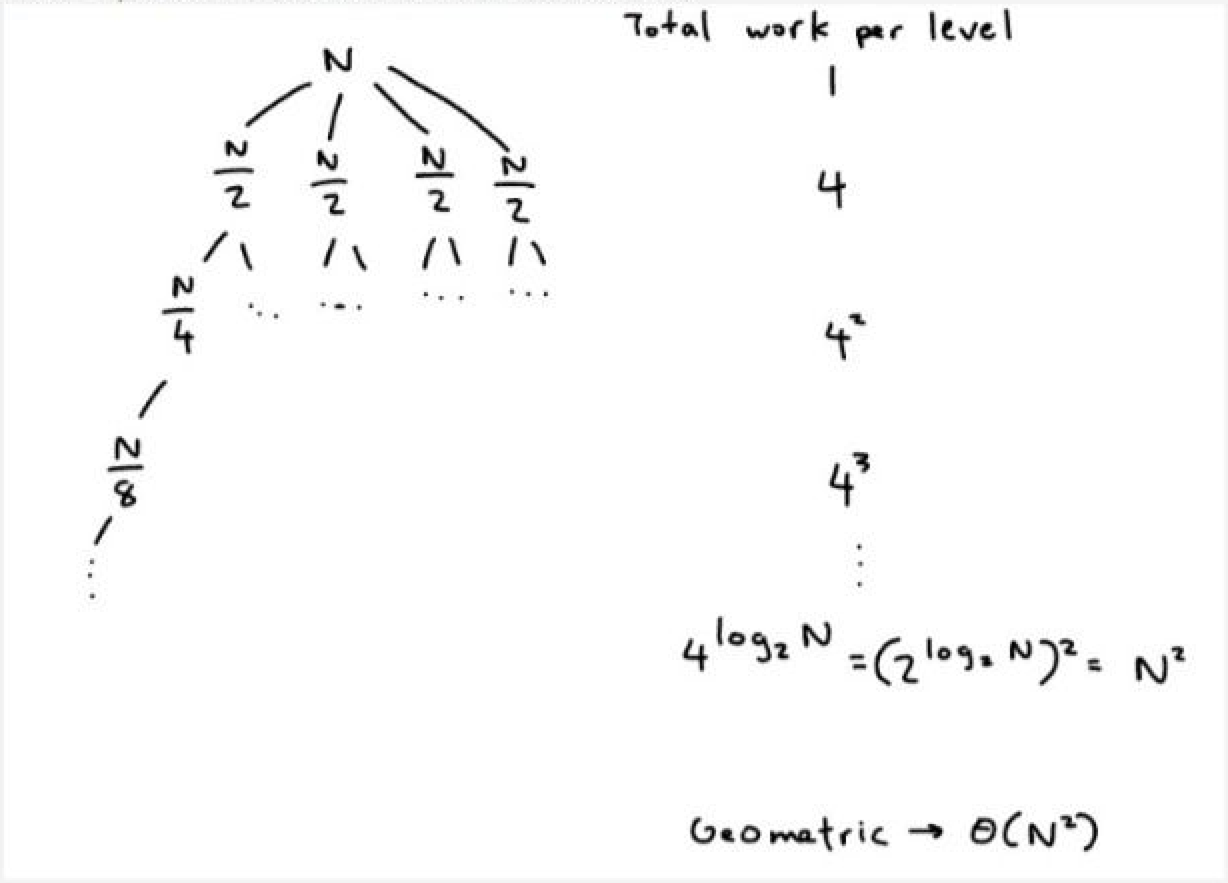
○ $C(N) = 1 + 4 + 16 + 64 + ... + 2^N$

○ $C(N) = 1 + 4 + 16 + 64 + ... + 4^N$

○ $C(N) = 1 + N + N^2 + N^3 + ... + N^N$

Explanation

In the initial call to f2, we invoke 1 print statement. Then, the next recursive call to f2 will invoke 4 print statements (one for each recursive call).



You can view a possible drawing of the recursive tree above (top half of the image). In order to access the last term of the summation, we can recognize that the summation is 4^k, where k is the level we're currently at. To reach the base case, we'll need ~$\log_2 n$ recursive calls, so the work being done at the bottomost level is $4^{log_2(N)}$, which simplifies to N^2 after applying change of base.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:24 PM**

Q3.3 f3

200 Points

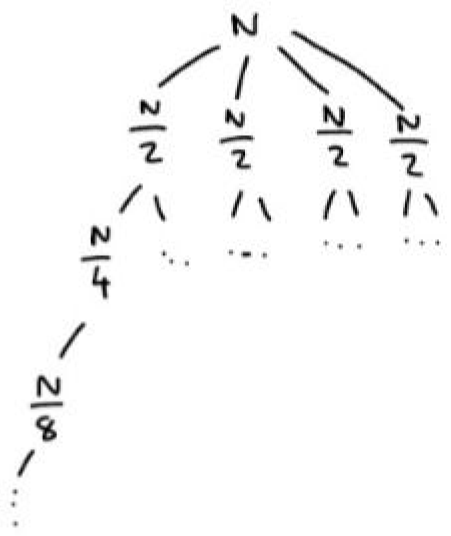
What is the runtime of f3(N)? You may assume N is a power of 2.

```java
public void f3(int N) {
    for (int i = 0; i < N; i += 1) {
        System.out.print("*");
    }
    int half = N / 2;
    if (N > 1) {
        f3(half);
        f3(half);
        f3(half);
        f3(half);
    }
}
```

- $\theta(1)$
- $\theta(\log N)$
- $\theta(N)$
- $\theta(N \log N)$
- [✓ $\theta(N^2)$]
- $\theta(N^2 \log N)$
- $\theta(2^N)$
- $\theta(2^N \log N)$
- $\theta(3^N)$
- $\theta(3^N \log N)$
- $\theta(4^N)$
- $\theta(4^N \log N)$

Explanation

Total work per level



$$N$$

$$4 \cdot \frac{N}{2} = 2N$$

$$16 \cdot \frac{N}{4} = 4N$$

$$4^3 \cdot \frac{N}{2^3} = 8N$$

$$\vdots$$

$$4^{\log_2 N} \cdot \frac{N}{2^{\log_2 N}} = (2^{\log_2 N})^2 \cdot \frac{N}{N} = N^2 \cdot 1 = N^2$$

Geometric $\rightarrow \Theta(N^2)$

You can view a possible drawing of the recursive tree above (bottom half of the image). Since the sum is geometric, we need to find the largest term, which is N^2.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:24 PM**

Q3.4 f4

150 Points

What is the runtime of the function below?

```
public void f4(int N) {
    int[] x = new int[N];
    for (int i = 0; i < N; i += 1) {
        x[i] = i;
    }
    for (int i = 0; i < N - 1; i += 1) {
        swap(x, i, i+1); // swaps x[i] and x[i+1] in constant time
    }
    insertionSort(x);
}
```

- ○ $\theta(1)$
- ○ $\theta(\log N)$
- [✓ $\theta(N)$]
- ○ $\theta(N \log N)$
- ○ $\theta(N^2)$
- ○ $\theta(N^2 \log N)$
- ○ $\theta(2^N)$
- ○ $\theta(2^N \log N)$

Explanation

For this, it may help to draw out a small example (say, `N = 5`). After the initial for loop, we're left with an array that looks like `[0, 1, 2, 3, 4]`. Then, after the next for loop, we're left with `[1, 2, 3, 4, 0]`. We can see here that the practical effect of the first two for loops is that the smallest element is placed at the end of the array. This means that we have a total of N inversions necessary to sort the array. As a result, calling insertion sort on the above array would take $\theta(N + N)$ time, which simplifies to $\theta(N)$. Summing together all the work done in this method call, we have $N + N + N + N$ which all simplifies down to $\theta(N)$. Note that initializing the array would take N time, but we didn't expect you to know that (and was irrelevant to the overall runtime).

✔ Correct

Save Answer Last saved on **Feb 26 at 9:24 PM**

Q3.5 f5

200 Points

Consider the function f5 below.

```
public void f5(int N) {
   for (int x = 2; x < N; x += 1) {
      boolean p = true;
      for (int f = 2; f <= Math.sqrt(x); f += 1) {
         if (x % f == 0) {
            p = false;
            break;
         }
      }
      if (p) {
         System.out.print(x + " ");
      }
   }
}
```

Which of the following can we say about the runtime of the function below? Check all that apply.

