CS 61B Spring 2023

Final

Tuesday May 9, 2023

Intro: Welcome to the CS61B Final!

Your name: Solutions

Your SID: _____ Login: sp23-s000

Location: Clarifications on last page

SID of Person to your Left: _____ Right: _____

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

_____

_____

Signature: _____

Tips:

- For answers which involve filling in a ◯ or ☐, **please fill in the shape completely.** If you change your response, **erase as completely as possible**. Incomplete marks may affect your score.
- ◯ indicates that only one circle should be filled in.
- ☐ indicates that more than one box may be filled in.
- You may not need to use all provided lines, but we **will not give credit for solutions that go over number of provided lines.**
- You may not use ternary operators, lambdas, streams, or multiple assignment.
- 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 with which you are comfortable 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'.

This page is intentionally left blank (except for this sentence and the page number).

Q1 I Got A ...

(800 Points)

```
class A {
    public void f() {
        B me = (B) this;
        g(this);
    }
    public void g(A x) {
        System.out.println("Mom");
        x.h();
    }
    public void h() {
        System.out.println("I got a +2");
    }
}

class B extends A {
    public void g(B x) {
        System.out.println("Brother");
        x.h();
    }
    @Override
    public void h() {
        System.out.println("I got a PB!");
    }
    public static void main(String[] args) {
        // CODE HERE
    }
}
```

Write what the `main` method in class B will print if we inserted and ran the code in `// CODE HERE`. **If it errors, write CE for compile error or RE for runtime error.**

```
A kam = new A();
kam.f();
```

RE

Runtime error when trying to cast kam to B in the f() function.

```
B park = new A();
park.f();
```

CE

Compiler error when trying to initialize an A object as static type B.

```
A mitch = new B();
mitch.f();
```

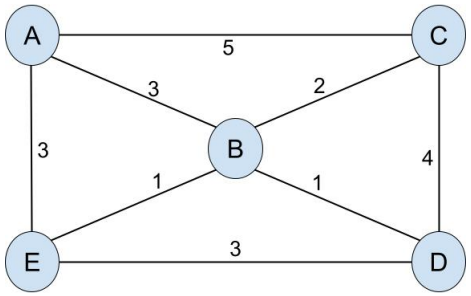
Mom
I got a PB!

We call f(), which then calls g(A) since mitch is static type A. Then, we call B::h due to dynamic type B.

Q2 Edgy Questions

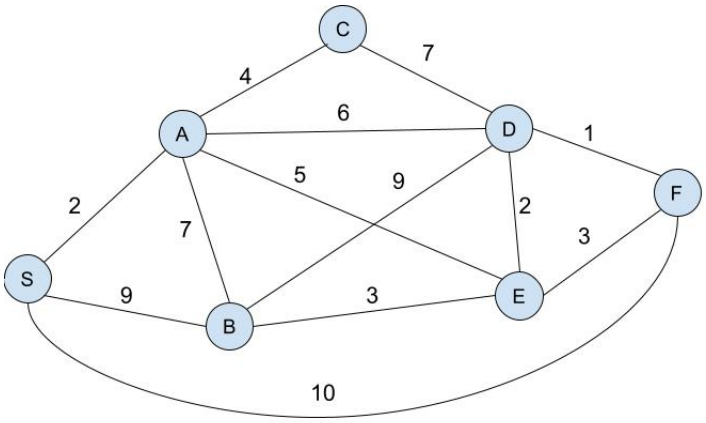
(1200 Points)

(a) ○ True ● False Kruskal's algorithm and Prim's algorithm will produce the same MST if Prim's algorithm is run from B, even if their tie-breaking schemes are different in the graph to the right.



BE, BD, and BC will be picked by both algorithms for any valid MST since they are the smallest weight edges to include those nodes in the graph, since these are the first 3 considered by both Kruskal's after the edges are sorted and the first 3 removed from the PQ for Prim's. The last node to connect, A, has two options: either the edge AE or the edge AB, both of which have weight 3. Picking between these relies on some tie-breaking scheme. If the two have different tie-breaking schemes, then each may product a different MST: consider a scheme for Prim's which tries to pick the largest letters (so it would pick AE) and a scheme for Kruskal's which tries to pick the smallest letters (so it would pick AB).

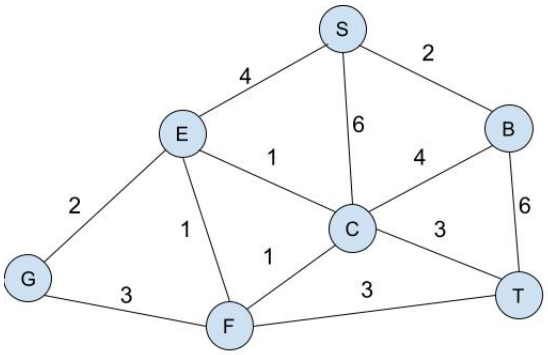
(b) ~~Consider the graph G below.~~



~~Suppose that we have started running Dijkstra's from S on and terminated the program when the Dijkstra's finished finding the shortest path to E. What other nodes' shortest path from S have still have priority infinity?~~ This question was removed from the exam.

