Section 10: Finals review

1. Finding closed forms

For each of the following, find a closed form using the tree method and a big-Theta bound using the master method. If you cannot use the master method on a particular recurrence, briefly explain why.

See the last page of this handout for a list of identities.

For each of these problems, we recommend you go through the following steps:

(a) Draw levels 0,1,2, and the last level of the tree.

(b) Find a formula for the input size at level i. Your formula will depend on i (the level) and n (the initial input to the function).

(c) Find the number of non-recursive levels

(d) Find a formula for the number of nodes at level i.

(e) Combine the last two parts to find the work done at some recursive level i.

(f) Find the work done at the base case.

(g) Combine (e) and (f) to find a final closed form for the total amount of work.

The following are sample recurrences, but almost any previous recurrence can be worked on using the previous steps.

(a) $A(x) = \begin{cases} 1 & \text{if } x = 1 \\ 3A(\frac{x}{3}) + 3x^2 & \text{otherwise} \end{cases}$

(b) $B(n) = \begin{cases} 1 & \text{if } n = 1 \\ 8B(\frac{n}{3}) + 2n & \text{otherwise} \end{cases}$

(c) $C(x) = \begin{cases} 3 & \text{if } x = 1 \\ 4C(\frac{x}{5}) + x^3 + 2 & \text{otherwise} \end{cases}$

(d) $D(n) = \begin{cases} 3 & \text{if } n = 1 \\ 3D(n-1) + 2 & \text{otherwise} \end{cases}$

2. Memory, locality, and dictionaries

(a) In lecture, we discussed three different optimizations for disjoint sets: union-by-rank, path compression, and the array representation.

If we implement disjoint sets using `Node` objects with a "data" and "parent" field and implement the first two optimizations, our `find` and `union` methods will have a nearly-constant average-case runtime.

In that case, why do we bother with the array representation?

(b) Suppose you implement a dictionary with a sorted array and another with an AVL tree. Consider the time needed to iterate over the key-value pairs of a `SortedArrayDictionary` vs an `AvlDictionary`. It turns out that iterating over the `SortedDictionary` is nearly 10 times faster then iterating over the `AvlDictionary`. Think about why that might be.

Now, suppose we take those same dictionaries and try repeatedly calling the `get(...)` method a few hundred thousand times, picking a different random key each time.

Surprisingly, we no longer see such an extreme difference in performance. The `SortedArrayDictionary` is at most only about twice as fast as the `AvlDictionary`.

Why do you suppose that is? Be sure to discuss both (a) why the difference in performance is much less extreme and (b) why `SortedArrayDictionary` is still a little faster.

3. Heaps

Consider the following list of numbers:

```
[1, 5, 2, -6, 7, 12, 3, -5, 6, 2, 1, 6, 7, 2, 1, -2]
```

(a) Insert these numbers into a min 2-heap (into a min-heap with up to two children per node). Show both the final tree and the array representation.

(b) Insert these numbers into a max 3-heap (nto a max-heap with up to three children per node). Show both the final tree and the array representation.

(c) Insert these numbers into a min 4-heap using Floyd's `buildHeap` algorithm. Show both the final tree and the array representation.

(d) Insert these numbers into a max 2-heap using Floyd's `buildHeap` algorithm. Show both the final tree and the array representation.

(e) Suppose we modify Floyd's `buildHeap` algorithm so we start from the front of the array, iterate forward, and call `percolateDown(...)` on each element. Why is this a bad idea?

4. Sorting

(a) During lecture, we focused on five different sorting algorithms: insertion sort, merge sort, quick sort, selection sort, and heap sort.

For each of these five algorithms, state:

- The best and worst-case runtimes
- Whether the sorting algorithm is stable
- Whether the sorting algorithm is in-place
- Whether the sorting algorithm is a general-purpose one, or if there are any restrictions on how it can or should be used.

(b) Suppose we want to sort an array containing 50 strings. Which of the above four algorithms is the best choice?

(c) Suppose we have an array containing a few hundred elements that is almost entirely sorted, apart from a one or two elements that were swapped with the previous item in the list. Which of the algorithms is the best way of sorting this data?

(d) Suppose we are writing a website named "sort-my-numbers.com" which accepts and sorts numbers uploaded by the user. Keeping in mind that users can be malicious, which of the above algorithms is the best choice?

