The University of Michigan
Electrical Engineering & Computer Science
EECS 281: Data Structures and Algorithms
Winter 2024

FINAL EXAM
KEY 2 – ANSWERS

Thursday, April 25, 2024
8:00AM – 10:00AM

Uniqname: _____ **Student ID:** _____

Name: _____

Uniqname of person to your left: _____

Uniqname of person to your right: _____

Honor Pledge:
"I have neither given nor received unauthorized aid on this examination,
nor have I concealed any violations of the Honor Code."

Signature: _____

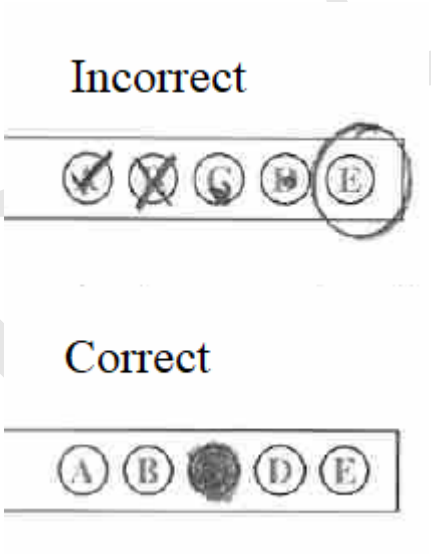
INSTRUCTIONS:

- The exam is closed book and notes except for one 8.5"x11" sheet of handwritten notes (both sides). All electronic device use is prohibited during the exam (calculators, phones, laptops, etc.).
- Print your name, student ID number, and uniqname **LEGIBLY**. Sign the Honor Pledge; this pledge applies to both the written and multiple choice sections. Your exam will not be graded without your signature.
- Record the **UNIQNAME** of the students to both your left and right. Write down `nullptr` if you are at the end of a row and there is no one on a given side.
- Record your **UNIQNAME** at the top of each odd-numbered page in the space provided. This will allow us to identify your exam if pages become separated during grading.
- This exam booklet contains **BOTH** the multiple choice and written portions of the exam. Instructions for each portion are provided at the beginning of their respective sections.
- **DO NOT** detach any pages from this booklet.
- There should be 24 pages in this book; if you are missing any pages, please let an instructor know.

Multiple Choice Portion [60 points]

MULTIPLE CHOICE INSTRUCTIONS:

- Select only **ONE** answer per multiple choice question.
- There are 24 multiple choice questions worth 2.5 points each.
- Record your choices for all multiple choice questions using the circles in the boxed area next to each question. Use a number 2 pencil, not a pen (so that you may change your answer). Fill the circle completely. There is no partial credit for multiple-choice questions. Make sure you bubble in the correct letter. See the example below for how to select your answer.



- There is no penalty for wrong answers: it is to your benefit to record an answer to each multiple-choice question, even if you're not sure what the correct answer is.
- The questions vary in difficulty — try not to spend too much time on any one question. Use your time wisely.
- When asked about memory complexity, consider all possible sources (stack and heap).

Record your answers in the bubbles next to each question.

1. Hashing FUN-ctions

Ⓐ Ⓑ Ⓒ **Ⓓ** Ⓔ

Which of the following statements is/are **TRUE** about a hash table that uses open addressing?

I. A hash function could map multiple keys to the same integer value

II. The average number of probes to find an element in the hash table may change when the load factor increases

III. Some keys may rehash to the same bucket when the size of a hash table's underlying array increases

A) I only

B) I and II only

C) I and III only

D) I, II, and III

E) None of the above

All of these choices require a basic understanding of hash tables. I is a collision, II shows that probes changes with load factor, III can still happen even after growing.

2. Choose Wisely

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

Which of the following methods has the best time complexity for optimally solving the fractional Knapsack problem?

A) Brute Force

B) Greedy

C) Dynamic Programming with a 1-dimensional memo

D) Dynamic Programming with a 2-dimensional memo

E) Branch and Bound

Brute force is much slower (exponential), both DP are quadratic, branch and bound is a speedup of exponential, but greedy is $O(n \log n)$.

3. MSTs on a Plane

Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

Let $G = (V, E)$ be a sparse, simple, connected graph containing **EXACTLY** one cycle with C vertices. If you are given the sum of all edge weights in G and a sorted array of the edge weights in the cycle, what is the best possible time complexity of calculating the total weight of the minimum spanning tree of G?

