CS 61BL Summer 2026

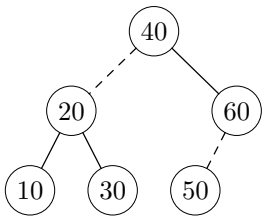
Final Exam Review

Exam Prep 05: August 7, 2026

1 LLRBs

from Fall 2022 Midterm 2

Consider the below integer LLRB. Like typical LLRBs, assume all values are unique.



(a) Draw the corresponding 2-3 tree.

(b) Give all integers which, when inserted into the LLRB, result in a single `rotateLeft` operation and no further cascading operations (i.e. no `rotateLeft`s, `rotateRight`s or `colorFlip`s). If this is not possible, write "impossible".

If it is possible, use **inclusive square bracket notation**, e.g. if your answer is "`[62, 64]` and `[66, 68]`", this is equivalent to saying 62, 63, 64, 66, 67, 68.

(c) Give all integers which, when inserted into the LLRB results in a single `rotateRight` operation and no further cascading operations (i.e. no `rotateLeft`s, `rotateRight`s or `colorFlip`s). If this is not possible, write "impossible". If it is possible, use inclusive square bracket notation.

(d) At most, how many additional inserts can be performed into the LLRB above without triggering any `colorFlip` operations? Do not count the operation that results in the first color flip.

2 Bob the Shopkeeper

from Summer 2025 Final

For this question, all subparts are independent and the code compiles.

For each subpart, mark whether the given object's **`hashCode`** implementation is valid. Then, write the best case and worst case runtimes for the given scenario in the form $\Theta(\cdot)$.

You may assume that Java's **`HashSet`** implementation:

- Uses external chaining with linked list buckets
- Has an initial array capacity of 8
- Resizes by doubling the array capacity with a load factor threshold of 0.5

(a)
```
class Vi {} // intentionally empty
```

Is **`Vi`**'s **`hashCode`** valid? ◯ Yes ◯ No

(b) In total, what is the best case and worst case amortized runtime (in terms of N) of inserting N new **`Vi`** objects into an empty **`HashSet`**?

Best Case: ☐ Worst Case: ☐

(c)
```
class Jinx {
    @Override
    public int hashCode() {
        return -982347079;
    }
}
```

Is **`Jinx`**'s **`hashCode`** valid? ◯ Yes ◯ No

(d) In total, what is the best case and worst case amortized runtime (in terms of N) of inserting N new **`Jinx`** objects into an empty **`HashSet`**?

Best Case: ☐ Worst Case: ☐

(e)
```
class Caitlyn {
    private static int count = 0;
    public Caitlyn() {
        count += 1;
    }

    @Override
    public boolean equals(Object o) {
        quadratic(count);
        return true;
    }
}
```

Assume `quadratic(N)` runs in $\Theta(N^2)$ time.

Is `Caitlyn`'s `hashCode` valid? ◯ Yes ◯ No

(f) We create N new `Caitlyn` objects. After all N `Caitlyn` objects are created, we insert each of them into an empty `HashSet`.

In total, what is the best case and worst case amortized runtime (in terms of N) for this entire process?

Best Case: ☐ Worst Case: ☐

3 Borbders

from Fall 2025 Midterm 2

Bob loves walking between countries, which are represented by the class below:

```
public class Country {
  public String name; // Name of the country.
  public List<Country> borders; // A list of bordering countries. Is never null.
}
```

Borders go both ways. If countryA appears in countryB's borders, then countryB appears in countryA's borders.

Implement the `WalkManager` class, which has two methods:

- `public WalkManager(List<Country> countries)`:

  Constructor. Given the list of all N countries in the world, prepare any useful instance variables. Must run in $O(N^2 \log N)$ time.

- `public boolean canWalk(Country a, Country b)`:

  Returns whether country `a` is reachable from country `b`, using **one or more borders**. Must run in $O(\log N)$ time.

Here is an example in a world that has $N = 4$ countries:

- `USA` borders [`Canada`, `Mexico`].
- `Canada` borders [`USA`].
- `Mexico` borders [`USA`].
- `Ethanland` borders [] (empty list).

```
WalkManager wm = new WalkManager(List.of(USA, Canada, Mexico, Ethanland));
```

`wm.canWalk(Canada, Mexico)`; Returns `true`: `Canada` to `USA` to `Mexico`.

