**CSC 312 REVISION PAPER**

1. Define the term algorithm and state the criteria that every algorithm must satisfy.

. Algorithm Definition:

An algorithm is a finite sequence of well-defined, unambiguous instructions that are executed in a specific order to solve a specific problem or achieve a desired outcome.

Criteria for an Algorithm:

1. Finite: The algorithm must terminate after a finite number of steps.
2. Definite: Each step in the algorithm must be clear and unambiguous, leaving no room for interpretation.
3. Input-Output: The algorithm must have a well-defined set of inputs and outputs.
4. Effectiveness: The algorithm must be effective in solving the intended problem or achieving the desired outcome.
5. Efficiency: The algorithm should be efficient in terms of time and resource consumption.
6. Explain the three basic design goals that one should strive when design for a program and how the complexity of a program can be measured . [4 marks]

Program Design Goals and Complexity

b. Design Goals:

1. Correctness: The program should produce the correct output for all valid inputs.
2. Readability: The program should be well-structured and easy to understand by humans.
3. Efficiency: The program should be efficient in terms of time and resource consumption.

Complexity Measurement:

The complexity of a program can be measured using various metrics, such as:

* Time complexity: Measures the running time of the program as a function of the input size.
* Space complexity: Measures the amount of memory required by the program as a function of the input size.
* Big O notation: Provides an upper bound on the time or space complexity of an algorithm.

1. Using an array of 100 elements, define best case, worst case and average case time complexities. [4 marks]

Array Time Complexities:

* Best Case: O(1) - Achieved when the desired element is found in the first iteration.
* Worst Case: O(n) - Achieved when the desired element is not found or is located at the end of the array.
* Average Case: O(n/2) - Achieved when the desired element is found on average after searching half of the array.

1. Explain any two factors that the running time of a program depends on factors.

Factors:

1. Input size: Larger inputs usually take longer to process.
2. Algorithm choice: Different algorithms have different time complexities.
3. Hardware platform: Faster hardware can execute programs faster.
4. Programming language: Some languages are inherently faster than others.
5. What is the smallest value of n such that an algorithm who's running times is than an algorithm whose running time is 100n faster than the algorithm whose running time is 2n on the same machine? [3 marks]

Comparing Running Times

e. Smallest n:

Let n be the smallest number for which the running time of the second algorithm is faster than the first.

* First Algorithm: 100n
* Second Algorithm: 2n

We need to find n such that:

2n < 100n

Dividing both sides by n:

2 < 100

Therefore, the smallest value of n is 2.

f.) explain the characteristic properties associated with a problem that can be solved using dynamic programming

Dynamic Programming:

Dynamic Programming is a technique for solving problems by breaking them down into smaller subproblems, solving them recursively, and storing the solutions for future reference.

Characteristic Properties:

* Optimal substructure: Optimal solutions to larger problems can be constructed from optimal solutions to smaller subproblems.
* Overlapping subproblems: The same subproblems are solved repeatedly during the recursive process.
* Memorization: Storing solutions to subproblems to avoid recomputation.

P, NP-hard, and NP-complete Problems

g.) Briefly explain the concept of P, NP-hard and NP- complete problems.

Complexity Classes:

* P: Problems that can be solved in polynomial time (efficiently).
* NP: Problems whose solutions can be verified in polynomial time (easily checked once found).
* NP-hard: Problems that are at least as hard as any NP problem (believed to be difficult to solve).
* NP-complete: Problems that are both NP and NP-hard (considered the hardest problems in NP).