(e) Suppose we want to sort an array of ints, but also want to minimize the amount of extra memory we want to use as much as possible. Which of the above algorithms is the best choice?

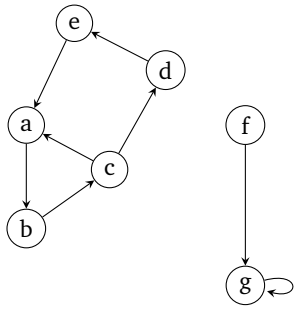
(f) On the homework, you were asked to find a way of building an array that would cause a version of quick sort implemented using the median-of-three pivoting strategy to run in $\mathcal{O}\left(n^2\right)$ time.

Now, find a way of making a version of quick sort implemented using the random pivot selection strategy run in $\mathcal{O}\left(n^2\right)$ time.

(g) How can you modify both versions of quicksort so that they no longer display $\mathcal{O}\left(n^2\right)$ behavior given the same inputs?

5. Graph basics

Consider the following graph:



(a) Draw this graph as an adjacency matrix.

(b) Draw this graph as an adjacency list.

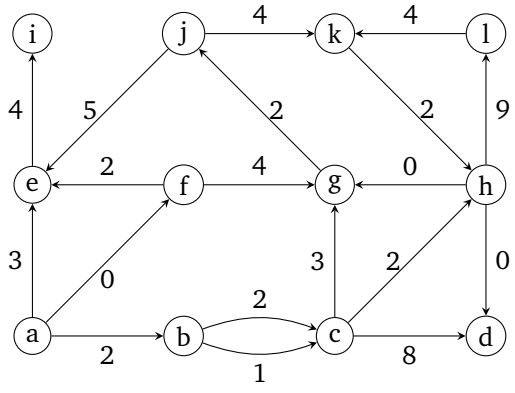
(c) Suppose we run BFS on this list, starting on node a. In what order do we visit each node? Assume we break ties by selecting the node that's alphabetically lower.

(d) Suppose we run DFS on this list, starting on node a. In order do we visit each node? Assume we break ties in the same way as above.

6. Dijkstra's algorithm

For each of the graphs below, list the final costs of each node, the edges selected by Dijkstra's algorithm, and whether or not Dijkstra's algorithm returned the correct result. In the case of ties, select the node that is the smallest alphabetically.

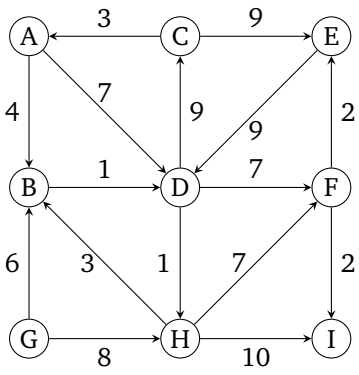
(a) Run Dijkstra's algorithm on the following graph starting on node a.



(b) Can you run Dijkstra's from any node in this graph to get all shortest paths? Why or why not?

(c) More generally, what are some of the reasons why running Dijkstra's might not work correctly?

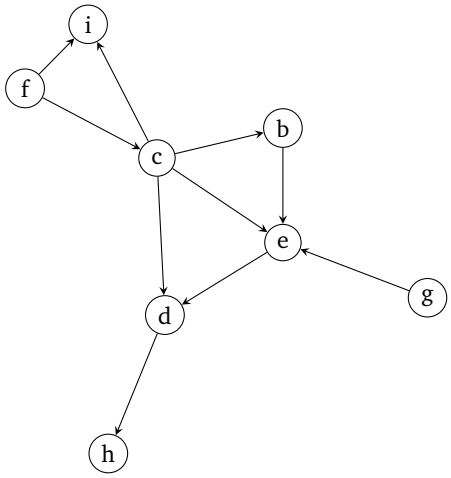
(d) Run Dijkstra's algorithm on the following graph starting on node G.