$O(1)$

$O(\sqrt{N})$

[✓ $O(N\sqrt{N})$]

[✓ $O(N^2\sqrt{N})$]

[✓ $O(N!)$]

Explanation

Since the answer options are looking for big O notation, we have to try to find valid upper bounds we can place on the runtime of `f5`. To do so, we will attempt to find the paths in this function that yield the longest runtime.

For this, we can see that entering the innermost if conditional will cause us to break and terminate execution of the for loop, so we should probably avoid that if we can. Moving onto the analysis of the code: the outermost for loop will have x take on a total of $\sim N$ values. This immediately rules out the first two answer choices (remember that the break keyword only exits you out of the innermost for loop the keyword is being executed in!) At this point, we actually don't need to do too much more ugly math - since we know that the inner for loop takes up to $\sqrt{x}$ time, and all values of x are upper bounded by N, we can definitively say that f5 is thus upper bounded by N (outer for loop) * $\sqrt{N}$ (inner for loop). This means that the latter two O bounds are also valid, since they have larger orders of growth compared to $O(N\sqrt{N})$.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:24 PM**

Q3.6 What does f5 do?

0 Points

Not for credit, what does f5 do? This problem will not be graded.

Given a number N, prints all the prime numbers less than N.

Explanation

Given a number `N`, prints all the prime numbers less than `N`.

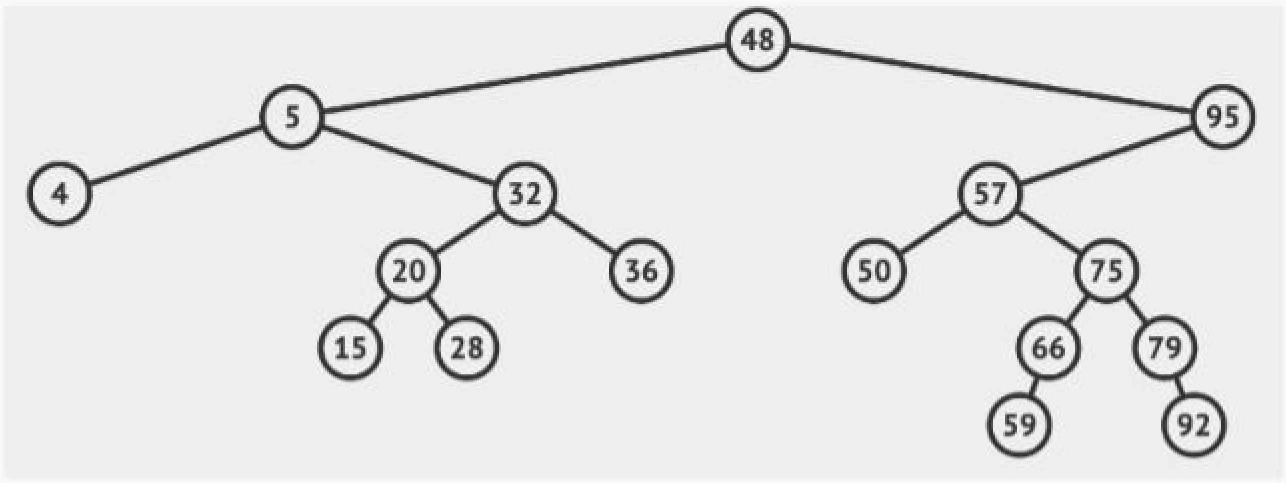
✔ Correct

Save Answer Last saved on **Feb 26 at 9:24 PM**

Q4 BSTs

500 Points

Suppose we have the BST shown below. This is a standard BST, with no special balancing operations.



Q4.1 add(58)

100 Points

If we add 58 to the BST, where will it end up?

- ○ The insertion will fail.
- ○ As the right child of 36.
- ○ As the right child of 50.
- ○ As the right child of 57.
- [✓ ● As the left child of 59.]

Explanation

58 is greater than 48, so we look to the right. It's less than 95, so we go left, greater than 57, so we go right, less than 75, less than 66, and then less than 59, so we add a new node to the left of 59.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:25 PM**

Q4.2 delete(48)
150 Points

If we delete(48) using the standard deletion procedure, also known as Hibbard deletion, of the values below, which could be the new root?

- ○ 5
- ○ 28
- ○ 32
- [✓ 36]
- ○ 57
- ○ 59
- ○ 95

Explanation
In Hibbard deletion, we can either take the inorder predecessor or successor of the element being removed. By this, we mean to run an inorder traversal - the predecessor is the element right before 48, and the successor is the one after. In this case, the inorder predecessor of 48 is 36, and the successor is 50. Since only 36 is offered as an option, we should select that.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:25 PM**

Q4.3 Rotation
150 Points

Suppose we call `rotateLeft(57)`, what happens to the height of the tree?

- ○ It decreases by 1.
- [✓ It is unchanged.]
- ○ It increases by 1.

Explanation
If we call rotateLeft(57), the right child of 57 becomes its parent. Therefore, 75 will take the place of 57, and 57 will become a left child of 75. 79 will continue to be the right child of 75, meaning that 66 must be the right child of 57 in order to preserve the BST property. As a result, the height of the tree is unchanged, with the longest path being `48 --> 95 --> 75 --> 57 --> 66 --> 59`.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:25 PM**

Q4.4 BST Median
100 Points

Suppose we have a BST of 13. This is a standard BST, with no special balancing operations. Which of the following could be the depth of the median?

- [✓ 0 (i.e. the median could be the root)]
- [✓ 1 (i.e. the median could be a child of the root)]
- [✓ 3]
- [✓ 5]
- [✓ 12]

Explanation

In a BST, we can first consider the two cases of a completely bushy tree and a completely spindly tree, and take all values that fall under the range of the two values. In a completely balanced (or bushy) tree, the median element must be the root. In a completely spindly tree, the median element will be halfway down the tree (at depth 6). But we can see there is an even worse case, where the tree zig-zags so that the median is at the very bottom of a height 12 tree. For example if we have a BST and insert `[1, 13, 2, 12, 3, 11, 10, 4, 9, 5, 8, 6, 7]`, the median will end up at depth 12

✓ Correct

Save Answer Last saved on **Feb 26 at 9:25 PM**

Q5 Shortest Paths

1350 Points

Q5.1 Input Differences

100 Points

To run, Dijkstra's needs a source vertex and a graph. It returns a correct shortest paths tree.

What does A* need in order to run? Check all that apply.

- [✓ A graph]
- [✓ A source vertex]
- [✓ A target vertex]
- A list of disallowed vertices
- A shortest paths tree
- [✓ A heuristic]

Explanation

The goal of A* is to find a shortest paths from the start to the goal -- it doesn't need a shortest paths tree to *run*, and it may not generate an SPT containing all vertices. Additionally, none of our graph algorithms require the usage of a list of disallowed vertices.

✓ Correct

Save Answer Last saved on **Feb 26 at 9:25 PM**

Q5.2 A* Output
50 Points

Suppose we try to get driving directions from Chicago to New York City using A*. Which can A* return? Assume nothing about `h(v)`.

- [ ] A correct shortest paths tree to every city from Chicago.
- [ ] A incorrect shortest paths tree to every city from Chicago. That is, every such path is suboptimal.
- [✓] A correct shortest path from Chicago to New York.
- [✓] An incorrect shortest path from Chicago to New York.
- [ ] None of the above

Explanation
This problem was struck due to ambiguities involving the real world.

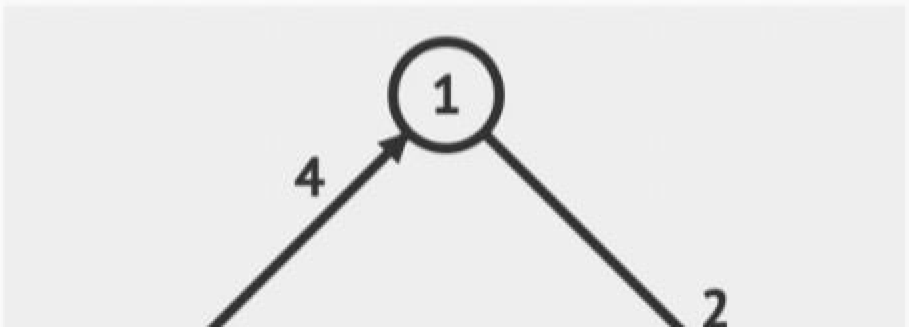
Since we cannot assume anything about `h(v)`, A* could return both a correct and incorrect shortest path from Chicago to New York. If our heuristic is not admissible or consistent, we could terminate A* without finding the true shortest path.