☐ S ☐ A ☐ B ☐ C ☐ D ☐ E ☐ F

(c) Consider the graph G with the heuristic h below.



Node	h(v, T)
S	3
B	6
C	2
E	2
F	5
G	12
T	0

Suppose we run A* from node S to node T on graph G. Break ties alphabetically. What is the order that vertices (including S and T) are visited? Please separate vertices with spaces.

S E C B T

The first vertex to visit is the start vertex. The distTo is initialized to ∞ for every value.

The values for E, B, and C are updated to the distance of S to the node plus the h(v, T). distTo is ∞ for G, F, C, and T, and is 6, 8, and 8 for E, B, and C, respectively. Thus, the smallest is 6 so E is popped off next. This updates G, F, and C distTo to 18, 10, and 7.

Now the smallest distTo is C, so it's popped off. The distTo of B, F, and T now become 8, 10, and 9. Notice that updates were not made for B and F since there is a faster way to get to those nodes. Next, B is popped off, which updates values once more for T (C was already visited) to be 8. T is now the smallest by distTo so it is popped off next.

(d) A binary tree is constructed of nodes that are instances of the following class:

```
public class Node {
   public int val;
   public Node left;
   public Node right;
}
```

Consider the following method:

```
public static Node mystery(Node root) {
   if (root.right == null) { return root; }
   else { return mystery(root.right); }
}
```

Which of these statements is correct about the method regardless of the contents of the tree when passed a reference to the root node of a binary tree?

■ It returns the last node visited by an in-order traversal

☐ It returns the last node visited by a post-order traversal

☐ It returns the last node visited by a level-order traversal

☐ None of the above

The mystery function recurses to the right side of the tree until there is no more right node. This is the rightmost node in the tree without a right child. An in-order traversal visits a node once all of its children on the left have been visited, which is equivalent to visiting a node just before the children on its right are visited. The last node to be visited by an in-order traversal is thus the rightmost node without a right child (since if it had a right child, the node would be visited and then its right subtree would be looked at.

The other traversals are not valid as they do not follow the logic above, but you can also construct counter-examples of when they are not true. A post-order traversal for a complete tree would visit the root last, since every node on the left and every node on the right would have to be visited. For a level-order traversal, consider an almost-complete tree that is missing its lowermost rightmost node. The last node would the node just to the left of that node (on the lowest layer, all the way to the right). However, this mystery function would return this node's parent (in the layer above).

Q3 Value Town

1200 Points

Mihir really wants to rate the value for each food he eats at Berkeley. He makes a Food object to save the food name and cost per meal. He then stores Foods as keys and ratings as values in a HashMap. Unfortunately, he ran into bugs in his program and scattered print statements to check the behavior.

For this problem, assume the MyHashMap is initialized with an array of length 4. Our MyHashMap uses external chaining as implemented in class to insert new items. You may assume MyHashMap resizes by doubling the size of the array after the load factor is greater than the the maximum load factor of 1.

```
1  public class Food {
2      public String name; public int price;
3      public Food(String name, int price) { this.name = name; this.price = price; }
4
5      @Override
6      public int hashCode() { return price; }
7      public static void main(String[] args) {
8          HashMap<Food, Integer> ratings = new MyHashMap<>();
9
10         Food burrito = new Food("burrito", 8);
11         ratings.put(burrito, 5);
12         ratings.put(new Food("taco", 6), 3);
13         System.out.println(ratings.getOrDefault(burrito, -1));
14         // getOrDefault: Returns the value or -1 if the map contains no mapping for the key.
15
16         burrito.price += 3;
17         System.out.println(ratings.getOrDefault(burrito, -1));
18
19         ratings.put(new Food("burger", 13), 1);
20         ratings.put(new Food("pizza", 7), 4);
21         System.out.println(ratings.getOrDefault(burrito, -1));
22
23         ratings.put(new Food("sandwich", 9), 2);
24         System.out.println(ratings.getOrDefault(burrito, -1));
25     }
26 }
```

(a) What is the output of line 13? 5

(b) What is the output of line 17? -1

Since the hashcode is calculated from the price of the Food, "burrito" was originally put in bucket 8 mod 4 = 0. However, the price increased to 11, so when we try to find "burrito", we search in bucket 11 mod 4 = 3. Then, we return -1 since we cannot find burrito.

(c) What is the output of line 21? -1

(d) What is the output of line 24? 5

When we inserted "sandwich" on line 20, we exceeded our maximum load factor of 1. As such, we must

recalculate the hashcode of each item and re-bucket them according to the new number of buckets, which is specified to double. Thus, "burrito" gets placed in bucket 11 $\mod 8 = 3$. Then, when we get burrito on line 24, we are able to go to the right bucket and get the item.