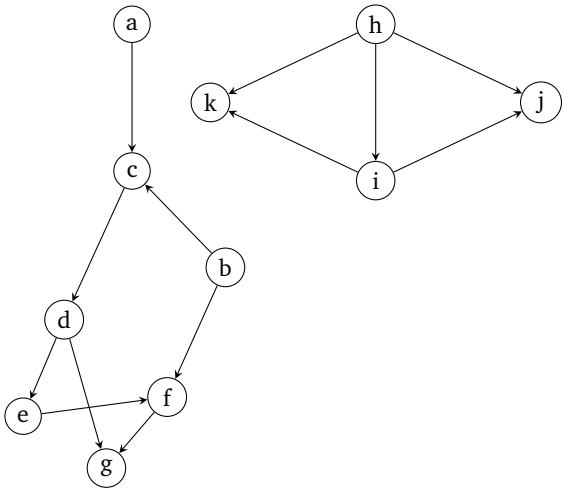
(e) When solving the last problem, you had a choice when picking the shortest path from G to H. Do ties matter in Dijkstra's? Why or why not? If so, provide another set of shortest paths.

7. Topological sort

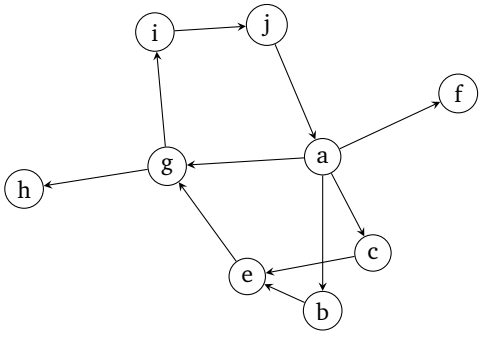
(a) List three different topological orderings of the following graph. If no ordering exists, briefly explain why.



(b) List three different topological orderings of the following graph. If no ordering exists, briefly explain why.

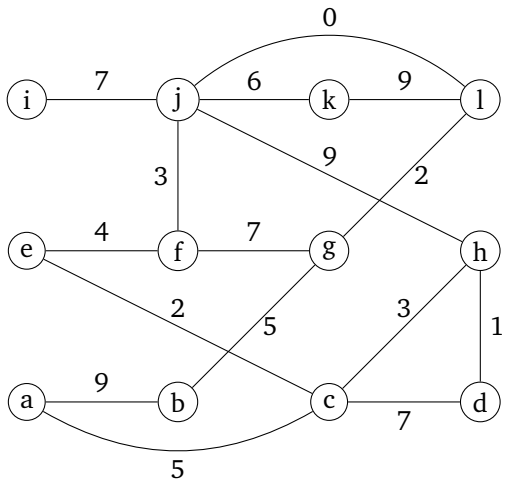


(c) List three different topological orderings of the following graph. If no ordering exists, briefly explain why.



8. Minimum spanning trees

Consider the following graph:



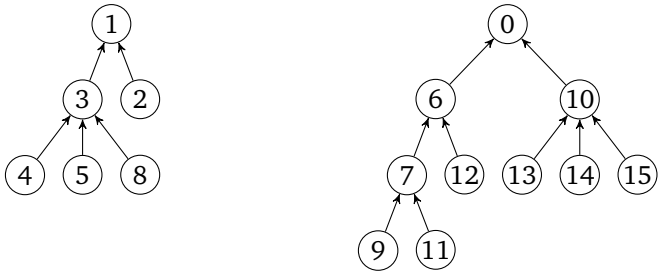
(a) Run Prim's algorithm on the above graph starting on node a to find a minimum spanning tree.

Draw the final MST and the costs per each node. In the case of ties, select the node that is the smallest alphabetically.

(b) Run Kruskal's algorithm on the above graph to find an MST. In the case of ties, select the edge containing the node that is the smallest alphabetically.

Draw the final MST.

(c) Suppose we have the following disjoint set. What happens when we run `union(7, 8)`? Draw both the new trees as well as the array representation of the disjoint set.



(d) Suppose we have a disjoint set containing 16 elements. Assuming our disjoint set implements the union-by-rank and path-compression algorithm, what is the height of the largest possible internal tree we can construct? Draw what this tree looks like, and what sequence of calls to `union(...)` and `findSet(...)` creates this tree.

9. Debugging

Suppose we are in the process of implementing a hash map that uses open addressing and quadratic probing and want to implement the `delete` method.

(a) Consider the following implementation of `delete`. List every bug you can find.