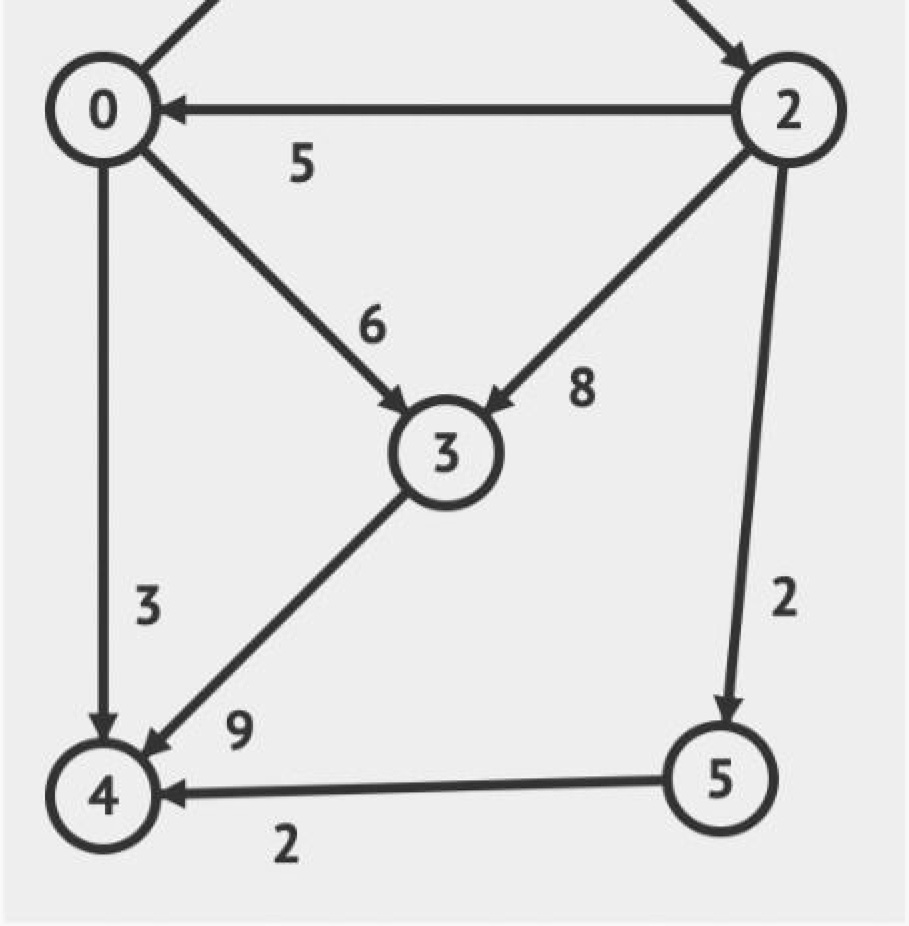
✓ Correct

Save Answer Last saved on **Feb 26 at 9:25 PM**

Q5.3 Dijkstra's Order
200 Points

Suppose we run Dijkstra's on the graph below **starting from vertex 2**. In what order will the vertices be removed from the priority queue?





- 0, 1, 2, 3, 4, 5
- 0, 1, 2, 5, 4, 3
- 0, 4, 1, 3, 5, 2
- 2, 0, 4, 1, 3, 5
- 2, 0, 3, 1, 5, 4
- [✓ 2, 5, 4, 0, 3, 1]

Explanation

Vertices are removed from the priority queue in the order of the shortest distance from the starting vertex. In this case, since we start from vertex 2, 2 is the first to be removed, which will perform a relaxation operation outwards to 0, 3, and 5. The priorities of those vertices are then updated to be 5, 8, and 2 respectively. Since 5 then has the next lowest priority of 5, we will remove that from the priority queue. This allows us to rule out the other answer choices and select the order that begins with "2, 5"

✔ Correct

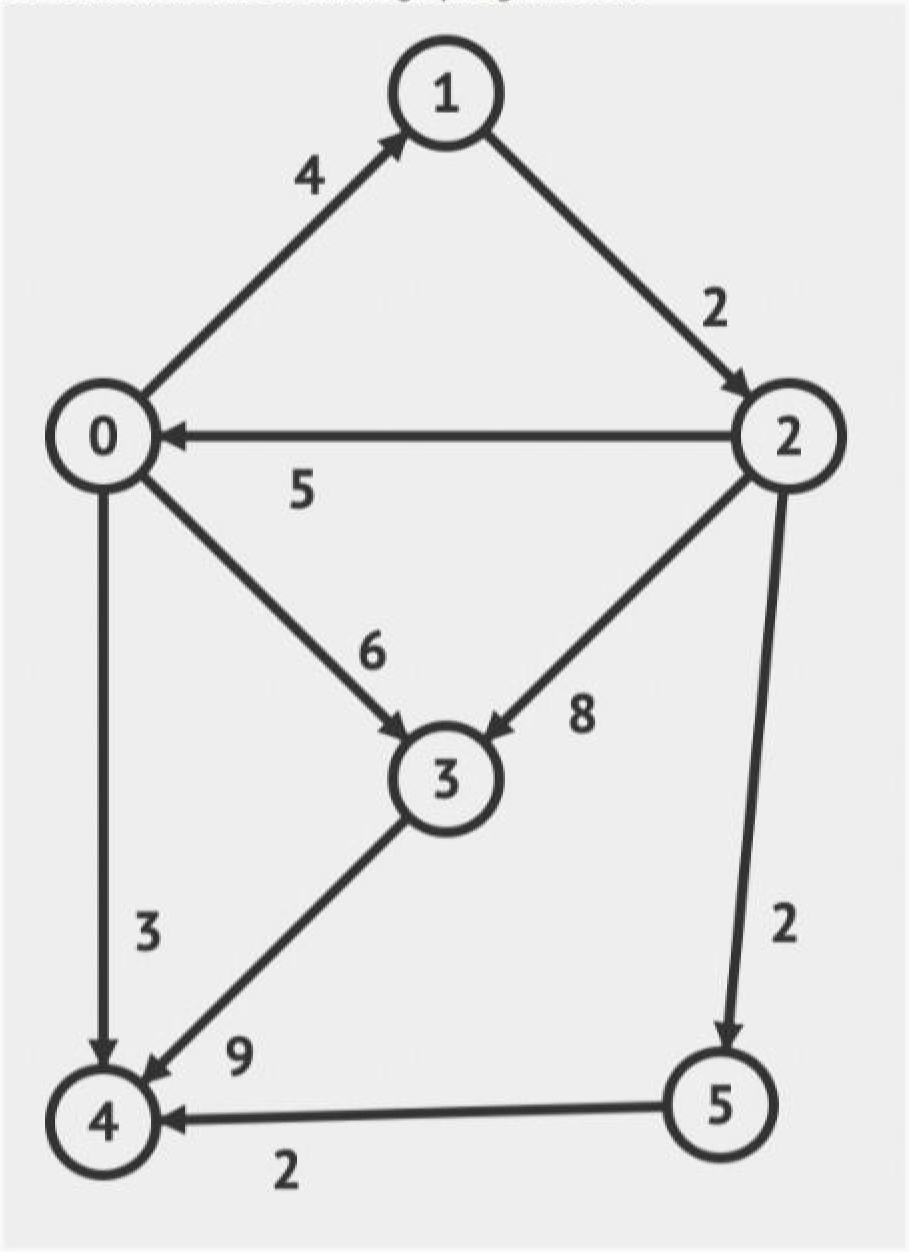
Save Answer Last saved on **Feb 26 at 9:25 PM**

Q5.4 BFS
150 Points

Suppose we run BFS from vertex 2. Which vertices could be visited **last**? Assume vertices are added to the queue in an arbitrary order, i.e. we don't know the order that each neighbor appears in a node's adjacency list.

Recall that after a vertex is visited (i.e. marked), it is never re-added to the fringe.