A) $\Theta(1)$

B) $\Theta(C)$

C) $\Theta(min(C, E - C))$

D) $\Theta(E \log(C))$

E) $\Theta(E \log(E))$

If there's only one cycle, we only need to remove one edge to turn the graph into a tree. Choose the longest edge, which is last in the sorted list of edges *in the cycle*, and subtract its weight.

Record your answers in the bubbles next to each question.

4. Name That Probe! (A) (B) (C) (D) (E)

Consider the following hashing function and hash table. If the contents of the hash table were produced by inserting `Elk`, `Ant`, `It`, `Is`, `Elf`, and `Elm` in this order (with no other operations), which collision resolution method does the hash table use? *Hint: 'A' words hash to 0, 'E' words hash to 4, and 'I' words hash to 8.*

```
1  int hash_1(string s) {
2    if (s.empty())
3      return 0;
4    else
5      return s.front() - 'A';
6  } // hash_1()
```

Ant	Elm			Elk	Elf			It	Is
0	1	2	3	4	5	6	7	8	9

A) Linear probing (i.e., $(\texttt{hash\_1}(s) + i) \mod 10$)

B) Quadratic probing (i.e., $(\texttt{hash\_1}(s) + i^2) \mod 10$)

C) Cubic probing (i.e., $(\texttt{hash\_1}(s) + i^3) \mod 10$)

D) Separate chaining

E) None of the above

Since there are only buckets shown and no linked lists, it can't be separate chaining. Start by making a table of the hash function for each string (Elk maps to 4, etc.), then run through each answer until you find that it is wrong.

5. The Truth About Knapsacks (A) (B) (C) (D) (E)

Which of the following claims about solving the 0-1 Knapsack problem is **TRUE**?

A) The brute force solution for solving the 0-1 Knapsack problem has a lower space complexity than the top-down dynamic programming solution

B) The best runtime complexity of a greedy approximation is $\Theta(MN^2)$, where M is the capacity of the knapsack, and N is the number of available items

C) Branch and bound is not applicable to solving the 0-1 Knapsack problem because there is no known approximation that underestimates total value for use in pruning

D) There is no bottom-up dynamic programming solution for the 0-1 Knapsack problem because there are no overlapping subproblems

E) Divide and conquer can be used to optimally solve the 0-1 Knapsack problem in worst-case $O(N \log N)$ time, where N is the number of available items

A: Top down DP needs quadratic space, whereas brute force is linear (list of which items chosen in the current solution under construction, plus linear for best ever seen). B: $\Theta(MN)$. C: when we prune in a maximization problem, we want the estimator for remaining to be an overestimate: use fractional greey (alternately, simple greedy is $\leq$ reality. D: we showed the solution in lecture. E: cannot be used because subproblems are not independent.

Record your answers in the bubbles next to each question.

6. Hash to the Rescue

Ⓐ Ⓑ **Ⓒ** Ⓓ Ⓔ

For which of the following tasks would the use of a hash-based container (e.g., `unordered_map` or `unordered_set`) improve the average-case asymptotic runtime complexity (compared to using a container that does not employ hashing)? You may assume that only STL-provided containers are used.

A) Counting how many times each lower-case letter appears in a newspaper article

B) Finding the k^{th} smallest item in a vector

C) Printing out the first word that appears twice in a text file

D) Printing out all words greater than "apple" and less than "grape" in a document

E) More than one of the above

A: convert each letter to an integer (see problem 4), and use that as an index into an array or vector, no hash table needed. B: hash tabls are useless for k-th largest. C: if this word is already in the hash table, we've found a duplicate, else add it. D: hash tables are unordered (by human standards).

7. BST Construction Zone

Ⓐ Ⓑ **Ⓒ** Ⓓ Ⓔ

Which of the following statements about constructing a binary search tree is **FALSE**?

A) Given a sorted range of n elements, the optimal algorithm can construct a binary search tree in $\Theta(n)$ time

B) Inserting n elements in sorted order would require $\Theta(2^n)$ space if the binary search tree is implemented using a vector

C) Given a sequence of n elements, randomizing the insertion order of these elements will guarantee that the maximum depth of the resulting BST is strictly less than n

D) If n elements are inserted in sorted order into a pointer-based binary search tree, then the total time complexity of deleting all of the elements in the order of insertion is $\Theta(n)$

E) Given a sequence of n distinct elements, it is possible for multiple insertion orders of these elements to produce the same binary search tree

A: start at the middle, this is the root; recurse for left half and right half ($T(n) = 2T(n/2) + c$). B: used as example in lecture. C: in the average case it is height $O(\log n)$, but there is no guarantee that a random order won't be increasing or decreasing, it's just unlikely. D: you get a stick going right, then each deletion is a 1-child case at the current root. E: consider $2, 1, 3$ and $2, 3, 1$.

