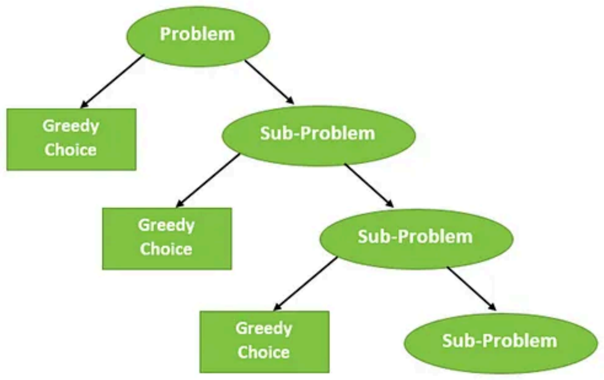
CSC124: Design & Analysis of Algorithms

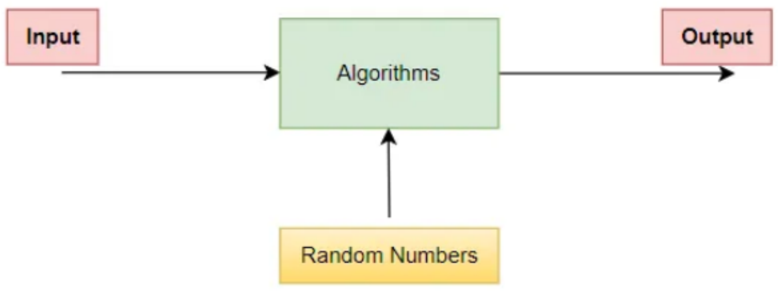
Ramel Cary B. Jamen 2019-2093 2023T2

1. **Concepts**: Describe the following

   a. **Greedy algorithms**
   Greedy algorithms solve problems by making the choice that seems best at the moment, hoping it leads to the overall best outcome. They focus on the immediate answer without considering the long run. While they might not always find the absolute perfect solution, they often come pretty close in a reasonable amount of time. This makes them a good choice for many situations.



   b. **Randomized algorithms**
   A randomized algorithm is essentially an algorithm that incorporates randomness at certain stages into the algorithm to enhance performance by evenly distributing inputs.



2. **Algorithms**. Given their contexts/classification, describe the following algorithms as computer science problems:
   a. **Divide and Conquer algorithms**
      i. **tower of hanoi** - This algorithm can be solved recursively by breaking down the problem into smaller subproblems. At each step, the algorithm recursively moves n-1 disks from the source rod to an auxiliary rod, then moves the nth disk from the source rod to the destination rod, and finally recursively moves the n-1 disks from the auxiliary rod to the destination rod.
      ii. **minmax** - This recursively explores the game tree, representing all possible moves and their outcomes. It alternates between maximizing and minimizing players at each level of the tree. This algorithm is typically used in games like chess, tic-tac-toe, or checkers to determine the best move.
      iii. **matrix multiplication** - The best known well performed solution is using a divide-and-conquer approach known as Strassen's algorithm. Strassen's algorithm recursively divides the matrices into smaller submatrices, computes various products with fewer multiplications, and combines them to form the final result.
      iv. **merge sort** - it recursively divides the array into two halves until each sub-array contains only one element. Then, it merges the sub-arrays by comparing elements and arranging them in the correct order. This merging process continues until the entire array is sorted.
      v. **fibonacci sequence** - it can be computed recursively using the following recurrence relation: F(n) = F(n-1) + F(n-2), with base cases F(0) = 0 and F(1) = 1. However, this recursive approach has exponential time complexity. Alternatively, dynamic programming or memoization techniques can be used to optimize the computation by storing the previously computed Fibonacci numbers.

