**Q1.** For the function defined by

$f(n) = 2n^3 + 3n^2 + 1$ and $g(n) = 2n^3 + 1$: show that

(i) $f(n) = \Theta(g(n))$

(ii) $f(n) \; != \Theta(n^2)$

(iii) $n^4 \; != \Theta(g(n))$

## Solution:

### Given:

- $f(n) = 2n^3 + 3n^2 + 1$
- $g(n) = 2n^3 + 1$

**Part (i)** $f(n) = \Theta(g(n))$

We need to show that $f(n) = \Theta(g(n))$, meaning that the growth rate of $f(n)$ and $g(n)$ is the same. To do this, let's compare their leading terms:

- $f(n) = 2n^3 + 3n^2 + 1$
- $g(n) = 2n^3 + 1$

For large $n$, the term $2n^3$ dominates both functions. Therefore, $f(n)$ and $g(n)$ have the same asymptotic behavior because the lower-order terms (like $3n^2$ and the constant 1) become insignificant as $n \to \infty$.

Thus, we can conclude:

$$f(n) = \Theta(g(n))$$

**Part (ii)** $f(n) \neq \Theta(n^2)$

We need to show that $f(n) \neq \Theta(n^2)$.

The function $f(n) = 2n^3 + 3n^2 + 1$ clearly has a cubic term $2n^3$, which grows faster than $n^2$. For large $n$, $n^3$ dominates the growth of $f(n)$, while $n^2$ grows slower. Since the highest order term in $f(n)$ is $n^3$, it cannot be bounded asymptotically by a function that grows like $n^2$.

Thus, we can conclude:

$$f(n) \neq \Theta(n^2)$$

**Part (iii)** $n^4 \neq \Theta(g(n))$

Here, we need to show that $n^4 \neq \Theta(g(n))$.

We know that $g(n) = 2n^3 + 1$. For large $n$, the term $2n^3$ dominates $g(n)$. Comparing this to $n^4$, it is clear that $n^4$ grows faster than $n^3$.

Since $n^4$ grows asymptotically faster than $g(n)$, $n^4$ cannot be bounded by $g(n)$.

Thus, we can conclude:

$$n^4 \neq \Theta(g(n))$$

## Ques 02. Find the optimal solution for the knapsack instance n=7 and M=15

**($P_1$,$P_2$,----------$P_n$) = (12,4,14,8,9,20,3)**
**($W_1$,$W_2$,----------$W_n$) = (3,2,5,6,4,1,7)**

## Solution:

To solve the knapsack problem for $n = 7$ and $M = 15$, where:

- The profits $P = (12, 4, 14, 8, 9, 20, 3)$
- The weights $W = (3, 2, 5, 6, 4, 1, 7)$

We aim to maximize the profit while ensuring that the total weight does not exceed the capacity $M = 15$.
For each item, we calculate the profit-to-weight ratio $P_i/W_i$:

$$\frac{P_1}{W_1} = \frac{12}{3} = 4, \quad \frac{P_2}{W_2} = \frac{4}{2} = 2, \quad \frac{P_3}{W_3} = \frac{14}{5} = 2.8$$

$$\frac{P_4}{W_4} = \frac{8}{6} = 1.33, \quad \frac{P_5}{W_5} = \frac{9}{4} = 2.25, \quad \frac{P_6}{W_6} = \frac{20}{1} = 20, \quad \frac{P_7}{W_7} = \frac{3}{7} = 0.43$$

We sort the items in descending order based on their profit-to-weight ratios:

1. $P_6$ (ratio = 20)
2. $P_1$ (ratio = 4)
3. $P_3$ (ratio = 2.8)
4. $P_5$ (ratio = 2.25)
5. $P_2$ (ratio = 2)
6. $P_4$ (ratio = 1.33)
7. $P_7$ (ratio = 0.43)

We now select items in the order of their ratios while ensuring that the total weight does not exceed 15.

- Select $P_6$ (Weight = 1, Profit = 20). Remaining capacity: $15 - 1 = 14$
- Select $P_1$ (Weight = 3, Profit = 12). Remaining capacity: $14 - 3 = 11$
- Select $P_3$ (Weight = 5, Profit = 14). Remaining capacity: $11 - 5 = 6$
- Select $P_5$ (Weight = 4, Profit = 9). Remaining capacity: $6 - 4 = 2$
- $P_2$ cannot be selected because it has weight 2, which exceeds the remaining capacity.

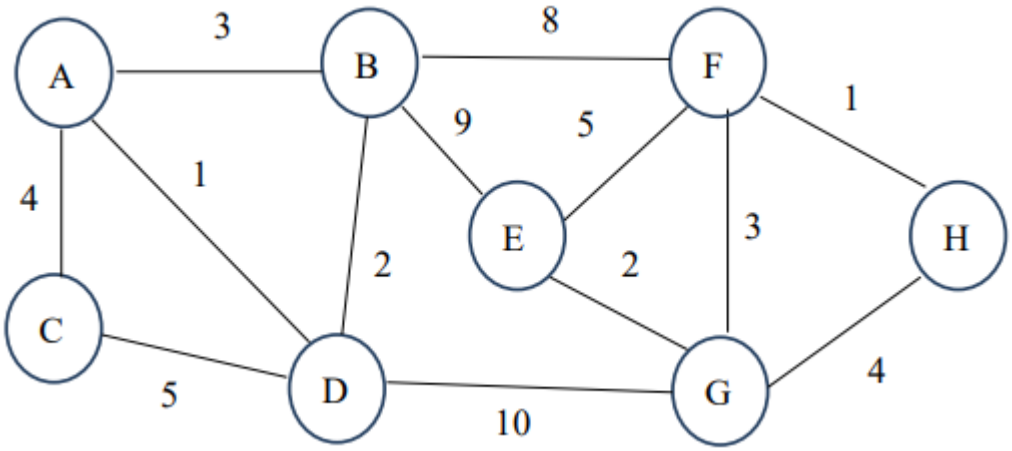
The items selected are:

