CS 2501: DSA2 Quiz Booklet

This booklet contains question pools for the quizzes for this course. You should use this resource to study potential questions that will be asked on the quizzes.

DO NOT write on this booklet, as we may use it multiple times in future weeks.

There are 10 pages to this quiz booklet.

A crash reduces
Your expensive computer
To a simple stone.

Graphs - Basic

Short Answer Questions

1. Given an undirected graph with V nodes (no loops allowed). What is the maximum number of edges the graph can have?
2. Given a directed graph with V nodes (no loops allowed). What is the maximum number of edges the graph can have?
3. Name one advantage and one disadvantage of storing a graph as an adjacency matrix.
4. Name on advantage and one disadvantage of storing a graph as an adjacency list.
5. Describe one way we can handle storing costs of graph edges. Make sure to answer for an adjacency list AND an adjacency matrix.
6. Does breadth-first search always find the shortest path between two nodes for an undirected, unweighted graph?
7. Which has better time-complexity? BFS or DFS?
8. Which has better space-complexity? BFS or DFS?
9. What is the run-time of BFS? Explain your answer.
10. What is the run-time of DFS? Explain your answer.
11. Briefly describe how you might use DFS to count the number of disconnected components (disconnected sub-graphs) in a graph.
12. What is the run-time of topological sort?
13. What is the run-time of Dijkstra's Algorithm?
14. Dijkstra's Algorithm may not work if given negative cost edges. Provide a counter-example to illustrate this.

Coding Questions

1. Psuedo-code a method that performs a breadth-first search on a graph. Assume each node stores a number num. Print out each num as you visit each node.
2. Psuedo-code a method that performs a depth-first search on a graph. Assume each node stores a number num. Print out each num as you visit each node.
3. Pseudo-code the topological sort algorithm.
4. Psuedo-code Dijkstra's Algorithm.

Graphs - Advanced

Short Answer Questions

1. Given a flow network and current flow values, produce the residual graph. A network to use will be drawn on the board.
2. When discussing flow networks, what is backflow? Explain why it is necessary.
3. Describe how the Ford-Fulkerson algorithm uses depth-first search. What purpose does it serve? Why not use breadth-first search?
4. What is the run-time of the Ford-Fulkerson Algorithm? Explain your answer.
5. Define the following terms regarding flow networks: Cut, capacity of a cut, and net-flow of a cut.
6. What does the max-flow, min-cut theorem state? What does this say about algorithms for the two problems?
7. What is a reduction? Why is it useful when comparing algorithms?
8. Describe how you would solve for the maximum-flow of a network that has multiple sources and multiple sinks.

Coding Questions

1. Psuedo-code the Ford-Fulkerson Algorithm.
2. Prove the flow-value lemma: The the net-flow across any cut is always equal to the flow f
3. Psuedo-code an algorithm that solves the Bipartite matching problem.

Find-Union: Prims and Kruskals

Short Answer Questions

1. Give an example of a graph (with edge costs) that has more than one minimum spanning tree.
2. How many edges does a minimum spanning tree have as a function of the original connected graph $G = (V, E)$? Explain why.
3. True or False: If the edge weights of a graph are all unique, then the minimum spanning tree is unique as well. Explain your answer.
4. Given a graph (drawn on the board) and a starting node, step through Prim's algorithm. Draw the MST and list the order in which the nodes are added to the tree.
5. What do we mean when we say that the Minimum Spanning Tree problem has optimal substructure?
6. What is the runtime of Prim's algorithm?
7. What is the runtime of Kruskal's algorithm?
8. Describe how the findSet(int i) method works for a find-union data structure? What is the runtime?
9. Describe how the union(int i, int j) method works for a find-union data structure. What is the runtime?
10. When implementing a find-union structure, what is union by rank? Why is it useful?
11. When implementing a find-union structure, what is path compression? Why is it useful?

Coding Questions

1. Write pseudo-code for Prim's algorithm
2. Write pseudo-code for Kruskal's algorithm
3. Pseudo-code the findSet(int i) method of a find-union structure.
4. Pseudo-code the union(int i, int j) method of a find-union structure.

Greedy Algorithms: Basic

Short Answer Questions

1. What is a *feasible solution*, when discussing optimization problems?
2. What is an *objective function* when discussing optimization problems?
3. Briefly describe why greedy algorithms are typically easy to implement.
4. What was the runtime of the greedy change making algorithm from class? Make sure to explain your variables and explain your answer.
5. When describing greedy algorithms, what is *optimal substructure*? Why is it useful?
6. Describe the greedy algorithm from class for solving the *continuous knapsack problem*.
7. What is the runtime of the greedy algorithm for the *continuous knapsack problem*.
8. Describe the greedy algorithm we used in class for the *interval selection problem*.
9. What was the runtime of the *interval selection* greedy algorithm from class? Explain your answer.

Coding Questions

1. Pseudo-code the *making change* algorithm from class.
2. Pseudo-code the *continuous knapsack* algorithm from class.
3. Pseudo-code the *interval selection* algorithm from class.

Greedy Algorithms: Advanced

Short Answer Questions

1. Consider a brute-force solution to the *making change* problem. What would the runtime of this algorithm be?
2. Given a coin-set of 1,6,10; provide a counter-example showing that the greedy algorithm no longer works.
3. Consider the *discrete knapsack problem*. For this problem, you must take the entire item and its full cost and weight. Provide a counter-example to show that the greedy selection rule for continuous knapsack won't work for this variant of the problem.
4. Consider a greedy algorithm for *interval selection* that selects the interval with the fewest conflicts (i.e., the fewest other intervals that are overlapping with this one). Provide a counter-example showing that this will not always produce the optimal solution.