For convenience, we show the graph again below:



- [ ] 0
- [x] 1
- [ ] 2
- [ ] 3
- [x] 4
- [ ] 5

Explanation

First, remember that BFS is run independent of edge weights. Starting from vertex 2, we know that its neighbors are added in *some* order to the queue, but we don't know which. However, since queues are FIFO (first in first out) data structures, we know that the neighbors of 2 (0, 3, and 5) *must* be removed (e.g. visited) before the other two vertices. Then, if 3 and 5 are added before 0, we are guaranteed that 4 is visited before 1, but if 0 is added before 3 and 5, depending on how nodes are visited, either 1 or 4 could be dequeued next. Thus, either 1 or 4 could be the last visited node when running BFS.

✓ Correct

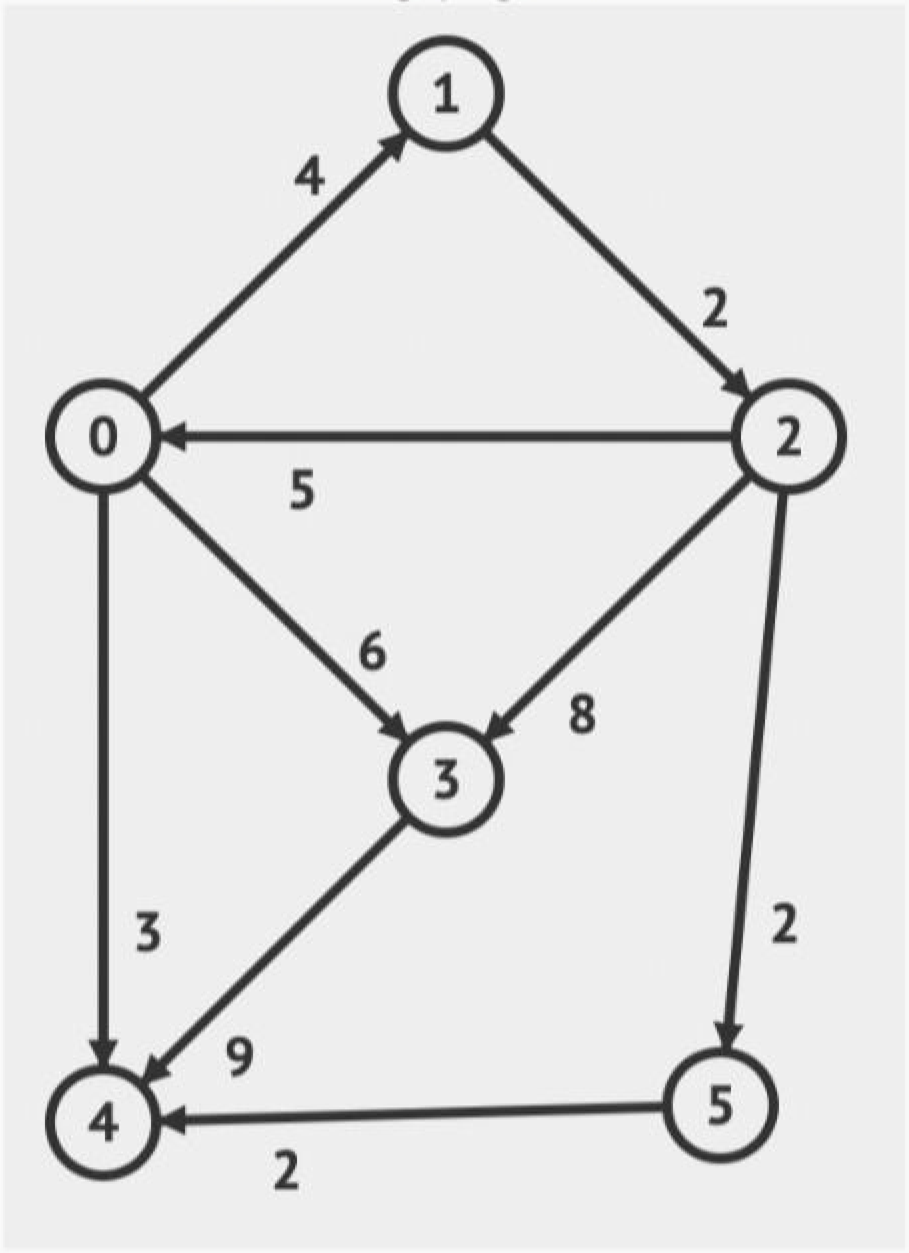
Save Answer Last saved on **Feb 26 at 9:25 PM**

Q5.5 Bizarro Stack BFS
250 Points

Suppose we run BFS, but modified to use a Stack instead of a Queue for storing the fringe. Suppose we start from 2. Which vertices could be visited **last**? Assume vertices are added to the stack in an arbitrary order, i.e. we don't know the order that each neighbor appears in a node's adjacency list.

Recall that after a vertex is visited (i.e. marked), it is never re-added to the fringe.

For convenience, we show the graph again below:



☐ 0

☑ 1

- [x] 1
- [ ] 2
- [x] 3
- [ ] 4
- [x] 5

Explanation

Replacing the fringe with a Stack (a LIFO, or last in first out) data structure in the BFS algorithm is analogous to the DFS algorithm that yields a pre-order traversal. Thus, we know that 2 must be the first node in the traversal. By observation, we can see that 1 only has an incoming edge from 0, and therefore, 0 must come before 1 in any DFS preorder traversal. This rules out 2 and 0. Lastly, we can rule out 4 because all of 2's neighbors have an edge leading to 4, meaning that regardless of the initial neighbor of 2's dequeued from the Stack, 4 will be in that path visited before we explore 2's second neighbor (e.g. if we go 2 --> 0 --> 1, after going back to 0 we can explore 3, which must then visit 4. In all possibilities, 4 goes before one of 2's neighboring vertices.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:25 PM**

Q5.6 relaxRandomlyThenDijkstras
150 Points

Suppose we create a variant of Dijkstra's that works by first relaxing E/2 edges in some random order, then calls Dijkstra's as normal. If an edge is relaxed from a vertex whose current distance is infinite, the relaxation operations have no effect (i.e. distTo and edgeTo values are unchanged, and no vertices in the PQ have their priority changed).

Which of the following can we say about this new algorithm, which we'll call `relaxRandomlyThenDijkstras`.

- [●] `relaxRandomlyThenDijkstras` always correctly computes the shortest paths tree (SPT) from s.
- ○ `relaxRandomlyThenDijkstras` correctly computes the shortest paths tree from s so long as none of the randomly relaxed edges are part of the SPT.
- ○ `relaxRandomlyThenDijkstras` correctly computes the shortest paths tree from s so long as all of the randomly relaxed edges are part of the SPT.
- ○ `relaxRandomlyThenDijkstras` never correctly computes the shortest paths tree (SPT) from s.

Explanation

Since this random relaxation only works from vertices without a current distance of infinite, this is analogous to relaxing $E/2$ times *beginning from the starting vertex*. Remember that relaxation operations simply update the `distTo`, `edgeTo`, and PQ values, but doesn't actually remove from the priority queue. Therefore, we can translate this random relaxation to exploring different paths from the starting vertex and yielding possible paths from there. This does not impact the order that vertices are removed from the PQ, since we remove vertices from the PQ in order of distance from the starting vertex.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:26 PM**

Q5.7
150 Points

Suppose we invert the weight of every edge on a graph and then run Dijkstra's from the source. Which of the following is always true?

- The output of Dijkstra's gives the longest simple path to every vertex. A simple path is a path without cycles.
- The output of Dijkstra's is still the same, i.e. it is the shortest paths tree.
- [✓ Neither of the above]

Explanation

- **For the first statement, consider a triangular graph:**

```
A ---(1)---- B
 \          /
 (1.5)    (1)
    \     /
     \   /
       C
```

A--B--C--A where AB = 1, BC = 1, and AC = 1.5. Negating the edge weights would still return AC as the shortest path, which is consistent with the original shortest path from A to C.

- **For the second statement, consider a triangular graph A--B--C--A, where AB = 1, BC = 1, and AC = 3. Negating the edge weights running Dijkstra's from A would yield AC as the shortest path from A to C, which is not true in the original graph.**

✔ Correct

Save Answer Last saved on **Feb 26 at 9:26 PM**

Q5.8
150 Points

Suppose we have a graph G, where all vertices are reachable from the source. If we double all of the edge weights, which of the following is true about the shortest paths tree?