(e) The bucket at index 0 is empty. ●True ○False

(f) The resulting map has a bucket that contains at least 2 Food objects. ○True ●False

(g) Suppose Mihir runs `ratings.put(new Food("sandwich", 9), 3);` below line 24 again. The number of elements in the map will increase. ●True ○False

The Food object on line 24 and the Food object in the problem statement are distinct, despite having the same name and price. Since we do not override the equals method, they are treated as two different keys, and thus a new entry gets placed into the MyHashMap, rather than replacing the value of the previous key-value pair.

Q4 Asymptotics

(2000 Points)

Give the runtime of the following `enigma` functions in terms of N. Your answers should be as simple as possible, excluding unnecessary constants like log bases and lower-order terms. Assume there is no limit on the size of an **`int`** (otherwise all run times are technically constant).

`enigma1`

$\Theta($ `2^N * N` $)$

```
public static void enigma1(int N) {
    for (int i = 0; i < Math.pow(2, N); i += 5) {
        for (int j = N; j >= 0; j -= 1) {
            System.out.println("Never gonna give you up");
        }
    }
}
```

Both loops are independent of each other (the value of j does not depend on the value of i). The outer loop goes for `2^N` whereas the inner loop goes for `N`. From independence, these are multiplied.

`enigma2`

$\Theta($ `log(N)` $)$

```
public static void enigma2(int N) {
    for (int i = N * 8; i > 1; i /= 4) {
        for (int j = 0; j < 10000; j += N) {
           System.out.println("Never gonna let you down");
        }
    }
}
```

Both loops are independent of each other (the value of j does not depend on the value of i). The outer loop goes for `log(N)` since N is being divided by 4 every time, which is done a logarithmic number of times (the 8 just adds 2 more times, which is constant) whereas the inner loop goes for 1 since the lower and upper bounds of j are fixed, so as N tends to ∞ the inner loop is constant. From independence, these are multiplied.

`enigma3`

$\Theta($ `N^2` $)$

```
public static void enigma3(int N) {
    for (int i = 2; i < N; i *= 2) {
        for (int j = 0; j < N * N; j += i/2) {
           System.out.println("Never gonna turn around");
        }
    }
}
```

Set up the following table which maps values of i, the ranges of j, and the increment of j (i/2) to find the number of iterations of work done per value of i. Then, sum those together.

i	2	4	8	16	...	N
j	$0 \to N^2$	$0 \to N^2$	$0 \to N^2$	$0 \to N^2$	$0 \to N^2$	$0 \to N^2$
i/2	1	2	4	8	...	N/2
iterations	N^2	$N^2/2$	$N^2/4$	$N^2/8$	...	2N

This becomes a dominating sum, since there's a multiplicative factor between terms. That leads to the first term (the largest term) dominating which gives the runtime.

enigma4

$\Theta($ N^(N-1) $)$

```
public static void enigma4(int N) {
    enigma4(N, N);
}
public static void enigma4(int x, int n) {
    if (x == 1) { return; }
    else {
        for (int i = 0; i < n; i += 1) {
            enigma4(x - 1, n);
        }
    }
}
```

Each node can be thought of as a pair of (x, N) values. The x starts at N from the first enigma4 function and gets decremented in the second by 1 within each iteration of the for loop. Notice that the for loop is bounded by N, which never changes since the recursive call passes in the initial value of n to every subsequent call. The return case is when x is 1. Therefore, we can generalize the sum as follows:

On some level k, where the first node with inputs N, N is considered level 1, there are N^{k-1} nodes. There are $N-1$ total levels, since the initial start is when x is N and the ending is when x is 1 where x is decremented at every level. Since there is constant work at every node, the number of nodes is the runtime. Thus, the total sum is $1 + N^2 + N^3 + ... + N^{N-1}$. This is a dominating sum since there's a multiplicative factor of N between terms, so the final term (the largest term) is the runtime.

Now, consider the following function ben. For the questions below, assume that the pseudocode is translated properly into Java and the program compiles.

```
public static void ben(int N) {
    if (N <= 625) { return; }
    for (int i = 0; i < N; i += 1) {
        System.out.println("And desert you.");
    }
    int answer = _____________;
    for (int j = 0; j < answer; j += 1) {
        ben(N / answer);
    }
}
```

What is the runtime if we set `answer = 0` ?

○ $\Theta(\log N)$ ● $\Theta(N)$ ○ $\Theta(N \log N)$ ○ $\Theta(N(\log N)^2)$ ○ $\Theta(N^2)$ ○ None of the above

There are no recursive calls; the print loop runs N times and the function returns.

What is the runtime if we set `answer = 2` ?