Note: You can assume that the given code compiles. Focus on finding run-time bugs, not compile-time bugs.

```
public class QuadraticProbingHashTable<K, V> {
    private Pair<K, V>[] array;
    private int size;

    private static class Pair<K, V> {
        public K key;
        public V value;
    }

    // Other methods are omitted, but functional.

    /**
     * Deletes the key-value pair associated with the key, and
     * returns the old value.
     *
     * @throws NoSuchKeyException if the key-value pair does not exist in the method.
     */
    public V delete(K key) {
        int index = key.hashCode() % this.array.length;

        int i = 0;
        while (this.array[index] != null && !this.array[index].key.equals(key)) {
            i += 1;
            index = (index + i * i) % this.array.length;
        }

        if (this.array[index] == null) {
            throw new NoSuchKeyException("Key-value pair not in dictionary");
        }

        this.array[index] = null;

        return this.array[index].value;
    }
}
```

(b) Let's suppose the `Pair` array has the following elements (pretend the array fit on one line):

["lily", V_2]	["castle", V_6]	["resource", V_1]	["hard", V_9]	["bathtub", V_0]
["wage", V_4]	["refund", V_7]	["satisfied", V_6]	["spring", V_8]	["spill", V_3]

And, that the following keys have the following hash codes:

Key	Hash Code
"bathtub"	9744
"resource"	4452
"lily"	7410
"spill"	2269
"wage"	8714
"castle"	2900
"satisfied"	9251
"refund"	8105
"spring"	6494
"hard"	9821

What happens when we call `delete` with the following inputs? Be sure write out the resultant array, and to do these method calls *in order*. (**Note**: If a call results in an infinite loop or an error, explain what happened, but don't change the array contents for the next question.)

(i) `delete("lily")`

(ii) `delete("spring")`

(iii) `delete("castle")`

(iv) `delete("bananas")`

(v) `delete(null)`

(c) List four different test cases you would write to test this method. For each test case, be sure to either describe or draw out what the table's internal fields look like, as well as the expected outcome (assuming the `delete` method was implemented correctly). **Hint**: You may use the inputs previously given to help you identify tests, but it's up to you to describe what kind of input they are testing generally.

10. Graphs and design

Consider the following problems, which we can all model and solve as a graph problem.

For each problem, describe (a) what your vertices and edges are and (b) pseudocode or an English description of how you would solve the given problem.

Your description does not need to explain how to implement any of the algorithms we discussed in lecture. However, if you *modify* any of the algorithms we discussed, you must discuss what that modification is.

(a) A popular claim is that if you go to any Wikipedia page and keep clicking on the first link, you will eventually end up at the page about "Philosophy". Suppose you are a given some Wikipedia page as a random starting point. How would you write an algorithm to verify this claim?

(b) Suppose you are given a list of statements about how cities are located relative to each other as input. For example, suppose we had the following statements as input:

- Seattle is north of Portland
- Seattle is west of Spokane
- Portland is south-east of Spokane
- Spokane is west of New York
- Seattle is south of Vancouver

These statements are all internally consistent with each other. Now, suppose we add one more statement to the list:

- Portland is north of Vancouver

If we add this statement to our list, we suddenly have an inconsistency! We previously said Seattle was north of Portland, and that Vancouver was north of Seattle. In that case, it's impossible for Portland to also be north of Vancouver.

How would you write an algorithm to determine whether a given list of statements is consistent or not?

(c) Suppose you have a bunch of computers networked together connected together (haphazardly) using wires. You want to send a message to every other computer as fast as possible. Unfortunately, some wires are being monitored by some shadowy organization that wants to intercept your messages.

After doing some reconnaissance, you were able to assign each wire a "risk factor" indicating the likelihood that wire is being monitored. For example, if a wire has a risk factor of zero, it is extremely unlikely to be monitored; if a wire has a risk factor of 10, it is more likely to be monitored. The smallest possible risk factor is 0; there is no largest possible risk factor.

Implement an algorithm that selects wires to send your message long such that (a) every computer receives the message and (b) you minimize the total risk factor. The total risk factor is define as the sum of the risks of all of the wires you use.

(d) Explain how you would implement an algorithm that uses your predictions to find any computers where sending a message would force you to transmit a message over a wire with a risk factor of k or higher.

(e) Suppose you have a graph containing $|V|$ nodes. What is the maximum number of edges you can add while ensuring the graph is always a DAG? Assume you are not permitted to add parallel edges.