- [✓ The SPT is always the same set of edges after doubling.]
- The SPT is sometimes the same set of edges after doubling.
- The SPT is never the same set of edges after doubling.

Explanation

Consider a triangular graph with three vertices A, B, C:

```
A ---------- B
 \          /
  \        /
   \      /
    \    /
      C
```

If the SPT states that $AB + BC < AC$ (implying that the shortest path from A to C goes through B instead of directly from A to C), the inequality still holds if we double all edge weights $2(AB) + 2(BC) < 2(AC) \rightarrow 2(AB + BC) < 2(AC) \rightarrow AB + BC < AC$. In other words, the relative ordering of edge weights and paths are unchanged, meaning that the SPT is still the same set of edges.

✓ Correct

Save Answer   Last saved on **Feb 26 at 9:26 PM**

Q5.9
150 Points

Suppose we have a graph G where all edge weights are 1. Which of the following are true?

- [✓ The output of Dijkstra's on G is always the same shortest paths tree as we get if we run BFS on G.]
- The output of Dijkstra's on G is sometimes the same shortest paths tree as we get if we run BFS on G.
- The output of Dijkstra's on G is never the same shortest paths tree as we get if we run BFS on G.

Explanation
If we specify that the starting vertex is the same and ties are broken in the same manner, the output of Dijkstra's will be the same as BFS, since BFS returns the SPT considering number of edges traversed (rather than considering edge weights). In this case, since all the edge weights are 1, the SPT considering edge weights is the same as the SPT considering number of edges traversed. Thus, the intended answer is always. However, without this specification, the answer is sometimes.

✓ Correct

Save Answer   Last saved on **Feb 26 at 9:26 PM**

Q5.10 Ocean Navigation
0 Points

During the Age of Sail (say, roughly the year 1600), navigators in the Northern Hemisphere had a somewhat easier time than navigators in the Southern Hemisphere. Why?

Polaris, the North Star, always appears in the direction of the north pole and provides a way to determine direction at sea. however, it is only visible in the Northern Hemisphere, and not the Southern Hemisphere

Explanation

Polaris, the North Star, always appears in the direction of the north pole and provides a way to determine direction at sea. however, it is only visible in the Northern Hemisphere, and not the Southern Hemisphere.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:26 PM**

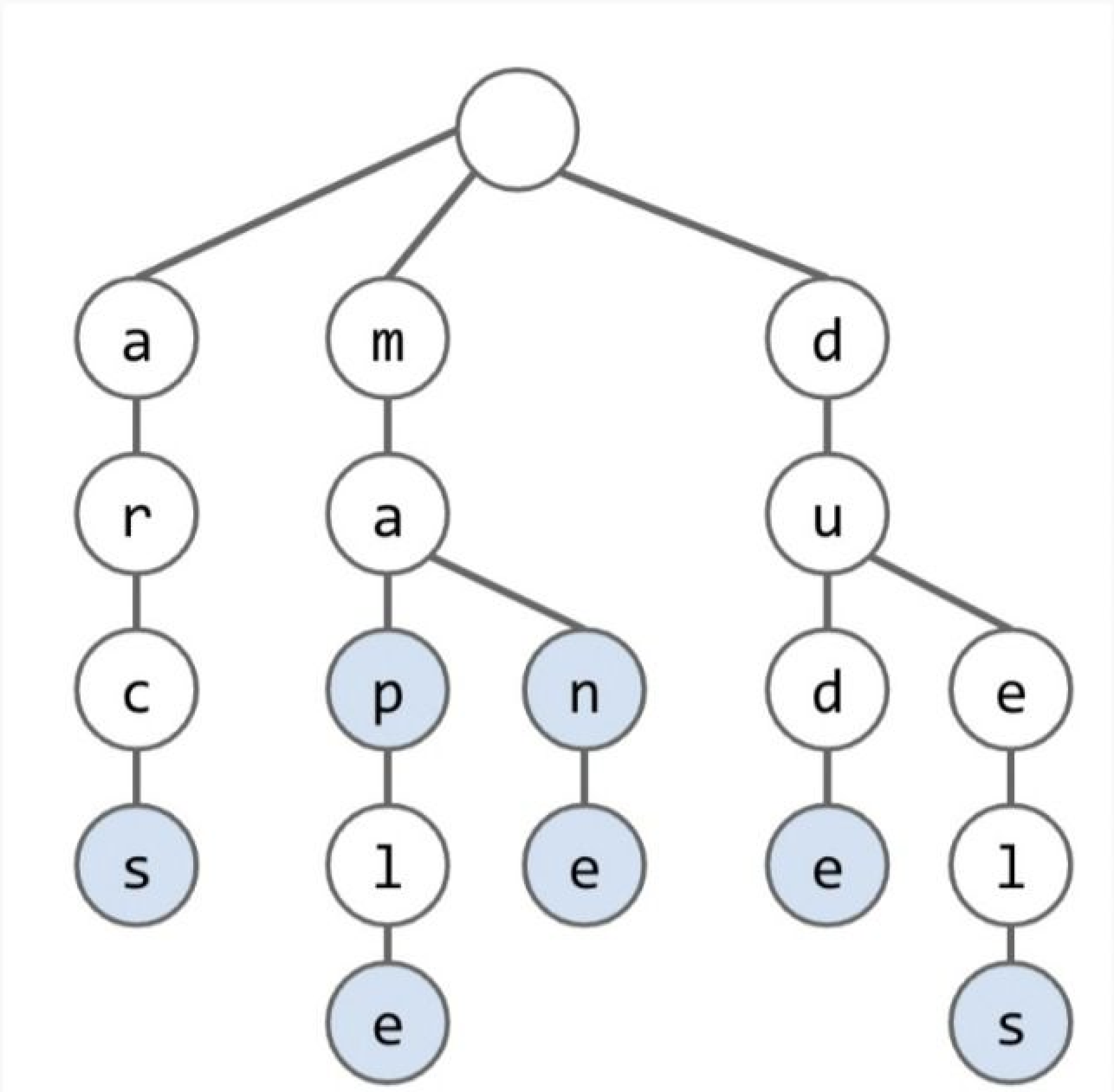
Q6 Tries

400 Points

Q6.1 keysWithPrefix

100 Points

Suppose we have the trie shown below. Recall that `isKey` is `true` for blue nodes and `isKey` is `false` for white nodes.



If we call `keysWithPrefix("ma")`, which keys will be returned. Recall that `keysWithPrefix` finds all keys with the given prefix.

- p
- e
- n
- ma
- [✓ map]
- mapl
- [✓ maple]
- [✓ man]
- [✓ mane]

Explanation

Given the input `"ma"`, we would start from the root and traverse down `m` and `a`, then yield all strings that end in a blue node.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:26 PM**

Q6.2
100 Points

What key, when added, creates the **fewest** new nodes? If multiple keys produce the minimum, select all that apply.