**QUESTION THREE**

1. Algorithm analysis is the study of an algorithm's efficiency with respect to resource utilization, discuss these resources [4 marks]

**Resources in Algorithm Analysis:**

When analyzing an algorithm's efficiency, we consider several resources:

1. **Time Complexity:** This measures the amount of time it takes to execute the algorithm. It is usually expressed as a function of the input size (n). Common time complexities include O(1), O(log n), O(n), O(n log n), and O(n^2).
2. **Space Complexity:** This measures the amount of memory required by the algorithm. It is also expressed as a function of the input size. Common space complexities include O(1), O(log n), O(n), and O(n^2).
3. **Auxiliary Space:** This refers to the additional space needed by the algorithm besides the space required to store the input and output.
4. **Communication Cost:** This measures the amount of data exchanged between different processors or memory units in a parallel or distributed computing environment.
5. **Energy Consumption:** This measures the energy consumed by the algorithm, especially relevant for battery-powered devices.
6. Differentiate between greedy algorithm and dynamic programing and state the nature of problems that can be solved using each approach. [4 marks]

Greedy vs. Dynamic Programming

b. Greedy vs. Dynamic Programming:

Greedy Algorithm:

* Makes locally optimal choices at each step based on immediate benefit.
* Simple and efficient for certain problems.
* Does not guarantee optimal solution for all problems.

Dynamic Programming:

* Solves subproblems optimally and combines them to solve the main problem.
* More complex and computationally expensive than greedy algorithms.
* Guarantees optimal solution for many problems.

Problems for Each Approach:

Greedy:

* Shortest path problems (Dijkstra's algorithm)
* Activity selection problem
* Fractional knapsack problem

Dynamic Programming:

* Longest common subsequence
* Edit distance
* Optimal binary search tree
* Traveling salesman problem

1. State fractional knapsack problem and give an algorithm for fractional knapsack problem using greedy strategy. [4 marks]

Fractional Knapsack Problem:

Given a set of items with weights and profits, maximize the total profit while filling a knapsack with limited capacity.

Greedy Algorithm:

1. Sort items by their profit-to-weight ratio (p/w).
2. Fill the knapsack with items in the sorted order until full or reaching the weight limit.
3. If the knapsack is not full, add a fraction of the last item that fills the remaining space without exceeding the weight limit.

This algorithm guarantees an optimal solution for the fractional knapsack problem.

1. Find an optimal solution to the fractional knapsack problem for an instance with number of items 7. Capacity of the sack W-15, profit associated with the items (p1.p2.....p7) (10,5,15,7,6,18,3) and weight associated with each item (w1,w2....,w7) (2,3,5,7,1,4,1). [6 marks]

Given:

* Number of items (n) = 7
* Knapsack capacity (W) = 15
* Profits (p) = [10, 5, 15, 7, 6, 18, 3]
* Weights (w) = [2, 3, 5, 7, 1, 4, 1]

Solution:

1. Sort items by p/w ratio:
   * Item 6: p/w = 18/4 = 4.5
   * Item 4: p/w = 7/7 = 1
   * Item 1: p/w = 10/2 = 5
   * Item 3: p/w = 15/5 = 3
   * Item 5: p/w = 6/1 = 6
   * Item 2: p/w = 5/3 = 1.66
   * Item 7: p/w = 3/1 = 3
2. Fill the knapsack:
   * Add item 6: W = 0 + 4 = 4, Profit = 0 + 18 = 18
   * Add item 4: W = 4 + 7 = 11, Profit = 18 + 7 = 25
   * Add item 1: W = 11 + 2 = 13, Profit = 25 + 10 = 35
   * Knapsack is full.

Therefore, the optimal solution is to take all of items 6, 4, and 1 for a total profit of 35.

1. Solve multiplication using Divide and Conquer strategy 12345678 21394276 [4 marks]

Step 1: Divide the numbers

Split both 12345678 and 21394276 into two halves:

* Left half of 12345678: 1234
* Right half of 12345678: 5678
* Left half of 21394276: 2139
* Right half of 21394276: 4276

Step 2: Recursively multiply the smaller numbers

Perform four smaller multiplications:

* ac = 1234 \* 2139
* bd = 5678 \* 4276
* ad = 1234 \* 4276
* bc = 5678 \* 2139

Step 3: Combine the results

Use the Karatsuba algorithm to combine the results:

* Part 1: ac \* 10^(2n) = 2639526 \* 10^8 = 263952600000000
* Part 2: (ad + bc) \* 10^n = (5224284 + 12186524) \* 10^4 = 174108080000
* Part 3: bd = 24302352 \* 1 = 24302352

Total product:

263952600000000 + 174108080000 + 24302352 = 264126842539128

Therefore, 12345678 \* 21394276 = 264126842539128.