○ $\Theta(\log N)$ ○ $\Theta(N)$ ● $\Theta(N \log N)$ ○ $\Theta(N(\log N)^2)$ ○ $\Theta(N^2)$ ○ None of the above

There are 2 recursive calls and the input is reduced by a factor of 2 per function call. For a given recursive level k, starting with k as 0 for the first iteration with 1 call for the function and an input of N, the amount of work on any level is the number of nodes multiplied by the amount of work at that node. This is $2^k * N/2^k$ = N (if this reasoning is confusing, try drawing out a couple of levels, and note the input size at each node, the work per node, and the number of nodes per level). There are a logarithmic number of time the input can be divided by 2 until it reaches some bounded constant (in this case, 625). So, the work per level is multiplied by the number of levels.

What is the runtime if we set `answer = N/2` ?

○ $\Theta(\log N)$ ● $\Theta(N)$ ○ $\Theta(N \log N)$ ○ $\Theta(N(\log N)^2)$ ○ $\Theta(N^2)$ ○ None of the above

There are N/2 recursive calls and input is reduced by a factor of N/2 per function call. Reducing the input size by this much means every subsequent function call has a fixed input size of 2 (N/(N/2) = 2). That triggers the base case on every subsequent function call. There are N recursive calls, each of which is a constant amount of work (to check base case).

What is the runtime if we set `answer = NlogN` ?

○ $\Theta(\log N)$ ○ $\Theta(N)$ ● $\Theta(N \log N)$ ○ $\Theta(N(\log N)^2)$ ○ $\Theta(N^2)$ ○ None of the above

There are NlogN recursive calls and the input is reduced by a factor of NlogN per function call. That reduces the input size to N/NlogN = 1/logN. In asymptotic analysis, note that N tends to ∞ which means 1/logN tends to 0. Each recursive call will thus trigger the base case. There are NlogN recursive calls, each of which is a constant amount of work (to check base case).

What is the runtime if we set `answer = N^2` ?

○ $\Theta(\log N)$ ○ $\Theta(N)$ ○ $\Theta(N \log N)$ ○ $\Theta(N(\log N)^2)$ ● $\Theta(N^2)$ ○ None of the above

There are N^2 recursive calls and the input is reduced by a factor of N^2 per function call. That reduces the

input size to $N/N^2 = 1/N$. In asymptotic analysis, note that N tends to ∞ which means 1/N tends to 0. Each recursive call will thus trigger the base case. There are N^2 recursive calls, each of which is a constant amount of work (to check base case).

What is the runtime if we set `answer = N^N`?

○ $\Theta(\log N)$ ○ $\Theta(N)$ ○ $\Theta(N \log N)$ ○ $\Theta(N(\log N)^2)$ ○ $\Theta(N^2)$ ● None of the above

This is an extension of the previous problem. There are N^N recursive calls and the input is reduced by a factor of N^N per function call. That reduces the input size to $N/N^N = 1/N^{N-1}$. In asymptotic analysis, note that N tends to ∞ which means $1/N^{N-1}$ tends to 0. Each recursive call will thus trigger the base case. There are N^N recursive calls, each of which is a constant amount of work (to check base case).

This page is intentionally left blank (except for this sentence and the page number).

Q5 Wordle

(2400 Points)

Write a class to store a Wordle dictionary. Your solution must allow for the following functions:

`insert(String word)`: Inserts a new word of length W into the dictionary and takes $O(W)$ time. **All `word` arguments are unique, non-null, and consists of only CAPITAL letters or the empty string**.

`contains(String word)`: Returns true if the dictionary contains the word and takes $O(W)$ time. **All `word` arguments are unique, non-null, and consists of only CAPITAL letters or the empty string**.

`wordList()`: Returns a list of all words in the dictionary, in alphabetical order. This must take $O(WN)$ time, where N is the number of words in the dictionary.

`getRandomWord(Random random)`: Returns a random word from the dictionary. This must take $O(W)$ time, so calling `wordList` will exceed your runtime. All words must be equally likely to be returned by your function regardless of the words in the dictionary.

You may assume all string operations in the reference sheet and string concatenation run in $O(1)$ time.

```
class Wordle { // we will use a modified trie data structure!
    private boolean valid;
    private Wordle[] children;
    private int size;

    public Wordle() {
        valid = false;
        children = new Wordle[26];
        size = 0;
    }

    private int charToInt(char c) { return c - 'A'; } // Converts A to 0, B to 1,... Z to 25
    private char intToChar(int i) { return (char) ('A' + i); } // Converts 0 to A, 1 to B,... 25 to Z

    public void insert(String word) {
        size += 1;
        if (word.length() == 0) {

            this.valid = true;
            return;
        }

        int index = charToInt(word.charAt(0));

        if (this.children[index] == null) {

            this.children[index] = new Wordle();
        }
        this.children[index].insert(word.substring(1));
    }
```

```
public boolean contains(String word) {
    return ( (__1__) && (__2__) ) || ( (__3__) && (__4__) && (__5__) );
}
```

Put the letter that corresponds to the code to run `contains`. An answer choice may be used more than once.