- [✓ arck]
- [✓ at]
- of
- [✓ maples]
- marg
- marge

Explanation

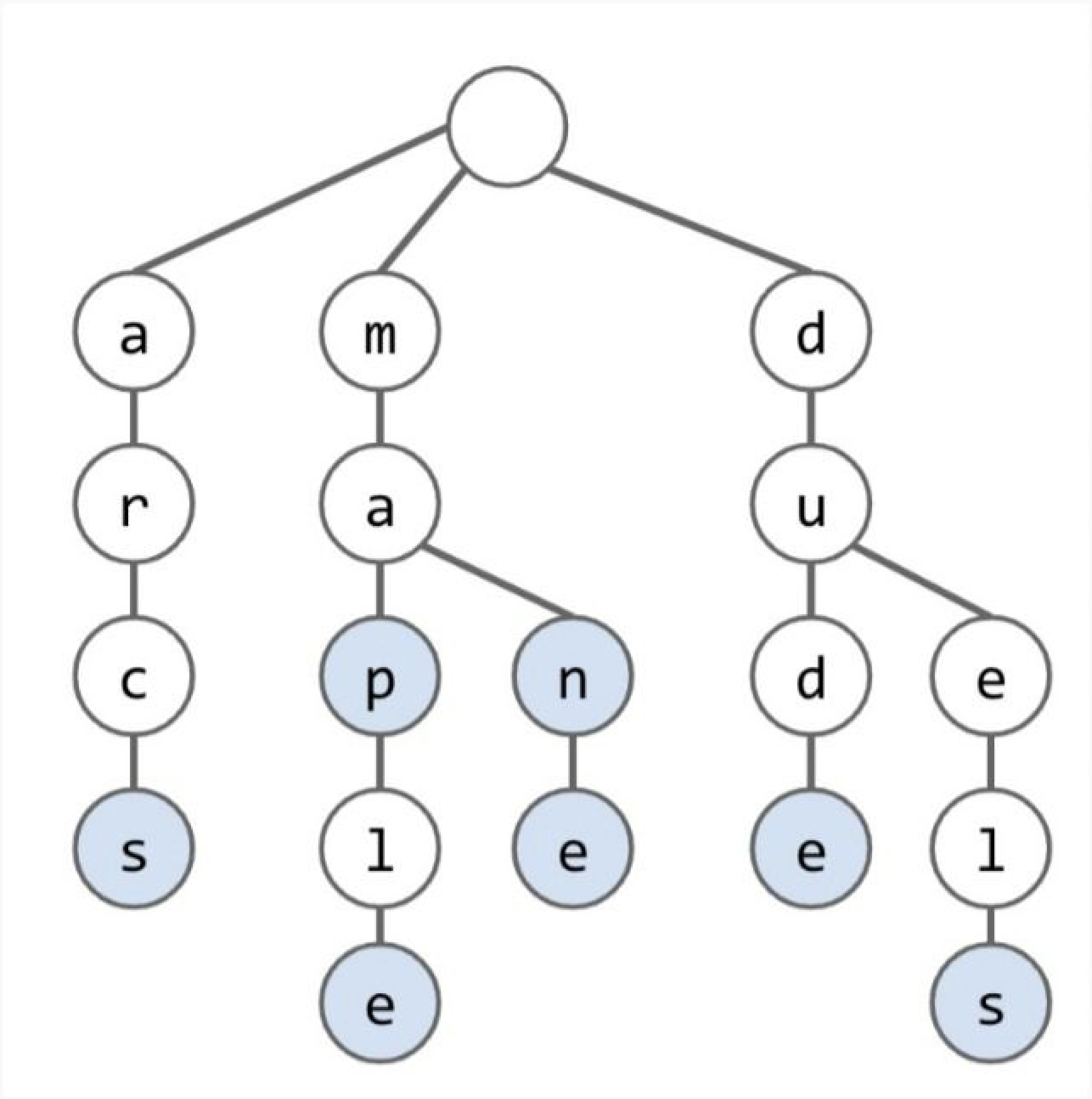
As an example, `"arck"` would create one new node branching off of the `c` node on the left branch, since `a --> r --> c` already exists in the trie. As a reminder, in our trie data structure, we only initially have access to the root node, so we cannot start traversal in the middle of the data structure. `"at"` and `"maples"` would only add 1 additional node.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:26 PM**

Q6.3
200 Points

Suppose we add the function `shortestPrefixOf` that accepts a `String s` and returns the shortest key in the Trie that is a prefix of `s`. Calling `shortestPrefixOf("maneuver")` on the Trie would return `"man"`. A string is a prefix of itself, e.g. calling `shortestPrefixOf("man")` on the Trie would return `"man"`. Assume `shortestPrefixOf` has been implemented correctly and as efficiently as possible. By efficient, we mean that it traverses no more nodes than necessary. What `String s` below would **maximize** the number of nodes needed to be traversed? If multiple words produce the maximum, select all that apply. In other words, what are the worst case inputs for this operation?



- ☐ arc
- ☐ arcs
- ☐ dude
- ☐ duel
- [✓ duels]
- [✓ duelsport]
- ☐ mane
- ☐ mapl
- ☐ maple
- ☐ maplesyrup

Explanation

Both `"duels"` and `"duelsport"` would traverse 5 nodes until the String `"duels"` is found. Although `"maple"` and `"maplesyrup"` contain a string present in the trie that's 5 letters long, since the string `"map"` exists in the trie, we would terminate the algorithm after just traversing over 3 nodes.

✓ Correct

Save Answer Last saved on **Feb 26 at 9:26 PM**

Q7 Mash Ups

700 Points

Q7.1 Hash Heap

150 Points

Suppose we implement a set as a hash table where each bucket is stored as a heap oriented array.

Suppose we insert N items into such a hash table. What is the worst case asymptotic runtime of a single `contains` operation? Do not assume anything about the distribution of items in the hash table.

- ○ $\Theta(1)$
- ○ $\Theta(\log N)$
- [✓ $\Theta(N)$]
- ○ $\Theta(N \log N)$
- ○ $\Theta(N^2)$
- ○ $\Theta(N^2 \log N)$

Explanation

Seeing if an element exists in a heap takes $\Theta(N)$ time since we'd have to search the entire heap.

✓ Correct

Save Answer Last saved on **Feb 26 at 9:27 PM**

Q7.2
150 Points

Suppose we implement a set as a hash table where each bucket is stored as a 2-3 tree.

Suppose we insert N items into such a hash table. What is the worst case asymptotic runtime of a single `contains` operation? Do not assume anything about the distribution of items in the hash table.

- ○ $\Theta(1)$
- [✓ $\Theta(\log N)$]
- ○ $\Theta(N)$
- ○ $\Theta(NlogN)$
- ○ $\Theta(N^2)$
- ○ $\Theta(N^2 \log N)$

Explanation

Seeing if an element exists in a 2-3 tree takes $\Theta(\log N)$ time, since there are only $\log N$ layers, and at each layer we're only making a constant number of comparisons (since each node can only hold at most 2 elements).

✓ Correct

Save Answer Last saved on **Feb 26 at 9:27 PM**

Q7.3 Wheels Within Wheels
125 Points

Suppose we implement a set as a hash table where each bucket is stored as a hash table (whoa!). Let's call each such bucket an "inner hash table". Assume that each inner hash table stores its buckets as a linked list.

Suppose we have M "inner hash tables", i.e. the outer hash table has M buckets. Suppose each inner hash table has P buckets. Thus there are a total of MP linked lists.

For example, if the outer hash table has M = 8 buckets, and the inner hash tables have P = 10 buckets each, then we have a total of 80 linked lists.

Assume both levels of hash table use the same `hashCode()`. **Assume we have N items which are**

evenly distributed by the given `hashCode`.

If $M = P$, what is the approximate length of the average list? Note that P does not appear as a variable in the answers below because $M = P$.

- 1
- N
- [✓ N/M]
- N/M^2
- N^2
- N^2/M
- N^2/M^2

Explanation

The key to this problem is that both levels use the same `.hashCode`, and that $M = P$. This means that in each inner hash table, only one linked list bucket will be used.

The "average list" is unfortunately ambiguous. If we assume that all lists exist, then there are N elements distributed over M^2 lists. If we instead assume that a list (inner bucket) doesn't exist until we try to use it, then only M lists exist. Both N/M and N/M^2 were accepted for credit.

✓ Correct

Save Answer Last saved on **Feb 26 at 9:27 PM**

Q7.4 Wheels Within Wheels 2

125 Points

For the same setup as in problem 7.3, again supposing that $M = P$, roughly how many buckets do we expect to have at least one item? Note that P does not appear as a variable below because $M = P$.

- 0
- N
- N^2
- N/M
- N/M^2
- N^2/M
- N^2/M^2
- [✓ M]
- M^2

Explanation

Regardless of whether we are referring to inner or outer buckets, we will have at least one item in M. For outer buckets, we expect all M will be used. For inner buckets, only one inner bucket will be used for each outer bucket, and again M.

✓ Correct

Save Answer Last saved on **Feb 26 at 9:27 PM**

Q7.5 RLBST
150 Points

Suppose we create a right-leaning binary search tree (RLBST), i.e. 2-3 nodes are represented by a right leaning glue node.

Suppose we insert the number 5 then 3 into an RLBST, what operation should we call to maintain our invariant?

- ○ rotateLeft(3)
- ○ rotateRight(3)
- ○ rotateLeft(5)
- [✓ rotateRight(5)]
- ○ colorFlip(3)
- ○ colorFlip(5)
- ○ No operation necessary

Explanation
This would result in a "left leaning violation", and to ensure that all red links are right leaning, we should call rotateRight on 5, so that 5's left child (3) should become the parent, with 5 as the red right child.