Coding/Proof Questions

1. Argue the the *making change* problem has *optimal substructure*.
2. Argue that *making change* has the *greedy choice property* (i.e., the largest coin must be in the optimal solution.). Use 1,5,10,25 as your coin set and only argue for change amounts of 25 or higher.
3. Prove by induction that the *interval selection* greedy algorithm from class is optimal.

Divide and Conquer: Basic

Short Answer Questions

1. In class, we saw a Div. and Conq. algorithm for *closest pair of points* that divided the list of points in half, recursively solved, then checked points across the dividing line before returning a solution. Describe what the runtime of this algorithm (be precise / prove it) and describe why this approach is not ideal.
2. Once we came up with a more clever algorithm for *closest pair of points*, what was the recurrence relation and runtime of this algorithm? Explain your answer.
3. Briefly describe how we more cleverly combined solutions to subproblems of the *closest pair of points* problem.
4. Describe the solution we produced for the *trominoes* problem.
5. What was the runtime of our solution to the *trominoes* problem. Explain your answer.

Coding/Proof Questions

1. Directly solve the following recurrence that we did in class: $T(n) = 2T(\frac{n}{2}) + n$.
2. Use the substitution method to solve the following recurrence that we did in class: $T(n) = 2T(\frac{n}{2}) + n$.
3. Use the substitution method to solve the following recurrence that we did in class: $T(n) = 2T(\frac{n}{2}) + 1$.

Divide and Conquer: Advanced

Short Answer Questions

1. Strassen's matrix multiplication algorithm is a clever way to reduce the runtime of this problem. What is the recurrence relation for the naive divide-and-conquer and also for Strassen's approach? Use the Master Theorem to present the runtimes of these two approaches.
2. When discussing the Master Theorem, what is the intuitive explanation of case 1? If your recurrence is $T(n) = aT(\frac{n}{b}) + f(n)$, then let $k = log_b(a)$. Case 1 states that if $f(n) \in O(n^{k-\epsilon}$ for some positive ϵ, then $T(n) \in \Theta(n^k)$.
3. When discussing the Master Theorem, what is intuitive explanation of case 2? Remember that case 2 states that if $f(n) \in \Theta(n^k)$ then $T(n) \in \Theta(f(n)log(n)) = \Theta(n^k log(n))$.
4. When discussing the Master Theorem, what is the purpose of the *regularity condition* (part of case 3). It states that $af(\frac{n}{b}) \leq cf(n)$ for some $c < 1$ and sufficiently large n.

Coding/Proof Questions

1. Pseudo-code a Div. and Conq. algorithm that finds the smallest and largest index of a given integer in a sorted array. If the element is not in the array, report that as well. State the runtime of your algorithm. Notice, that the element might be in the array multiple times.
2. Pseudo-code a Div. and Conq. algorithm that finds the largest and smallest elements in an array (not sorted). What is the runtime of your algorithm? Is it better than doing a linear scan?
3. Write a recursive divide and conquer algorithm that takes an unsorted array and turns it into a min-heap. You may assume you have access to *percolateDown(node)*.

Dynamic Programming: Basic

Short Answer Questions

1. Dynamic programming often works when problems have *overlapping subproblems*. What exactly does this mean? Provide an example from class.
2. When using dynamic programming, what is *memoization*? Describe why it is useful for solving problems.
3. When using *memoization*, why do we typically use an array instead of some other data structure?
4. When solving the *discrete knapasack problem*, what was the recurrence for our recursive case for solving this problem in terms of smaller subproblems?
5. When solving the *discrete knapsack problem*, what was the size of the array we used to store solutions? Briefly describe why.
6. When solving the *weighted interval scheduling* problem, what was the function $p(j)$. Briefly describe why it is useful.
7. When solving the *weighted interval scheduling* problem, what was the recursive function we used for our dynamic programming solution? Briefly explain the intuition behind the formula.
8. What is the runtime of our dynamic programming solution to *weighted interval scheduling*? Briefly explain.

Coding/Proof Questions

1. Write psuedo-code for a simple dynamic programming solution to *fib(n)*, a function that returns the nth fibonacci number.
2. Write a method that given a list of weighted intervals, computes the array $p(j)$ that we need to run the dynamic programming algorithm from class. Your algorithm must be $O(nlog_2(n))$.
3. Write psudo-code for our solution to the *weighted interval scheduling* problem. You may assume $p(j)$ is already calculated for each interval j.

Dynamic Programming: Advanced

Short Answer Questions

1. Briefly describe why our runtime for *discrete knapsack* from class is not actually a polynomial time algorithm.
2. When solving the *longest common subsequence (LCS)* problem, what was the runtime of a brute-force solution? Briefly describe why.
3. When solving the *LCS* problem, what was our recursive solution in the case that the last two characters match? Briefly describe the intuition behind this.
4. When solving the *LCS* problem, what was our recursive solution in the case that the last two characters **DO NOT** match? Briefly describe the intuition behind this.
5. When solving the *LCS* problem, what was the size of the array we used to store the subproblems?
6. When solving the *making change* problem, what was the recursive case when the highest coin was too high for the amount of change to be made? Explain why?
7. When solving the *making change* problem, what was the recursive case when the highest coin could be used for the given amount of change? Explain why?

Coding/Proof Questions

1. Psuedocode a method that, given an array produced by our *LCS* algorithm, performs backtracking to determine the actual string that is the longest common subsequence.
2. Psudocode a method that, given an array produced by our dynamic programming algorithm for *making change*, produces the actual coins to use in the solution.