8. Backtracking

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

For which of the following problems is backtracking most applicable?

A) Finding the shortest path between two vertices in a graph

B) Finding a solution to a puzzle where all rows and columns must be filled with unique numbers, given some starting subset of cells with fixed values

C) Finding a given element of a recursively defined integer sequence

D) Finding the combination of indivisible items with given values and weights that maximizes total value within a certain weight capacity

E) Finding the fewest number of coins that can be dispensed for a given amount of change

A: backtracking doesn't optimize. B: constraint satisfaction, use backtracking (sounds like Sudoku). C: an example would be the Fibonacci numbers; there are no constraints, so no need to backtrack. D and E: same reasoning as A.

Record your answers in the bubbles next to each question.

9. Ultrasuperdynamic Programming

Ⓐ Ⓑ Ⓒ **Ⓓ** Ⓔ

Solutions A and B show valid functions that compute the n^{th} Fibonacci number (starting with the 0^{th} number in the sequence: 0, 1, 1, 2, 3, 5, 8...). Which of the following statements is **TRUE**?

Solution A

```
1   int fibonacci_A(int n) {
2     if (n <= 1)
3       return n;
4     int last1 = 0;
5     int last2 = 1;
6     for (int i = 0; i < n; ++i) {
7       if (last1 < last2)
8         last1 += last2;
9       else
10        last2 += last1;
11    } // for
12    return max(last1, last2);
13  } // fibonacci_A()
```

Solution B

```
1   int helper(int n,
2              vector<int> &table) {
3     if (n <= 1)
4       return n;
5     else if (table[n] != 0)
6       return table[n];
7     else
8       table[n] = helper(n - 1, table)
9         + helper(n - 2, table);
10    return table[n];
11  } // helper()
12
13  int fibonacci_B(int n) {
14    vector<int> table(n + 1);
15    return helper(n, table);
16  } // fibonacci_B()
```

A) Solution A does **NOT** use dynamic programming

B) Solution A uses a top down approach

C) Solution B uses a bottom up approach

D) Solutions A and B have equal runtime complexities

E) More than one of the above

A: a clever solution that retains the two most recent values instead of recalculating them. B: starts from the two smallest values and calculates every succssive value; this is bottom-up. C: checks the memo, calls recursively if needed on a slightly smaller value, stores result and returns; this is top-down. D: they are both $O(n)$.

10. Maximum Inserting

Ⓐ Ⓑ Ⓒ Ⓓ **Ⓔ**

Suppose you are given a balanced AVL tree of real numbers ($\mathbb{R}$) that currently contains n nodes. If you are allowed to choose the values and insertion order, what is the maximum number of new nodes that you can insert into this AVL tree before a rotation is required to rebalance the tree? *Hint: This is a theoretical tree, not an implementation of one.*

A) 1

B) 2

C) $\log(n)$

D) n

E) ∞

If you pick the insertion points carefully, you can keep it balanced at all times, and never require a rotation. The hint about it being theoretical means that although there are an infinite number of real numbers, a `double` can hold an extremely large but finite number of different values.

Record your answers in the bubbles next to each question.

11. Algorithm Families

(A) (B) (C) (D) (E)

Which of the following is a **FALSE** statement regarding algorithm families?

A) A divide and conquer strategy can always use memoization to reduce runtime at the cost of increased memory usage

B) An algorithm that returns the first solution to satisfy all constraints does not guarantee an optimal solution

C) A greedy algorithm will not always return the globally-optimal solution because it never reconsiders decisions that have already been made

D) Combine and conquer is an algorithmic strategy that solves a problem from the bottom up

E) None of the above

A: divide and conquer is for problems that have independent subproblems; DP (and a memo) is for overlapping and repeated subproblems.

12. Hash Table with String Keys

(A) (B) (C) (D) (E)

Consider a hash table with string keys. Which of the following is the best strategy for computing an integer hash from a string? Note that both 7 and 28183 are prime numbers.