(a) `this.valid`
(b) `this == null`
(c) `word.length() == 0`
(d) `this.contains(word.substring(1))`
(e) `children[charToInt(word.charAt(0))]==null`
(f) `children[charToInt(word.charAt(0))].contains(word.substring(1))`
(g) `!this.valid`
(h) `this != null`
(i) `word.length() != 0`
(j) `!this.contains(word.substring(1))`
(k) `children[charToInt(word.charAt(0))]!=null`
(l) `!children[charToInt(word.charAt(0))].contains(word.substring(1))`

1. a 2. c 3. i 4. k 5. f

```
public List<String> wordList() {
    List<String> words = new ArrayList<>();

    if (this.valid) {  words.add(""); }
    for (int i = 0; i < 26; i += 1) {
        if (children[i] != null) {

            for (String s: children[i].wordList()) {

                words.add(intToChar(i) + s);
            }
        }
    }
    return words;
}

public String getRandomWord(Random random) {
    int randNum = random.nextInt(this.size);

    if (randNum == size - 1 && valid) { return ""; }
    int letter = -1;
    while (randNum >= 0) {
        letter += 1;

        if (children[letter] != null) {

            randNum -= children[letter].size;
        }
    }
    return intToChar(letter) + children[letter].getRandomWord(random);
}
}
```

Q6 Sorta Sorting

(1400 Points)

Emily loves geese, but they always wander about. Typically, Emily needs to do some sort-esque work with the geese's weight, but comparison based sorts take at least $O(N \log N)$ time which gives her geese time to run away! We need to find a way to perform her work in linear time.

For this entire question, arr is an array of at least 100 nonnegative integers (0 is a nonnegative integer). `sort` is an comparison-based sort that sorts in ascending order in $O(N \log N)$ time **that non-destructively returns a copy of a sorted array.**

Consider the following functions:

Part A:

```
// arr is an array of at least 100 nonnegative integers
public static int foo(int[] arr) {
    int[] sortedArr = sort(arr);
    return sortedArr[9];
}
```

(i) Describe in 10 words or less what `foo` does.

Picks the 10th smallest element.

(ii) Rewrite `foo` so it runs in $O(N)$ time instead of $O(N \log N)$ time.

```
public static int foo(int[] arr) {

    MinHeap<Integer> h = new MinHeap<>();

    for (int i = 0; i < 10; i++) {

        h.insert(arr[i] * -1);
    }
    for (int i = 10; i < arr.length; i++) {

        h.insert(arr[i] * -1); // we insert before we pop or we might pop the 10th smallest element.

        h.deleteMin();
    }
    return h.deleteMin() * -1;
}
```

The size of the heap never exceeds 10 items. The first 10 items in any order are added, and then every subsequent item is added, the heap is rebalanced, and then the minimum is deleted, then rebalanced. Note that since we can bound the number of items in the heap to a constant, the runtime of deleting the minimum and inserting are 10 and log10 respectively which are also constants. Thus, the overall runtime is N, since the first loop is only 10 iterations (a constant) but the second loop is N - 10 iterations.

Part B:

```
// arr is an array of least 100 nonnegative integers
public static int bar(int[] arr) {
    int[] sortedArr = sort(arr);
    int i = -1;
    for(int j = 0; j < sortedArr.length; j += 1) {
        if (sortedArr[j] - i >= 2) {
            return i + 1;
        }
        i = sortedArr[j];
    }
    return i + 1;
}
```

(i) Select the option that properly describes what `bar` does.

○ Returns the smallest nonnegative integer
○ Returns the smallest duplicate nonnegative integer
● Returns the smallest nonnegative integer not in the list
○ Returns the smallest nonnegative integer that is not duplicated in the list

bar sorts an array, then iterates through it checking this condition with regards to i. Consider the initial case before any updates to i, when it is -1. The check is whether sortedArr[j] + 1 $\geq$ 2 which is the same as sortedArr[j] $\geq$ 1, in which case i + 1 is returned (which would be 0). Note additionally that in order for this condition to be false, sortedArr[j] must be 0 (since arr is an array of nonnegative integers). This is confusing, but it means that if the value in the array is not 0, then 0 is returned (which is not in the list). If the value in the array is 0, then the next value is looked at, which now has the check being sortedArr[j] $\geq$ 2. This time, if the value in the array is 1, the next iteration happens; otherwise, 1 is returned. This continues on and on, but is the intuition for why this function returns the smallest nonnegative integer not in the list (since if it was in the list, it would update the check for the smallest value being present). We can make this judgement to keep cycling through the array since the array is sorted.