**QUESTION FOUR**

Determine the best case and worst-case time complexities of the following two functions

[4 marks]

fun1() and fun2():

int funl (int n)

if (n <=1) return n return 2\*fun1 (n-1);

}

int fun2(int n)

{

if (n <=1) return n

return fun2(n-1)+fun2(n-1):

Time Complexity of fun1 and fun2:

fun1:

* Best Case: O(1) - Occurs when n = 0 or 1.
* Worst Case: O(n) - Occurs when n > 1.

fun2:

* Best Case: O(1) - Occurs when n = 0 or 1.
* Worst Case: O(2^n) - Occurs when n > 1.

b. Explain Euclid's algorithm for computing ged(m, n), hence compute the ged and the lem of (31415 and 14142)

[4 marks]

Euclid's Algorithm:

Steps:

1. Divide the larger number (m) by the smaller number (n).
2. Let r be the remainder.
3. If r is 0, then n is the greatest common divisor (gcd).
4. Otherwise, replace m with n and n with r.
5. Repeat steps 1-4 until r is 0.

Example:

Given m = 31415 and n = 14142:

1. 31415 / 14142 = 2 remainder 2831
2. 14142 / 2831 = 5 remainder 1117
3. 2831 / 1117 = 2 remainder 597
4. 1117 / 597 = 1 remainder 520
5. 597 / 520 = 1 remainder 77
6. 520 / 77 = 6 remainder 56
7. 77 / 56 = 1 remainder 21
8. 56 / 21 = 2 remainder 14
9. 21 / 14 = 1 remainder 7
10. 14 / 7 = 2 remainder 0

Therefore, gcd(31415, 14142) = 7.

Least Common Multiple (lcm):

lcm(m, n) = m \* n / gcd(m, n)

Therefore, lcm(31415, 14142) = 31415 \* 14142 / 7 = 65472900

e. If the first program P1 takes 100n2 milliseconds and the second program P2 takes 5n³ milliseconds. Determine and recommend which program P1 or P2 is better and at what condition?

[4 marks]

**Comparing P1 and P2:**

**P1:** O(n^2) **P2:** O(n^3)

P1 is better for small values of n, as its time complexity grows quadratically. However, for large values of n, P2 becomes the better choice despite its higher time complexity, as the cubic growth outperforms the quadratic growth in the long run.

The exact condition for P1 to be better than P2 is:

n < (100/5)^(1/2) ≈ 4.47

Therefore, P1 is better than P2 for n < 4.47, and P2 becomes the better choice for n ≥ 4.47.

d. One of the two software packages. A or B. should be chosen to process data collections. containing each up to 109 records. Average processing time of the package A is TA(n)= 0.001 milliseconds and the average processing time of the package B is Ta(n) 500vn milliseconds. Which algorithm has better performance in a "Big-Oh" sense? Work out exact

conditions when these packages outperform each other.

Comparing Package A and B:

Package A: O(n) Package B: O(n^0.5)

Big-Oh Comparison:

* Package A grows linearly with n.
* Package B grows slower than linearly, with a square root factor.

Therefore, in the "Big-Oh" sense, Package B has better performance as its time complexity grows slower than Package A's linear growth.

Performance Outperformance:

Package B outperforms Package A when:

n > (500/0.001)^2 ≈ 2.5e+08

Therefore, Package B becomes the faster option for data collections containing more than 2.5e+08 records. For smaller data collections, Package A offers better performance.

**QUESTION TWO**

a. What are the general rules followed when analyzing running time of programs.