A) Add up the character values in the string

B) At each character, add the character value plus (7 * the current running total) to the total. Then set total = (total % 28183)

C) Multiply all the character values in the string. Then set total = (total % 28183)

D) At each character, add the character value plus (10 * the current running total) to the total. Then set total = (total % 50000)

E) At each character, add the character value plus (7 * the current running total) to the total

A: it works, but hashes "spot" and "pots" to the same value. B: a decent choice, though a value larger than 7 would work better (closer to the number of different characters). C: same problem as A. D: since 10 and 50000 are not prime, this is not as good of a choice. E: choice B is better due to the second prime number in the modulus.

Record your answers in the bubbles next to each question.

13. Pruning

(A) (B) (C) **(D)** (E)

You are implementing a program that solves the traveling salesperson problem using a branch and bound strategy, and you need to select a heuristic for estimating the remaining distance in a candidate tour. Which of the following heuristics would allow your program to find the optimal TSP tour most quickly?

A) Calculate the total distance of all permutations of the remaining vertices and return the shortest distance found

B) Find and return the longest pair-wise distance among the remaining vertices

C) Find the shortest pair-wise distance among the remaining vertices and return this distance multiplied by the total number of vertices in the graph

D) Calculate the weight of the MST of the remaining vertices and return half of this distance

E) Ignore the remaining points and always return 0

A: this is TSP, and is too time-complex for an estimate. B: smaller than reality, but so much smaller that it won't prune enough. C: if this was multiplied by (the number of remaining vertices minus one) it would be valid probably too small to prune well; as-is, it is probably longer than reality. D: decent though not dividing by half would prune more. E: valid but never prunes.

14. Always be PRE-pared

(A) (B) **(C)** (D) (E)

Given a binary tree with the following inorder and postorder traversals, what is the preorder traversal?

Inorder: 7, 2, 45, 29, 21, 5, 36, 16, 12, 100

Postorder: 2, 7, 45, 21, 36, 5, 16, 100, 12, 29

A) 2, 5, 7, 12, 16, 21, 29, 36, 45, 100

B) 7, 5, 29, 12, 100, 16, 36, 21, 45, 2

C) 29, 45, 7, 2, 12, 16, 5, 21, 36, 100

D) 29, 45, 7, 2, 12, 16, 5, 100, 21, 36

E) 100, 16, 36, 21, 45, 2, 7, 29, 5, 12

Similar to lecture, where we used the preorder and inorder traversals to reconstruct the original tree. Since 29 is last in the postorder traversal it must be the root. Everything in the inorder traversal that precedes 29 is in 29's left subtree, everything after 29 is in its right subtree; apply recursively. Once you have the original tree, run a preorder traversal of it.

15. Find the Successor

(A) (B) **(C)** (D) (E)

What is the inorder successor of the root node after inserting the following values (in order) into an initially-empty AVL tree?

```
5, 7, 9, 12, 19, 21, 25
```

A) 9

B) 12

C) 19

D) 21

E) 25

Run through the insertions, rotating as necessary (5, 7, 9; RL(5); 12, 19, etc.). AFTER you've solved it by hand, use `https://visualgo.net/en/bst` and be sure to switch it to AVL Tree at the top.

Record your answers in the bubbles next to each question.

16. Don't Be Greedy!

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

Given a knapsack with a weight capacity of 60 kg, which of the following lists of items will cause a **ratio-based greedy** 0-1 Knapsack solver to produce a suboptimal solution? Note that each item listed may be included in the knapsack at most once.

A) ($1, 5 kg), ($5, 7 kg), ($20, 20 kg), ($40, 40 kg)

B) ($70, 2 kg), ($60, 30 kg), ($61, 40 kg), ($120, 60 kg)

C) ($70, 2 kg), ($50, 30 kg), ($60, 30 kg), ($120, 60 kg)

D) ($10, 1 kg), ($10, 15 kg), ($400, 25 kg), ($30, 30 kg)

E) ($4, 2 kg), ($8, 4 kg), ($16, 8 kg), ($32, 16 kg)

Process each set of data through the algorithm, and see which one produces a value that is lower than a value that you can produce yourself.

17. Branching Out

Ⓐ Ⓑ Ⓒ **Ⓓ** Ⓔ

How many rotations are needed to perform the following operations on an initially empty AVL tree? Note that a double rotation operation counts as two rotations. Deletions replace the node to be removed with the inorder successor.