(ii) Rewrite `bar` so it runs in $O(N)$ time instead of $O(N \log N)$ time.

```
public static int bar(int[] arr) {
    boolean[] counts = new boolean[arr.length];
    for (int i = 0; i < arr.length; i += 1) {

        if (arr[i] < arr.length) {

            counts[arr[i]] = true;
        }
    }
    for (int i = 0; i < arr.length; i += 1) {

        if (!counts[i]) {

            return i;
        }
```

```
    }

    return arr.length;
}
```

The variable i represents the index within the array that also corresponds to the boolean counts array for the same index. In order to utilize the counts array, we use the first loop to initialize it with values and then use the second loop to find the smallest nonnegative integer not in the list. Let's begin by approaching the algorithm conceptually. Regardless of if the initial array is sorted, we can look through the values and update counts to track that we have seen a specific value. For example, if arr was an array containing 4, 19, 0, 2, and 1, there would be a corresponding array counts that would have the values at the indices 0, 1, 2, and 4 all being true. Note that this is taking the values of the arr array and using that as indices. An ArrayIndexOutOfBoundsError is avoided by making the check to see whether the value is a valid index for the array.. The last part is to leverage this; the counts array is technically "sorted" in terms of its indices, so those are iterated through. The first index in the counts array that has a false value is returned directly; else every number from 0 to N - 1 pops up in an array of length N so N is returned as the smallest nonnegative integer not in the list.

While formal proofs are not required in this class, here is a longer explanation for why the underlying logic works. Say there is a counts array and an input arr array set up as per the question. If arr is a sequence of numbers, the sequence with the maximum smallest nonnegative integer not in the list (referred to as "the goal value") is simply a list containing every value from 0 to N - 1, which yields N. Any other list either has duplicates or a missing value in the sorted order of the numbers. If there is a duplicate, then those duplicates reduce the goal value since they now take up space of other values (which, if omitted, are now the goal value; for example, if the list had 0 0 1 2 3 ... the final value could only be N - 2, or there would have to be another missing value i.e. 0 0 1 2 3 ... [skipped value] ... N - 1). This missing value must be less than N, else there would have to be more than N values in the array (to account for the extra values). Going above N has similar logic; it means that there is some smaller value that was skipped which would be returned as the goal value before this larger value was reached.

Q7 Actually Sorting

(1200 Points)

Part 1:

(a) What is the output of Hoare's partition algorithm using the first index as a pivot?

Before Value	4	6	2	5	1	7	3	8
After Value	1	3	2	4	5	7	6	8

(b) ○ True ● False There exists a comparison algorithm that sorts any list of 2023 numbers in at most 2024 comparisons in the worst case.

(c) ● True ○ False Any comparison-based sorting algorithm can be made stable by modifying the comparison operation to also consider the original index of the elements being compared.

Part 2:

(a) You are sorting a list of N distinct integers.

(i) What is the runtime of quick sort if your quick sort algorithm always picks the 10th smallest element as the pivot? $\Theta(N^2)$
This is asymptotically equivalent to picking the smallest element as the pivot, which we know is $\Theta(N^2)$.

(ii) What is the runtime of quick sort if your quick sort algorithm always picks the 10th percentile (10% smallest) element as the pivot? $\Theta(N \log N)$
This is asymptotically equivalent to picking the median (50% smallest) element as the pivot, which we know is $\Theta(N \log N)$.

(b) Now consider a new sorting algorithm, `quickMergeSort`. The algorithm is given in pseudocode below.

```
quickMergeSort(list) {
        If list.size <= 1 return list
        Pick a pivot in constant time
        Partition around pivot
        mergeQuickSort(left partition)
        mergeQuickSort(right partition)
}
mergeQuickSort(list) {
        If list.size <= 1 return list
        Split list into two halves
        left half = quickMergeSort(left half)
        right half = quickMergeSort(right half)
        merge(left half, right half)
}
```

(i) Whats the runtime if `quickMergeSort` always picks the median as the pivot? $\Theta(N \log N)$
This is the same as the best case of mergesort/quicksort, which is $\Theta(N \log N)$.

(ii) Whats the runtime if `quickMergeSort` always picks the minimum as the pivot? $\Theta(N \log N)$
For every call of quickMergeSort, the minimum item is always selected as the pivot. Thus, the left partition is of size 0, and the right partition is of size N-1. The recursive call on the left partition will not contribute any more work, but on the right partitions we run mergeQuickSort. This splits up the list into lists of size N-1 (we will be a little loose with our math; this is ok since it is a constant factor). We then run quickMergeSort on each half, each one then producing another call to mergeQuickSort of size $\frac{N-2}{2}$. This pattern continues, and the logarithmic decrease of the size of each recursive call results in a $\Theta(N \log N)$ runtime.

Part 3:

The intermediate steps in performing various sorting algorithms on the same input list are shown. The steps do not necessarily represent consecutive steps in the algorithm (that is, many steps are missing), but they are in the correct sequence.