✓ Correct

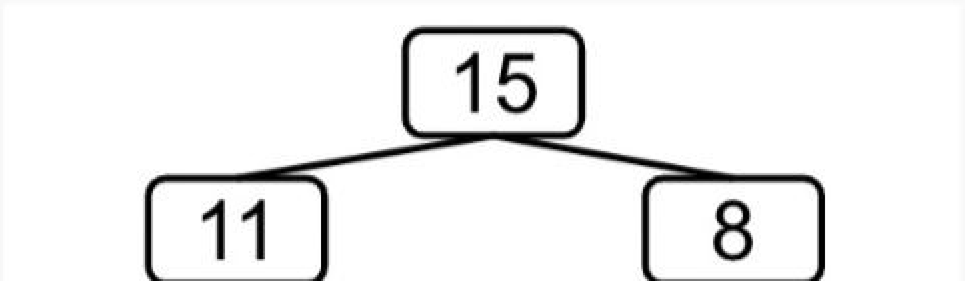
Save Answer Last saved on **Feb 26 at 9:27 PM**

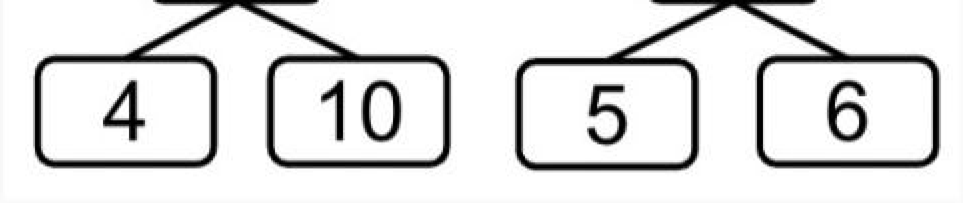
Q8 Heap Verification
700 Points

In class we talked about many possible representations of a tree.

Suppose we use a new representation of a complete tree which we'll call a `TwoDArrayHeap`.

As an example, consider the heap below.





The `TwoDArrayHeap` representation of the heap above is `[[15], [11, 8], [4, 10, 5, 6]]`.

Suppose we want to write a method to verify that a given array is a valid `TwoDArrayHeap`. **For simplicity, assume the number of items is $2^k - 1$ for some k, i.e. it is one less than a power of 2.**

A private helper method is given below. You will fill in the blanks, and assume the ... are appropriate code that handles the right side of each node.

```
private boolean isValidHeap(int[][] heap, int r, int c) {
   if (____________) { return true; }
   int left = ________;
   int right = ....;
   if (heap[r][c] < heap[___][____]) { return false; }
   if (heap[r][c] < ...) { return false; }
   return ________________ && ...;
}
```

To tell if the entire tree is a given array is a valid `isValidHeap`, we fill in the function below:

```
private boolean isValidHeap(int[][] heap) {
   return isValidHeap(heap, ____, ____);
}
```

Q8.1 Call to private helper method

100 Points

What call should we make to `isValidHeap(int[][] heap)` to tell that the entire array is a valid `isValidHeap`? Assume the helper method is working correctly.

- [✓ `isValidHeap(0, 0);`]
- `isValidHeap(0, heap.length - 1);`
- `isValidHeap(heap.length - 1, 0);`
- `isValidHeap(heap.length - 1, heap.length - 1);`

Explanation

To check for validity, we should likely start with the root element and work our way downwards. In the 2-D array representation, the root element is located in the heap[0], so we should pass these parameters into the method call.

✔ Correct

Save Answer   Last saved on **Feb 26 at 9:27 PM**

Q8.2 First blank in helper method

150 Points

```
private boolean isValidHeap(int[][] heap, int r, int c) {
   if (____________) { return true; }
   int left = ________;
   int right = ....;
   if (heap[r][c] < heap[___][____]) { return false; }
   if (heap[r][c] < ...) { return false; }
   return _________________ && ...;
}
```

What should go in the first blank in the helper method:

- ○ `if (heap[r][c] > 0) { return true; }`
- ○ `if (r >= 0) { return true; }`
- ○ `if (r >= Math.log2(heap.length)) { return true; }`
- [✓ `if (r >= heap.length - 1) { return true; }`]

Explanation

If the `r` parameter (or the height of the heap) exceeds the height of the heap, we should conclude that our checks have finished and return true. We notice that there is no iteration here, so the solution is likely looking for recursion (which terminates once we pass the bottommost layer).

✔ Correct

Save Answer Last saved on **Feb 26 at 9:27 PM**

Q8.3 left assignment

150 Points

```
private boolean isValidHeap(int[][] heap, int r, int c) {
   if (____________) { return true; }
   int left = ________;
   int right = ....;
   if (heap[r][c] < heap[___][____]) { return false; }
   if (heap[r][c] < ...) { return false; }
   return _________________ && ...;
}
```

What should be assigned to left:

- ○ `int left = 0;`
- ○ `int left = r + 1;`
- ○ `int left = 2*r;`
- ○ `int left = c + 1;`
- [✓ `int left = 2*c;`]

Explanation

This can be calculated by following the example above.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:27 PM**

Q8.4 If statement
150 Points

```
private boolean isValidHeap(int[][] heap, int r, int c) {
   if (_____________) { return true; }
   int left = ________;
   int right = ....;
   if (heap[r][c] < heap[___][____])  { return false; }
   if (heap[r][c] < ...)  { return false; }
   return __________________ && ...;
}
```

What comparison should be made?

- `if (heap[r][c] < left)  { return false; }`
- `if (heap[r][c] < heap[left][left])  { return false; }`
- `if (heap[r][c] < heap[r][left])  { return false; }`
- [✓ `if (heap[r][c] < heap[r + 1][left])  { return false; }`]
- `if (heap[r][c] < heap[left][c])  { return false; }`
- `if (heap[r][c] < heap[left][c + 1])  { return false; }`

Explanation

In a max heap, we should compare the parent and ensure that it is larger than both of its children. To access one layer down, we should increment r by 1.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:27 PM**

Q8.5 Return
150 Points

```
private boolean isValidHeap(int[][] heap, int r, int c) {
   if (_____________) { return true; }
   int left = ________;
   int right = ....;
   if (heap[r][c] < heap[___][____])  { return false; }
   if (heap[r][c] < ...)  { return false; }
   return __________________ && ...;
}
```

What return should be made?

- `return isValidHeap[r][0] && ...;`
- `return isValidHeap[r + 1][0] && ...;`
- `return isValidHeap[r][c] && ...;`

○ `return isValidHeap[r + 1][c] && ...;`

○ `return isValidHeap[r][left] && ...;`

[✓ `return isValidHeap[r + 1][left] && ...;`]

Explanation

We then recurse by moving down one layer and calling `isValidHeap` on the left and the right child.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:27 PM**

Q9 IterableComparator

650 Points

Suppose we want to write a class `IterableComparator` that compares two iterables. Iterables are compared based on how many items there are available.

For example an `IterableComparator` would consider a list containing [3, 9, 12, 15] to be greater than a set containing {"cat", "dog", "fish"} since the list has 4 items and the set has 3 items.

The code is as follows:

```
1: public class IterableComparator implements Comparator<Iterable> {
2:     public _____ ________(________ a, _________ b) {

3:         int ac = 0;
4:         for (___________) {
5:             _____________;
6:         }

7:         int bc = 0;
8:         for (___________) {
9:             _____________;
10:        }

11:        return _______________;
12:    }
13:}
```

Q9.1 Line 2, blank 1

100 Points

What can go in the first blank on line 2 (after the word `public`)? Check all that apply.

☐ boolean

☐ void

☑ int

☐ None of these

Explanation

This question requires knowledge of the Comparator interface, which requires the `public int compare(___, ___)` method to be implemented. Without this method, the compiler would flag an error.

✓ Correct

Save Answer Last saved on **Feb 26 at 9:27 PM**

Q9.2 Line 2, blank 2
100 Points

What can go in the second blank on line 2 (after your answer to the previous problem)? Check all that apply.

- [✓ compare]
- compareTo
- IterableComparator
- None of these

Explanation

Same explanation as above.

✓ Correct

Save Answer Last saved on **Feb 26 at 9:27 PM**

Q9.3 Line 2, blanks 3 and 4
100 Points

What can go in the third and fourth blanks on line 2, i.e. what are the possible types of a and b? Check all that apply.

- Iterator
- [✓ Iterable]
- Comparator
- Comparable
- None of the above

Explanation

Since the class signature specifies that the type of objects that are being compared are of type

`Iterable`, the type in the parameters must also match.