`wm.canWalk(Ethanland, USA)`; Returns `false`: No sequence of borders exists between `Ethanland` and `USA`.

The skeleton code begins on the next page.

Note: You may use the `WeightedQuickUnion` class from `algs4`. As a reminder, it has the following methods:

- `WeightedQuickUnion(int n)`: Constructor. Initializes a new `WeightedQuickUnion` with size `n`.
- `union(int a, int b)`: Unions the sets that `a` and `b` belong to.
- `isConnected(int a, int b)`: Checks if `a` and `b` are in the same set.

Implement WalkManager according to Bob's requirements.

```
public class WalkManager {
  // Add any instance variables below (if necessary)
  _____;
  _____;

  public WalkManager(List<Country> countries) {
    int N = countries.size();
    _____;
    _____;

    for (int i = 0; i < _____; i += 1) {
      Country curr = _____;
      _____;
    }

    for (_____) {
      for (Country other : _____) {
        _____
      }
    }
  }

  public boolean canWalk(Country a, Country b) {
    return _____;
  }
}
```

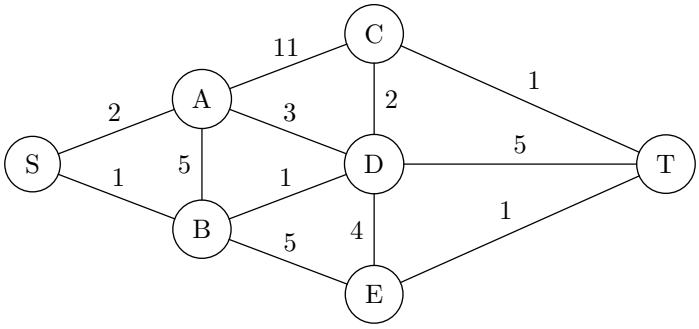
4 Lost in the Ceryneian H.I.N.D.

from Spring 2024 Final

After being repeatedly hacked in past exams, you've finally tracked down the hackers responsible as members of the Ceryneian H.I.N.D. With the assistance of an investigator (codenamed Heracles), you have managed to infiltrate their internal network. Help Heracles catch the Ceryneian H.I.N.D.

The internal network is a **weighted, undirected graph**, with start vertex S and target vertex T. You want to delete one edge from the graph, such that when deleted, the shortest path length from S to T is increased by as much as possible.

For example, consider the graph below:



Before deleting an edge, the shortest path from S to T has length 5. If we delete edge BS, the shortest path from S to T will instead have length 8; deleting any other edge will result in a shortest path from S to T that is shorter than 8. Thus, we should delete edge BS.

Design an algorithm (in English or pseudocode) that, given a graph, returns the edge that should be deleted. If multiple edges yield the same maximal shortest-path length, your algorithm may return any of them. You may assume that:

- All edges have positive integer edge weights.
- $S \neq T$
- The input graph is connected, and cannot be disconnected by removing any one edge.
- You have access to a working black-box implementation of Dijkstra's algorithm, which runs in $\Theta(E \log V)$ time.

For credit, your algorithm must run in $O(VE \log(V))$ time, where V and E are the number of vertices and edges in the graph, respectively.

Hint: Try to devise a naive solution first, then optimize that solution down to the specified runtime bound.

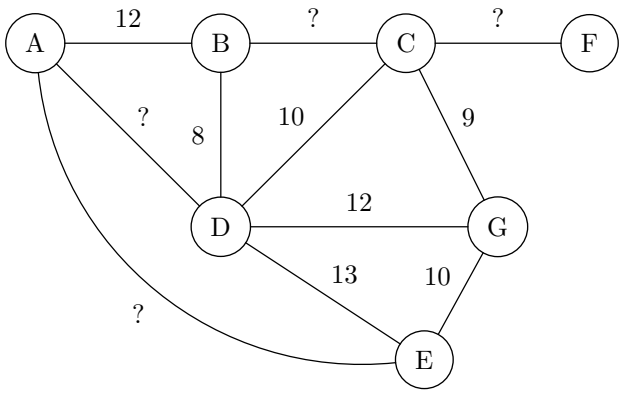
5 A Very Kapeeshable MST

from Summer 2023 Final

Omar needs to save the endangered monkfish, monkdogs, and monkbears that are scattered across the world at various locations. As a result, he creates a weighted, undirected graph G where vertices represent locations, edges represent routes, and edge weights correspond to the cost of taking a route. He wants to find an MST in G that will allow him to visit all the locations with the cheapest total cost.