Insert `11`, Insert `43`, Insert `24`, Insert `34`, Delete `11`, Insert `19`, Insert `8`, Insert `2`

A) 3

B) 4

C) 5

D) 6

E) 7

Process the insertions, rotating as necessary; check your work as in Question 15.

Record your answers in the bubbles next to each question.

18. Modify Dijkstra

(A) (B) (C) (D) (E)

Which of the following statements is/are **TRUE**?

I. Adding the same, constant amount of weight to every edge of a positively-weighted, directed graph will not change the shortest path returned by Dijkstra's algorithm for any two points in the graph

II. Given a graph with all negative edge weights, Dijkstra's algorithm can be modified to find the most-negative route between any two vertices in the graph (-9 is *more negative* than -2)

III. Given a graph with all negative edge weights, Dijkstra's algorithm can be modified to find the least-negative route between any two vertices in the graph

A) I only

B) II only

C) III only

D) II and III only

E) I, II, and III

Start finding counter-examples. For I, consider edges and weights A-B=3, B-C=3, C-D = 3, A-X=5, X-D=5. The shortest path from A-D is weight 9, path A-B-C-D. However, if we add 2 to every edge weight, the shortest path is now A-X-D at weight 14. For II, consider A-B=-2, B-C=-2, A-X=-1, X-D=-10. If we're picking the *more negative* first, B gets marked true (distance -2), then C is marked true (distance -4), then X (distance -1). But now we've found a better way to reach C at weight -11, but it's already marked true, and as a greedy algorithm Dijkstra is not allowed to change it's mind.

19. Go the Distance

(A) (B) (C) (D) (E)

You are given a graph, which is represented using the distance matrix below. How many distinct minimum spanning trees can be built using this graph, and what is the weight of the MST?

	A	B	C	D	E	F	G
A	∞	1	4	∞	5	1	7
B	1	∞	2	1	6	2	∞
C	4	2	∞	3	∞	∞	∞
D	∞	1	3	∞	4	∞	∞
E	5	6	∞	4	∞	2	4
F	1	2	∞	∞	2	∞	7
G	7	∞	∞	∞	4	7	∞

A) Weight of MST = 9, only one distinct MST can be built

B) Weight of MST = 10, only one distinct MST can be built

C) Weight of MST = 11, only one distinct MST can be built

D) Weight of MST = 10, more than one distinct MST can be built

E) Weight of MST = 11, more than one distinct MST can be built

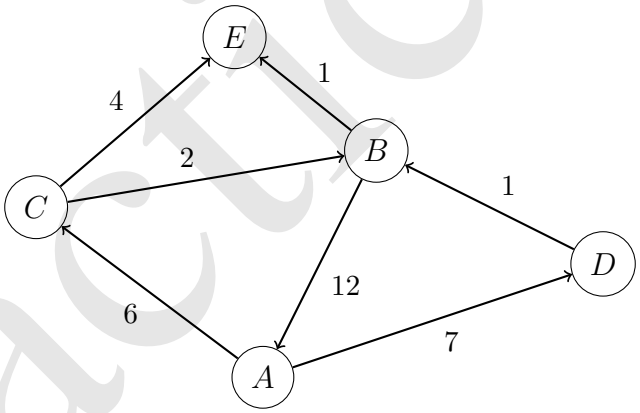
The simplest way to do this is run through using Kruskal, circle each edge that you use and cross out its duplicate (if you choose edge AB weight, cross out BA), and cross out any edge that you cannot use because it would produce a cycle. Add up the circled numbers for the total weight. If you ever had a choice between two different equal-weight edges and could only pick one, then there would be more than one MST possible. Add up the circled numbers for the total weight.

Record your answers in the bubbles next to each question.

20. Compute a Path

(A) (B) (C) (D) (E)

Consider the following directed graph. What path does Dijkstra's algorithm discover from A to E, and what is its weight? Only update predecessors for weights that are improvements.



A) Weight 9, Path: ACBE

B) Weight 9, Path: ADBE

C) Weight 10: Path ACE

D) Either A or B

E) Either A, B or C

Run Dijkstra's algorithm on this graph, starting at A (distance 0). Finding the smallest false vertex yields A, mark it as true. Update the neighbors: C (distance 6), and D (distance 7). Back through the outer loop, dind the smallest false vertex (C), mark it as true, update neighbors (B distance 8 and E distance 10)...