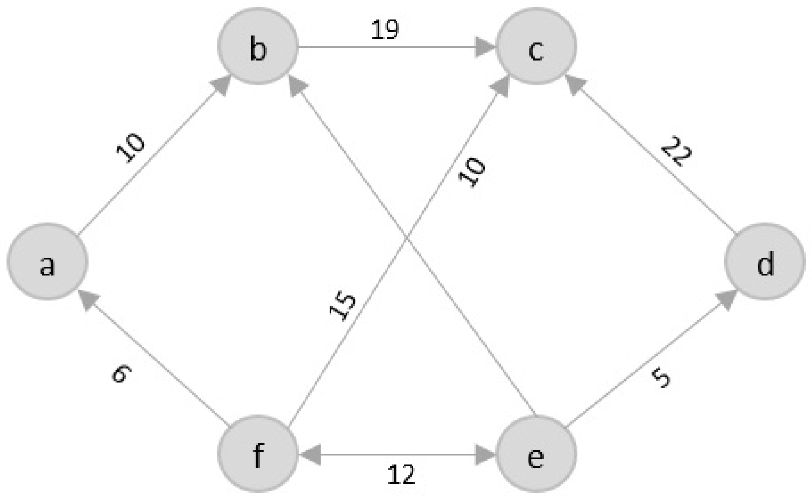
   b. **Greedy algorithms**
      i. **Coin change (coin counting) problem** - Given a set of coin denominations and a target amount, find the minimum number of coins needed to make up that amount. A solution is that it greedily selects the largest denomination coin that can be used without exceeding the target amount until the target amount is reached.
      ii. **Traveling salesperson** - Given a list of cities and the distances between each pair of cities, find the shortest possible route that visits each city exactly once and returns to the starting city. A solution is that it greedily selects the nearest unvisited city at each step until all cities are visited, then return to the starting city.
      iii. **0-1 Knapsack Problem** - Given a set of items, each with a weight and a value, determine the maximum value of items that can be included in a knapsack of limited capacity without exceeding the capacity. A solution is

that it greedily selects items based on their value-to-weight ratio until the knapsack capacity is reached.

iv. **Map-coloring** - Given a map with regions (vertices) and their connections (edges), assign colors to each region such that no two adjacent regions have the same color, using the fewest possible colors. A solution is that it greedily assigns colors to regions one by one, choosing the first available color not already used by neighboring regions.

v. **Vertex-coloring** - Given an undirected graph, assign colors to each vertex such that no two adjacent vertices have the same color, using the fewest possible colors. A solution is that it greedily assigns colors to vertices one by one, choosing the first available color not already used by neighboring vertices.

3. **<u>Problem solving.</u>** What is the solution to this Traveling Salesperson Problem (shortest path)?



Shortest Path - A F C D E B A
Minimum Cost - 68

4. **<u>Analysis.</u>** What are the asymptotic runtimes of the divide-and-conquer implementations of the following algorithms
   a. **tower of hanoi** - time complexity of $O(2^n)$ where n is the number of disks.
   b. **minmax** - finding the minimum and maximum elements in an array has a time complexity of $O(n)$.
   c. **matrix multiplication** - has the time complexity of $O(n^3)$ because of the recurrence relation of the algorithm.
   d. **merge sort** - The time complexity is $O(n\ log\ n)$ because each merge step takes linear time, and there are log n levels of recursion.
   e. **fibonacci** - The Fibonacci sequence can be computed using matrix exponentiation techniques, which have a time complexity of $O(n\ log\ n)$.

5. **Programming.** Implement the function for **tower of hanoi**, to transfer disks in `tower1` to `tower3`

```
def towerOfHanoi(n, tower1, tower2, tower3)
{
        if n == 1:
                print(f"Move disk 1 from {tower1} to {tower3}")
                return
        towerOfHanoi(n-1, tower1, tower3, tower2)
        print(f"Move disk {n} from {tower1} to {tower3}")
        towerOfHanoi(n-1, tower2, tower1, tower3)
}
```

6. **Programming.** Implementation of **minmax** using the divide and conquer strategy

```
def minmax(Array, start, end)
{
        if start == end:
                return Array[start], Array[start]

        if end == start + 1:
                return (Array[start], Array[end]) if Array[start] < Array[end] else
                (Array[end], arr[start])

        mid = (start + end) // 2
        min1, max1 = minmax(Array, start, mid)
        min2, max2 = minmax(Array, mid + 1, end)

        return min(min1, min2), max(max1, max2)
}
```

7. **Programming.** Implement the greedy coin change problem

```
def coinchange(amount, ArrayOfDenominations)
{
    denominations.sort(reverse=True)

    coins_used = []
    total_coins = 0

    for coin in denominations:
        while amount >= coin:
            amount -= coin
            coins_used.append(coin)
            total_coins += 1

    if amount == 0:
        return coins_used, total_coins
    else:
        return "Change not possible", 0
}
```

Deliverable:

Submit a PDF file containing answers to all items in this assignment.

ALL programming codes should be in the form of a function definition (as outlined).

- DO NOT include test runs.