However, his internet goes out before he can finish writing down all the costs of the routes!

Consider the graph G below:



(a) Choose all edges that must exist in every possible MST of G. Edges are represented as unordered sets, i.e. `{A, B}` is equivalent to `{B, A}`.

☐ `{A, B}` ☐ `{A, E}` ☐ `{B, D}` ☐ `{C, F}` ☐ `{D, E}` ☐ `{E, G}`

☐ `{A, D}` ☐ `{B, C}` ☐ `{C, D}` ☐ `{C, G}` ☐ `{D, G}`

(b) Teresa wants to create a new graph M where the number of possible cuts in M is equal to the number of edges in an MST of M. Fill in the blanks with the correct numbers (non-negative integers) to create this graph. Assume the created graph is connected, and has at least one vertex and one edge.

The number of vertices in the new graph must be: _____

The number of edges in the new graph must be: _____

6 Frankensort

from Summer 2024 Final

Dominic is experimenting with how he can use different sorts inside of quicksort.

Quicksort:

- *Partition the array around a chosen pivot element p*
- *Quicksort the left subarray*
- *Quicksort the right subarray*

Below are versions of quicksort Dominic is considering:

Heap-Quicksort: Modify the partitioning algorithm to mimic heapsort. Choose the leftmost element to be p. Perform in-place heapsort on the array. Terminate partitioning after popping off and inserting p in heapsort.

Insertion-Quicksort: Modify the partitioning algorithm to mimic insertion sort. Choose the center element in the array to be p (center-left if length is even). Perform insertion sort on the array. Terminate partitioning after you finish swapping p as far left as it can go in insertion sort.

MSD-Quicksort: Modify the partitioning algorithm to mimic MSD radix sort. Choose the leftmost element to be p. Stably partition the array by the most significant digit, creating 3 buckets in the following order: less than, equal to, and greater than the pivot (when comparing only that digit). Partitioning is complete when the bucketing is finished. Recursively quicksort the left and right buckets by the same digit. Recursively quicksort the center bucket by the next most significant digit.

For each sequence of intermediate states of an array during quicksort, select all of the algorithms from above which could have produced the sequence. Note that these states are not necessarily the first few intermediate states, and there may be states which are skipped. Assume the first state is the array before sorting has begun. Each sequence is valid - none are impossible.

(a)
```
61 52 83 27 38 49 74 19 67
52 38 49 19 27 61 67 74 83
49 38 27 19 52 61 67 74 83
```

☐ Heap-Quicksort ☐ Insertion-Quicksort ☐ MSD-Quicksort

(b)
```
56 22 35 43 49 78 67 84 91
22 35 43 49 56 78 67 84 91
22 35 43 49 56 67 78 84 91
```

☐ Heap-Quicksort ☐ Insertion-Quicksort ☐ MSD-Quicksort

(c) Which of the below algorithms are guaranteed to be correct?

☐ Heap-Quicksort ☐ Insertion-Quicksort ☐ MSD-Quicksort

(d) Which of the below algorithms are stable? Consider an algorithm stable based on its output, even if the output is incorrect.

☐ Heap-Quicksort ☐ Insertion-Quicksort ☐ MSD-Quicksort

7 Flip Flop (Extra Practice, Recommended)

Suppose we have the flip function as defined below. Assume the method unknown returns a random integer between 1 and N, exclusive, and runs in constant time. For each definition of the flop method below, give the best and worst case runtime of flip in big Theta notation as a function of N.

```
public static void flip(int N) {
  if (N <= 100) {
    return;
  }
  int stop = unknown(N);
  for (int i = 1; i < N; i++) {
    if (i == stop) {
      flop(i, N);
      return;
    }
  }
}
```

(a)
```
public static void flop(int i, int N) {
  flip(N - i);
}
```

Best case: _____	Worst case: _____

(b)
```
public static void flop(int i, int N) {
  int minimum = Math.min(i, N - i);
  flip(minimum);
  flip(minimum);
}
```

Best case: _____	Worst case: _____

(c)
```
public static void flop(int i, int N) {
  flip(i);
  flip(N - i);
}
```

Best case: _____	Worst case: _____