21. Proffice Hours Overrun

(A) (B) (C) (D) (E)

Suppose you are given a list of N students, $\{s_1, s_2, \ldots, s_N\}$, waiting for help in office hours. Each value in the list represents the amount of time needed for an instructor to answer the student's questions. Which algorithmic technique can be used to find the optimal ordering of students to minimize the combined wait time for all students using the least amount of time and space? Assume that there is only one instructor.

A) Brute force

B) Dynamic Programming

C) Backtracking

D) Branch and Bound

E) Greedy

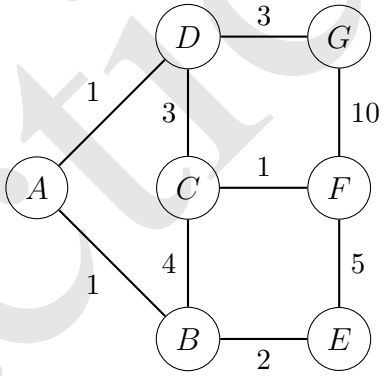
Greedy based on the only known value (time needed to help that student) will minimize overall wait times. Greedy will be optimal here because, at any choice, any larger choice adds a larger value to every remaining students' wait time.

Record your answers in the bubbles next to each question.

22. Too Many Edges

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

Consider the following graph. When using Kruskal's algorithm to compute the minimum spanning tree, what is a valid order for adding edges to the MST?



A) (A,D), (A,B), (C,F), (D,G), (C,D), (B,E)

B) (C,F), (A,B), (A,D), (B,E), (C,D), (D,G)

C) (C,F), (C,D), (A,D), (A,B), (B,E), (D,G)

D) (A,B), (A,D), (B,E), (C,D), (C,F), (D,G)

E) None of the above

(A,B), (A,D) or (C,F) must be the first three (all three, since they don't produce a cycle), which eliminates choice D. (B,E) must come next (it doesn't produce a cycle), which eliminates choices A and C. Both (C,D) and (D,G) have a weight of 3, and selecting both does not produce a cycle.

23. Longest Arithmetic Subsequence

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

You are given a vector of integers, and you want to return the length of the longest arithmetic subsequence in this vector. An arithmetic sequence is a sequence $\{v_0, v_1, \ldots, v_n\}$ where the value of $v_{i+1} - v_i$ is the same for all values $0 \leqslant i \leqslant n-1$. For example, given the vector

```
{281, 370, 381, 181, 470, 481, 482, 485}
```

the longest arithmetic subsequence is `{281, 381, 481}`, which has a length of 3. What is the time complexity of completing this task if you use the most time-efficient algorithm?

A) $\Theta(n)$

B) $\Theta(n^2)$

C) $\Theta(n^2 \log n)$

D) $\Theta(n^3)$

E) $\Theta(2^n)$

This is a DP problem that requires a square matrix (half of which gets filled in). Look up the name of the problem for a longer explanation.

Record your answers in the bubbles next to each question.

24. Mystery Tree Function

Ⓐ Ⓑ Ⓒ Ⓓ ⓔ

Consider the following snippet of code. `INT_MIN` and `INT_MAX` represent the smallest and largest integers that the `int` type can store, respectively.

```
1   struct TreeNode {
2     int value;            // value of the node
3     TreeNode* left;       // pointer to the left child node
4     TreeNode* right;      // pointer to the right child node
5   };
6
7   bool helper(TreeNode* root, int lower, int upper) {
8     if (root == nullptr)
9       return true;
10    if (root->value < lower || root->value > upper)
11      return false;
12    else
13      return    helper(root->left, lower, root->value)
14             && helper(root->right, root->value, upper);
15  } // helper()
16
17  bool mystery_function(TreeNode* root) {
18    return helper(root, INT_MIN, INT_MAX);
19  } // mystery_function()
```

If given the root node of a tree, what does `mystery_function()` do?

A) The function returns **`true`** if and only if all values in the tree are between `INT_MIN` and `INT_MAX`

B) The function returns **`true`** if and only if all left children in the tree have values that are one less than the value of their parent

C) The function returns **`true`** if and only if the tree is balanced

D) The function returns **`true`** if and only if there are no duplicate values in the tree

E) The function returns **`true`** if and only if the tree is a valid binary search tree

At each recursive call, it determines if the left subtree has values from `lower` to `root->value`, and everything in the right subtree has values from `root->value` to `upper`.