Select the algorithm it illustrates from among the following choices: LSD (least significant digit) radix sort, MSD (most significant digit) radix sort, insertion sort, selection sort, and heap sort.

(a) ● LSD Radix ○ MSD Radix ○ Insertion ○ Selection ○ Heap

```
1430, 3292, 7684, 9002, 1001, 595, 4243, 1338, 4393, 130, 193
1430, 130, 1001, 3292, 9002, 4243, 4393, 193, 7684, 595, 1338
1001, 9002, 1430, 130, 1338, 4243, 7684, 3292, 4393, 193, 595
```

(b) ○ LSD Radix ○ MSD Radix ● Insertion ○ Selection ○ Heap

```
1338, 1430, 3292, 7684, 193, 595, 4243, 9002, 4393, 130, 1001
193, 1338, 1430, 3292, 7684, 595, 4243, 9002, 4393, 130, 1001
193, 595, 1338, 1430, 3292, 7684, 4243, 9002, 4393, 130, 1001
```

(c) ○ LSD Radix ○ MSD Radix ○ Insertion ○ Selection ● Heap

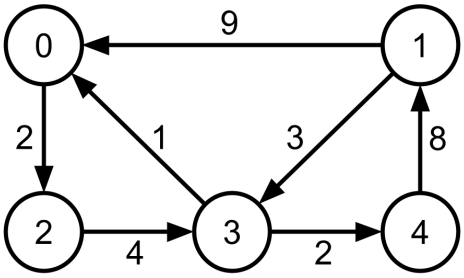
```
1430, 3292, 7684, 9002, 1001, 595, 4243, 1338, 4393, 130, 193
9002, 7684, 4393, 4243, 3292, 1001, 595, 193, 1338, 1430, 130
130, 4393, 4243, 3292, 1001, 595, 193, 1338, 1430, 7684, 9002
```

Q8 The Javaugean Stables

(2600 Points)

Part A:

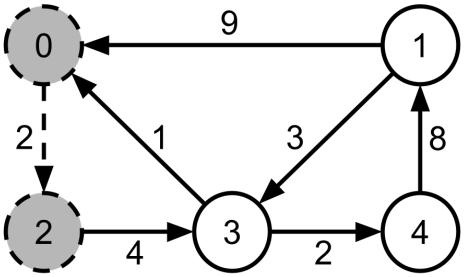
8 months after successfully defeating the Lambdanean Hydra, Heracles is tasked with another labor: cleaning the Javaugean Stables.



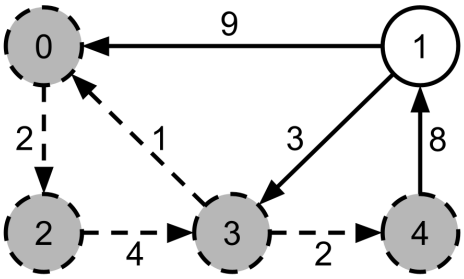
The Javaugean Stables are represented as a directed graph with positive edge weights, with nodes denoting individual stables, and directed edges representing paths between them. You may assume that the graph is fully connected, and that the stables can be fully flooded at some depth.

The Alpheus river runs through stable 0, and Heracles plans to divert its waters through all the stables in order to clean them. Water can flow along paths as long as its depth is at least the weight of the edge. **Help Heracles determine the minimum depth of water needed in order to flood every stable with water starting from stable 0.**

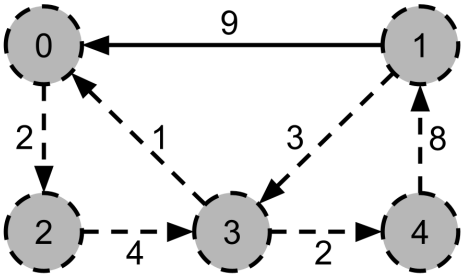
Your code must run in $O(E \log E)$ time, and your code's runtime may not depend on the weights of the edges.



In the above example, if Heracles sets the depth of water to 2, water can flow along the edge 0 to 2, so stable 2 gets flooded.



If Heracles sets the depth of water to 4, stable 3 and stable 4 also get flooded.



In order to flood all stables, Heracles needs to set the depth of water to at least 8.