[3 marks]

. General Rules:

1. Ignore constant factors: Focus on the dominant term as input size grows.
2. Focus on loops: Analyze the number of times loops iterate.
3. Consider nested loops: Multiply the complexities of nested loops.
4. Analyze recursive functions: Consider the number of recursive calls and the work done in each call.
5. Use asymptotic notations: Big O, Theta, and Omega notations help express complexity in a concise way.

b. Explain the concept of control abstraction in Dynamic Programming.

[2 marks]

Control Abstraction:

In dynamic programming, control abstraction refers to the process of focusing on the subproblems and their relationships rather than the details of how to solve them. This allows for a more modular and efficient approach by:

* Identifying subproblems: Breaking down the larger problem into smaller, easier-to-solve subproblems.
* Memorization: Storing solutions to subproblems to avoid redundant calculations.
* Optimal substructure: Utilizing the property that optimal solutions to larger problems can be built from optimal solutions to their subproblems.

c. Explain the differences between Prim's and Kruskal's algorithm. Link this to efficiency analysis.

[2 marks]

Differences:

|  |  |  |
| --- | --- | --- |
| Feature | Prim's Algorithm | Kruskal's Algorithm |
| Algorithm type | Greedy | Greedy |
| Data structure used | Priority queue | Disjoint-set data structure |
| Edge selection | Chooses cheapest edge connected to the growing tree | Chooses cheapest edge overall |
| Efficiency | O(E log V) | O(E log E) |

drive\_spreadsheetExport to Sheets

Efficiency Analysis:

* Prim's algorithm has a time complexity of O(E log V), where E is the number of edges and V is the number of vertices.
* Kruskal's algorithm has a time complexity of O(E log E), which is generally faster for dense graphs (many edges).

d. Define Asymptotic Notations? Explain their significance in analyzing algorithms?

14 Marks]

Asymptotic notations are mathematical tools used to describe the behavior of a function as its input size grows towards infinity. They provide a concise and efficient way to compare the efficiency of different algorithms and algorithms with different input sizes.

Asymptotic Notations:

* Big O (O) notation: Represents the upper bound on the growth rate of a function.
* Theta (Θ) notation: Represents the exact growth rate of a function.
* Omega (Ω) notation: Represents the lower bound on the growth rate of a function.

Significance:

* Asymptotic notations provide a concise and intuitive way to compare the efficiency of different algorithms.
* They help us understand how the running time of an algorithm scales with the input size.
* They enable us to focus on the dominant term in the running time, ignoring constant factors and lower-order terms.

e. Explain N-queens problem using Backtracking and draw the state space tree of 4-queens problem

[ 6 Marks]

N-Queens Problem:

The task is to place N queens on an N x N chessboard such that no two queens attack each other (diagonally, vertically, or horizontally).

Backtracking:

Backtracking is a systematic search algorithm that explores all possible solutions by making tentative decisions and backtracking when they lead to invalid states.

State Space Tree for 4-Queens Problem:

Each node represents a partial placement of queens on the board. Child nodes represent placements where a new queen is placed on a different row without violating any constraints.

* Level K of the tree represents configurations with K queens placed on the board.
* A leaf node represents a complete placement of 4 queens on the board, which is a valid solution if no two queens attack each other.

f. Describe the travelling salesman problem and discuss how to solve it using dynamic programming?

[3 marks]

Given a list of cities and the distances between them, find the shortest possible route that visits each city exactly once and returns to the starting city.

Dynamic Programming Solution:

1. Define subproblems: Subproblem(i, S) represents the shortest possible route that starts at city i, visits all cities in the set S, and returns to city i.
2. Solve subproblems: Use recursion to solve smaller subproblems and build the solution for larger ones.
3. Memorize solutions: Store solutions to subproblems to avoid redundant calculations.

Dynamic programming allows for an efficient solution to the travelling salesman problem by breaking it down into smaller subproblems and utilizing memoization to avoid recomputing solutions.