Module 3 - Brute Force Algorithms

1. Understanding Brute Force Approach

Concepts:

Definition: A straightforward method based directly on the problem's statement and definitions.
Applications:

- **Computing powers (e.g., an)**
- **Computing factorials (e.g., n!)**
- **Matrix multiplication**
- **Searching for a key in a list**

Strengths & Weaknesses:

Strengths:

- **Wide applicability**
- **Simplicity**
- **Reasonable algorithms for some crucial problems like sorting, searching, matrix multiplication**

Weaknesses:

- **Rarely yields efficient algorithms**
- **Some brute-force algorithms are unacceptably slow**
- **Not as constructive as other design techniques**

Selection Sort

Concepts:

Algorithm Description: Repeatedly finds the minimum element from the unsorted part and moves it to the sorted subarray.

Steps:

Maintain two subarrays: one sorted and one unsorted.

In each iteration, pick the smallest element from the unsorted subarray and move it to the sorted subarray.

```
void selectionSort(int arr[], int n) {
   for (int i = 0; i < n - 1; ++i) {
      int minIndex = i;
      for (int j = i + 1; j < n; ++j) {
         if (arr[j] < arr[minIndex]) {
            minIndex = j;
         }
      }
      // Swap the found minimum element with the first element
      int temp = arr[minIndex];
      arr[minIndex] = arr[i];
      arr[i] = temp;
   }
}
```

Analogy: Think of organizing books on a shelf. You repeatedly find the smallest book from the unorganized section and place it at the start of the organized section

Brute-Force String Matching

Concepts:

Description: Compares a given pattern with all substrings of a text character by character until a match or mismatch is found.

Steps:

Align the pattern at the beginning of the text.

Compare each character of the pattern with the corresponding character in the text.

If all characters match, success; otherwise, realign the pattern one position to the right and repeat.

Pseudocode:

```
int bruteForceStringMatch(char T[], char P[], int n, int m) {
   for (int i = 0; i <= n - m; ++i) {
      int j = 0;
      while (j < m && P[j] == T[i + j]) {
         ++j;
      }
      if (j == m) return i; // Match found
   }
   return -1; // No match found
}
```

Efficiency: Θ(mn) comparisons in the worst case.

Analogy: Imagine looking for a specific word in a long paragraph. You check each possible starting point letter by letter until you find the word or exhaust the paragraph.

4. Exhaustive Search

Concepts:

Definition: A brute-force technique that generates all potential solutions systematically and evaluates them.

Method:

Generate a list of all potential solutions.

Evaluate each solution, disqualify infeasible ones, and keep track of the best one found so far.

Announce the solution(s) after completing the search.

5. Traveling Salesman Problem (TSP)

Concepts:

Problem Statement: Given n cities with known distances between each pair, find the shortest tour that passes through all cities exactly once before returning to the starting city.

Brute-Force Solution: Generate all permutations of cities and calculate the total distance for each.

Example:
Consider cities A, B, C, D with distances:

A-B: 2, A-C: 8, A-D: 5
B-C: 3, B-D: 4
C-D: 7

Possible tours and their costs:

A → B → C → D → A: 2+3+7+5 = 17
A → B → D → C → A: 2+4+7+8 = 21
A → C → B → D → A: 8+3+4+5 = 20
A → C → D → B → A: 8+7+4+2 = 21
A → D → B → C → A: 5+4+3+8 = 20
A → D → C → B → A: 5+7+3+2 = 17

Analogy: Imagine planning a road trip where you want to visit multiple cities. You explore every possible route to find the shortest one.

Knapsack Problem

Concepts:

Problem Statement: Given items with weights and values, determine the most valuable subset of items that fit into a knapsack of capacity W.

Brute-Force Solution: Evaluate all subsets of items and select the one with the highest value without exceeding the weight limit.

Items:

ITEM	WEIGHT	VALUE
1	2	$20
2	5	$30
3	10	$50
4	5	$10

Knapsack Capacity: 16

Subset Evaluation:

SUBSET	TOTAL WEIGHT	TOTAL VALUE
{1}	2	$20
{2}	5	$30
{3}	10	$50
{4}	5	$10
{1,2}	7	$50
{1,3}	12	$70
{1,4}	7	$30
{2,3}	15	$80
{2,4}	10	$40
{3,4}	15	$60
{1,2,3}	17	Not Feasible
{1,2,4}	12	$60
{1,3,4}	17	Not Feasible
{2,3,4}	20	Not Feasible
{1,2,3,4}	22	Not Feasible