✓ Correct

Save Answer Last saved on **Feb 26 at 9:27 PM**

Q9.4
100 Points

For your convenience the code is repeated below.

```
1: public class IterableComparator implements Comparator<Iterable> {
2:     public _____ ________(________ a, _________ b) {
3:         int ac = 0;

4:         for (__________) {
5:             ______________;
6:         }

7:         int bc = 0;
8:         for (__________) {
9:             ______________;
10:        }

11:        return ________________;
12:    }
13:}
```

What could go in line 4? Note that there is no "None of the above" choice.

- int x : a
- [✓ Object x : a]
- int i = 0; i < a.size(); i += 1;
- a = new Iterable(); a.hasNext(); a.next();

Explanation

The first answer choice is invalid if the iterable object is not storing `int`s. Since `a` has type Iterable, we can use an enhanced for loop, and all java classes inherit from the Object superclass, so the second option is valid. Next, the Iterable interface does not specify a size() method, so that may not compile. Lastly, you cannot instantiate an interface, so the syntax `new Iterable()` is not valid.

✓ Correct

Save Answer Last saved on **Feb 26 at 9:27 PM**

Q9.5
100 Points

What could go in line 5?

- [ ] ac = ac + a.next();
- [✓ ac = ac + 1;]
- [ ] return ac;
- [ ] None of the above

Explanation

This follows from the choice earlier, where we should increment ac for every element in a.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:27 PM**

Q9.6
150 Points

What could go in line 11 and return the right answer? Assume lines 8 and 9 use the same idea as lines 4 and 5.

- [ ] ac;
- [ ] bc;
- [ ] ac > bc;
- [✓ ac - bc;]
- [✓ (ac - bc) * 2;]
- [ ] None of the above

Explanation

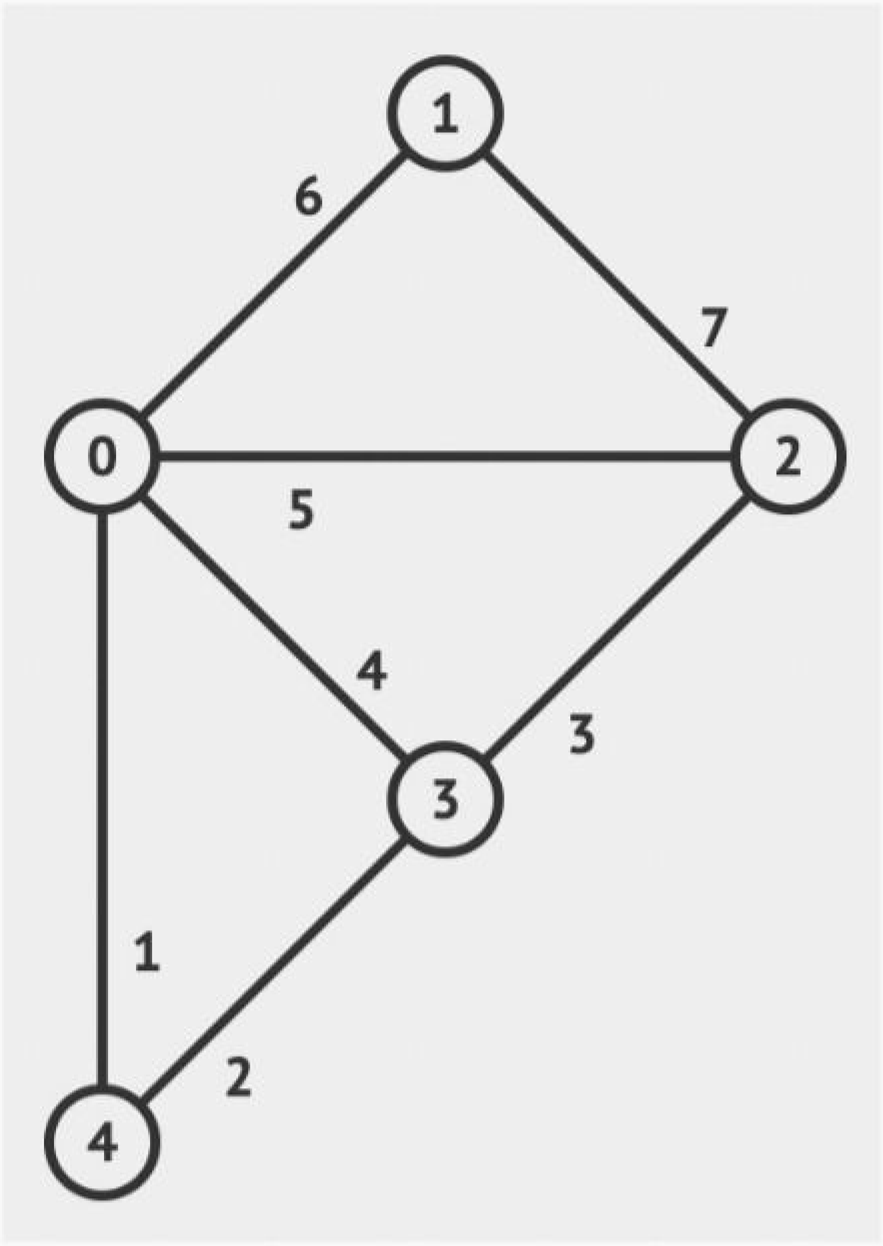
compare should return a negative number if the number of elements in `a` is less than the number of elements in `b`, 0 if the numbers are same, and a positive number of the number of elements in `a` is more than the number of elements in `b`. We can achieve this by comparing the counts stores in `ac` and `bc` - both options above are valid.

✔ Correct

Save Answer Last saved on **Feb 26 at 9:27 PM**

Q10 MSTs
200 Points

Suppose we have the graph below.



Q10.1 Kruskal's Algorithm

100 Points

Recall that in Kruskal's algorithm, we use a disjoint sets object to check for cycles, making exactly one `isConnected` call per edge.

If we run Kruskal's algorithm, for which edges will `isConnected` return true during these cycle checks?

- ☐ isConnected(0, 4)
- ☐ isConnected(3, 4)
- ☐ isConnected(2, 3)
- ☑ isConnected(0, 3)
- ☑ isConnected(0, 2)

☐ isConnected(0, 1)

☐ isConnected(1, 2)

Explanation

Kruskal's algorithm starts by sorting the edges by weight, then adding edges until we hit |V-1|. Per the problem statement, the underlying implementation uses a disjoint sets object to check for cycles. Here is the order of isConnected calls: 1. isConnected(0, 4) --> False 2. isConnected(3, 4) --> False 3. isConnected(2, 3) --> False 4. isConnected(0, 3) --> True 5. isConnected(0, 2) --> True 6. isConnected(0, 1) --> False. At this point we have 4 edges added to our MST and the algorithm terminates. We accepted isConnected(1, 2) as ambiguous, because it would return true if the algorithm continued.

✓ Correct

Save Answer Last saved on **Feb 26 at 9:28 PM**

Q10.2 Prim's algorithm

100 Points

If we run Prim's from vertex 2, in what order will the vertices be added to the MST?

○ 2, 3, 0, 1, 4
○ 2, 3, 0, 4, 1
○ 2, 3, 1, 0, 4
○ 2, 3, 1, 4, 0
[✓ 2, 3, 4, 0, 1]
○ 2, 3, 4, 1, 0

Explanation

Prims adds vertices based on the distance to the MST created thus far. For the first iteration of Prim's, we create a cut separating 2 and the rest of the graph. The crossing edges across this cut are 1-2, 0-2, and 2-3. The minimum crossing edge is 2-3, and by the cut property we then add 3 to the MST. The next cut will separate 2 & 3 and the rest of the graph, with 1-2, 0-2, 0-3, and 3-4 as the crossing edges. The minimum crossing edge is 3-4, and thus 4 is added to the minimum spanning tree. Repeat this two more times to yield the MST generated by Prim's algorithm.

✓ Correct

Save Answer Last saved on **Feb 26 at 9:28 PM**

Q11 Wolf Shirt

0 Points

You're done! I hope you enjoyed 61B! You've earned your wolf shirt.

Feel free to write anything here about the exam or anything else:

Have a great winter break :)

See you around...

✓ Correct

Save Answer Last saved on **Feb 26 at 9:28 PM**

Save All Answers

Submit & View Submission ›