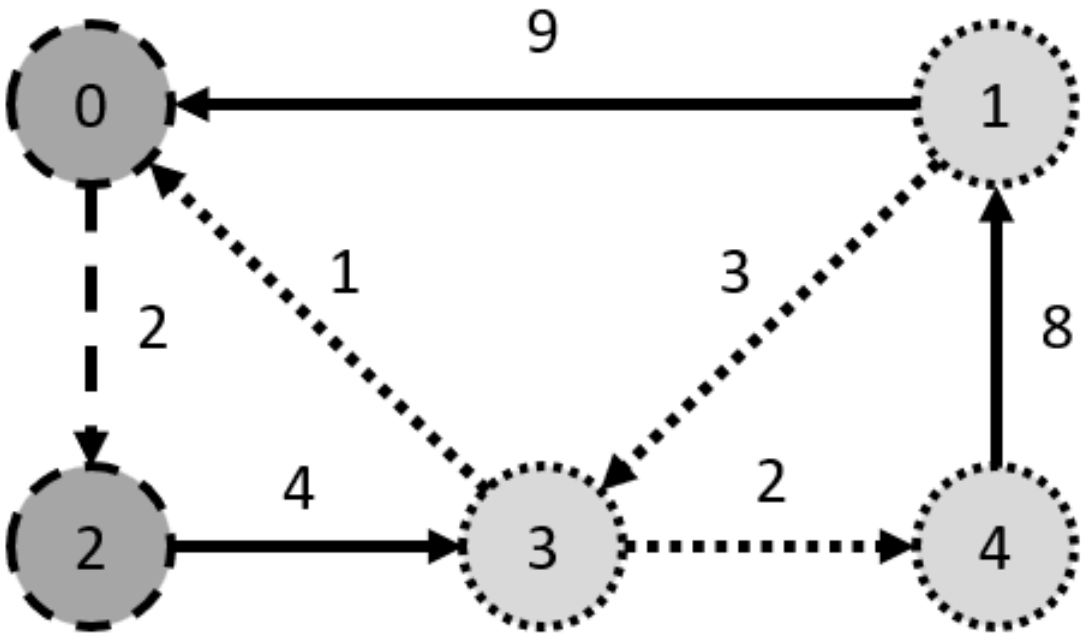
```
public class StablesQuestion {
    public static int Stables(Graph g) {
        MinHeap<GraphEdge> h = new MinHeap<>();
        Queue<Integer> q = new Queue<>();

        boolean[] visited = new boolean[g.size()];
        int depth = 0;

        q.insert(0);

        while (h.size > 0 || q.size > 0)) {
            if (q.size() > 0) {

                int n = q.remove();

                visited[n] = true;

                for (GraphEdge i : g.neighbors(n)) {

                    h.insert(i);
                }
            } else {

                GraphEdge smallestedge = h.removeMin();

                if (!visited[smallestedge.to]) {

                    q.insert(smallestedge.to);

                    depth = Math.max(depth, smallestedge.weight);
                }
            }
        }
        return depth;
    }
}
```

This question is similar to a breadth-first search (due to the presence of a queue) modified to meet the depth calculation requirement of the problem. The visited array is initialized to the size of the nodes within the graph, and it begins by adding 0 since that is the start node. Then, there are two parts: if there's anything in the queue, the first node is popped off, the visited array is updated, and then the edges are added into the minheap. Note that the minheap will rebalance as edges are added in so the smallest weight edge is at the top - since the goal is to minimize the overall depth. The else case handles what happens if the queue is empty from the node being popped off; the smallest edge is taken off. The visited array is now used to ensure that a repeat access of a node is avoided (to avoid a cycle of alternating between the same nodes) and the queue adds that node while updating the depth to account for the weight of this smallest edge.

This solution meets the runtime requirements. The minheap and the queue at any point is bounded by the number of edges present, since once a node is visited, it's not visited again (the point of the boolean visited array), so the edges from that node would not be considered more than once. Within the loop, insertion would take logE time for the heap and constant time for a queue (could store a pointer to the last item and append directly). Note that despite the for loop nested within the while loop, we can still say that the total number of items ever added and removed from the minheap is upper bounded by the number of edges present, so the runtime is still ElogE.

Part B:

Heracles realizes that the Peneus river flows through stable 1, and can also be used to clean stables. By using both rivers, Heracles hopes to reduce the depth he needs to dig. Write an algorithm to determine the new minimum depth.



In the above example, Heracles only needs to dig to a depth of 3 to flood all stables now. Stables 0 and 2 are flooded from stable 0 (Alpheus river) via the dashed lines, and stables 1, 3, and 4 are flooded from stable 1 (Peneus river) via the dotted lines.

```
// Hint: Find a way to change the graph so you can use your answer to part A.
public static int TwoRiverStables(Graph g) {
    Graph newGraph = new Graph(g);

    newGraph.addEdge(new GraphEdge(0, 1, 0));
    return Stables(newGraph);
}
```

As per the hint, there must be a modification for the graph that leverages the same algorithm from part A. By adding a new edge between the 0 and 1 nodes with weight 0, we can guarantee that this edge will be chosen first (unless there are other edges with depth 0, which would mean that any of these could be chosen) before the positive weight edges. Then, the search can be done from nodes that are connected to either 0 or 1.

Nothing on this page is worth any points.

61B

(0 Points)

Please pick a integer between 1 to 10000, inclusive on both ends, that you believe will be the median of all the integers guessed for this question.

5000

Feedback

(0 Points)

Leave any feedback, comments, concerns, or drawings below!

Semester's Yokover. Have a great summer!