Written Portion [40 points]

WRITTEN PORTION INSTRUCTIONS:

- Please write your code legibly in the space provided. Solutions that are difficult to read may receive fewer points.
- Use the back of the question page to work out solutions for each problem and then rewrite your final answer in the space provided.
- The directions for each problem will specify which STL algorithms/functions may be used in your solution.
- Solutions that exceed the allowed line count will lose one (1) point per line over the limit.
- Partial credit will be awarded for partial solutions.
- Errors in program logic or syntax may result in a reduced score.
- Be sure to consider all possible sources of memory (stack and heap) and time complexity.

---

25. Dynamic Programming: Zero-Sum Game [18 Points]

Write a function that, given a vector of integers, returns **`true`** if the vector contains a *contiguous* subsequence that sums to 0 and **`false`** otherwise. Note that your function should return **`false`** for an empty vector.

For example, if `vector<int> a = {7, 3, 5, -9, 1}`, calling `zero_contiguous_sum(a)` would return **`true`** because the contiguous subsequence `{3, 5, -9, 1}` sums to zero.

Similarly, if `vector<int> b = {3, -2, 4, 2, -3}`, calling `zero_contiguous_sum(b)` would return **`false`** because no contiguous subsequence in `b` sums to zero.

Complexity: $O(n)$ average-case time and $O(n)$ space, where n is the number of elements in the vector.

Implementation: Use the back of this page as a working area, then rewrite **neatly** on the front.
Limit: 15 lines of code (points deducted if longer). You **MAY** use anything in the STL.

```
bool zero_contiguous_sum(vector<int> &nums) {

  unordered_set<int> sums;
  int curr_sum = 0;
  sums.insert(curr_sum);
  for (const auto &num : nums) {
    curr_sum += num;
    if (sums.find(curr_sum) != sums.end()) {
      return true;
    } // if
    sums.insert(curr_sum);
  } // for
  return false;
} // zero_contiguous_sum()
```

This page is intentionally left blank.
You may use this page as working space.

26. Programming: Range Queries [22 Points]

You are given two vectors of input, `data` and `queries`. The `data` vector contains **N** unsigned integers, and the `queries` vector contains **Q** query structs. Query structs are defined as follows:

```
struct Query { unsigned int id, start, end; };
```

Write a function that prints the answers to all queries on a single line. An answer to a query is the number of times in the `data` vector that `id` appears in the range $[\texttt{start}, \texttt{end}]$, **inclusive**.

For example, given these **`data`** and **`queries`** vectors, your code should produce the output shown:
`data`: `{5, 4, 3, 5, 3, 3, 2, 1, 3}`
`queries`: `{ {4, 7, 8}, {3, 0, 3}, {3, 0, 8}, {7, 0, 8}, {5, 0, 4} }`
Output: `0 1 4 0 2`

Explanation: The first element of the result vector is `0` because `4` never occurs between indices `7` and `8` (inclusive) in `data`. Similary, the second element of the result vector is `1` because `3` occurs once between indices `0` and `3` (inclusive). Notice that queried items may not appear in the data at all.

Complexity: $O(N^2 + Q)$ average-case time and $O(N^2)$ space.

Implementation: Use the back of this page as a working area, then rewrite neatly on the front.
Limit: 25 lines of code (points deducted if longer). You MAY use anything in the STL.
Hint: You may want to preprocess the data before answering queries.

```
void range_queries(const vector<unsigned int> &data,
                   const vector<Query> &queries) {

  //Step 1: preprocessing
  std::unordered_map<unsigned int, std::vector<unsigned int>> ranges;

  unsigned int index = 0;
  //Create 0-N data for each item in data
  for (auto d : data) {
    //Ensure that the data has an appropriate size range-vector
    auto &vec = ranges[d];
    if (vec.empty())
      vec.resize(data.size(), 0);
    //Extend all ranges
    for (auto &r : ranges)
      r.second[index] = index ? r.second[index - 1] : 0;
    //Increment the range for d's corresponding vector
    ++vec[index++];
  }

  //Step 2: answering queries
  //Main idea: if we know # times element within [0, A] and [0, B] then
  //the number between [A, B] is the difference
  for (auto &q : queries) {
    auto &vec = ranges[q.id];
    if (vec.empty())
      cout << "0 ";
    else {
      unsigned int A = q.start == 0 ? 0 : vec[q.start - 1];
      unsigned int B = vec[q.end];
      cout << B - A << " ";
    }
```

```
  }
  cout << endl;
}
```