(f) Suppose you were walking in a field and unexpectedly ran into an alien. The alien, startled by your presence, dropped a book, ran into their UFO, and flew off.

This book ended up being a dictionary for the alien language – e.g. a book containing a bunch of alien words, with corresponding alien definitions.

You observe that the alien's language appears to be character based. Naturally, the first and burning question you have is what the alphabetical order of these alien characters are.

For example, in English, the character "a" comes before "b". In the alien language, does the character "ℑ" come come before or after character "ϱ"? The world must know.

Assuming the dictionary is sorted by the alien character ordering, design an algorithm that prints out all plausible alphabetical orderings of the alien characters.

11. P and NP

Consider the following decision problem:

`ODD-CYCLE`: Given some input graph G, does it contain a cycle of odd length?

You want to show this decision problem is in NP.

(a) What is a convincing certificate a solver could return for this problem?

(b) Describe, in pseudocode, how you would implement the verifier.

(c) What is the worst-case runtime of your verifier?

(d) Do you think it's likely this algorithm also happens to be in P? Why or why not?

12. Short answer

For each of the following questions, answer "true", "false", or "unknown" and justify your response. Your justification should be short – at most 2 or 3 sentences.

Note: We will not ask you true/false questions of this type on the exam, but this is a convenient way to remind you of useful facts that don't fit nicely into other problems.

(a) If we implement Kruskal's algorithm using a general-purpose sort, Kruskal's algorithm will run in nearly-constant time.

(b) If we have an efficient way of solving some arbitrary NP problem, we have an efficient way of solving ALL NP problems.

(c) It is possible to reduce all problems in P to some problem in NP.

(d) Dijkstra's algorithm will always return the incorrect result if the graph contains negative-length edges.

(e) Suppose we want to find a MST of some arbitrary graph. If we run Prim's algorithm on any arbitrary node, we will always get back the same result.

(f) $\mathcal{O}\left(n^2 \log(3) + 4\right) = \mathcal{O}\left(4n + n^2\right)$

(g) There is an efficient way of solving the `SAT` decision problem.

(h) Iterating over a list using the iterator is always faster then iterating by repeatedly calling the `get(...)` method.

(i) We can always sort some list of length n in $\Theta(n)$ time.

(j) In a simple graph, if there are $|E|$ edges, the maximum number of possible vertices is $|V| \in \mathcal{O}\left(|E|^2\right)$.

(k) Consider the following question: SHORT-PATH input: Graph G with non-negative edge weights, vertices u, v and a number k output: YES if there is a u, v path and no otherwise.

Is SHORT-PATH in NP? In P?

(l) The `.get(...)` method of hash tables has a worst-case runtime of $\mathcal{O}(n)$, where n is the number of key-value pairs.

(m) A hash table implemented using open addressing is likely to have suboptimal performance when $\lambda > 0.5$.

(n) The `peekMin(...)` method in heaps has a worst-case runtime of $\mathcal{O}(\log(n))$.

(o) If a problem is in NP, that means it must take exponential time to solve.

(p) For any given graph, there exists exactly one unique MST.

Identities

Splitting a sum

$$\sum_{i=a}^{b}(x+y) = \sum_{i=a}^{b} x + \sum_{i=a}^{b} y$$

Gauss's identity

$$\sum_{i=0}^{n-1} i = 0 + 1 + \ldots + n - 1 = \frac{n(n-1)}{2}$$

Factoring out a constant

$$\sum_{i=a}^{b} cf(i) = c\sum_{i=a}^{b} f(i)$$

Finite geometric series

$$\sum_{i=0}^{n-1} r^i = \frac{r^n - 1}{r - 1}$$

Change-of-base identity

$$\log_a(n) = \frac{\log_b(n)}{\log_a(b)}$$

Master theorem

Given a recurrence of the form:

$$T(n) = \begin{cases} d & \text{if } n = 1 \\ aT\left(\frac{n}{b}\right) + n^c & \text{otherwise} \end{cases}$$

We know that:

- If $\log_b(a) < c$ then $T(n) \in \Theta\left(n^c\right)$
- If $\log_b(a) = c$ then $T(n) \in \Theta\left(n^c \log(n)\right)$
- If $\log_b(a) > c$ then $T(n) \in \Theta\left(n^{\log_b(a)}\right)$