- $P_6$ (Profit = 20, Weight = 1)
- $P_1$ (Profit = 12, Weight = 3)
- $P_3$ (Profit = 14, Weight = 5)
- $P_5$ (Profit = 9, Weight = 4)

The total profit is $20 + 12 + 14 + 9 = 55$, and the total weight is $1 + 3 + 5 + 4 = 13$, which is within the capacity of 15.

Thus, the optimal solution is a total profit of **55** with the selected items being $P_6$, $P_1$, $P_3$, and $P_5$.

**Ques 03. Apply Kruskal's Algorithm on the following graph to find minimum cost spanning tree**




## Solution:

First, we list the edges of the graph along with their weights:

$$\text{Edge (A, B)} = 3, \quad \text{Edge (A, C)} = 4, \quad \text{Edge (A, D)} = 1$$

$$\text{Edge (B, D)} = 2, \quad \text{Edge (B, E)} = 9, \quad \text{Edge (B, F)} = 8$$

$$\text{Edge (D, C)} = 5, \quad \text{Edge (D, E)} = 2, \quad \text{Edge (E, F)} = 5$$

$$\text{Edge (E, G)} = 2, \quad \text{Edge (F, G)} = 3, \quad \text{Edge (F, H)} = 1$$

$$\text{Edge (G, H)} = 4, \quad \text{Edge (D, G)} = 10$$

We sort the edges by their weights in ascending order:

1. $\text{Edge (A, D)} = 1$
2. $\text{Edge (F, H)} = 1$
3. $\text{Edge (B, D)} = 2$
4. $\text{Edge (D, E)} = 2$
5. $\text{Edge (E, G)} = 2$
6. $\text{Edge (A, B)} = 3$
7. $\text{Edge (F, G)} = 3$
8. $\text{Edge (A, C)} = 4$
9. $\text{Edge (G, H)} = 4$

10. $\text{Edge (D, C)} = 5$
11. $\text{Edge (E, F)} = 5$
12. $\text{Edge (B, F)} = 8$
13. $\text{Edge (B, E)} = 9$
14. $\text{Edge (D, G)} = 10$

We will add edges to the MST, ensuring no cycles are formed.

- **Edge (A, D) = 1** (selected)
- **Edge (F, H) = 1** (selected)
- **Edge (B, D) = 2** (selected)
- **Edge (D, E) = 2** (selected)
- **Edge (E, G) = 2** (selected)
- **Edge (A, B) = 3** (selected)
- **Edge (F, G) = 3** (not selected as it forms a cycle)

At this point, we have selected 6 edges (as the number of vertices $n = 8$, and the number of edges in the MST should be $n - 1 = 7$).

- **Edge (A, C) = 4** (selected to complete the MST)

The edges in the MST are:

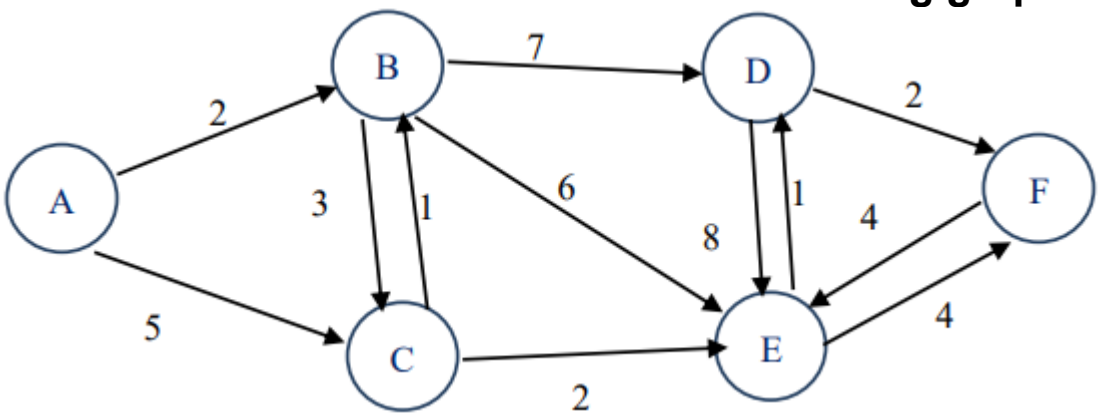
1. $\text{Edge (A, D)} = 1$
2. $\text{Edge (F, H)} = 1$
3. $\text{Edge (B, D)} = 2$
4. $\text{Edge (D, E)} = 2$
5. $\text{Edge (E, G)} = 2$
6. $\text{Edge (A, B)} = 3$
7. $\text{Edge (A, C)} = 4$

The total minimum cost of the spanning tree is:

$$1 + 1 + 2 + 2 + 2 + 3 + 4 = 15$$

Thus, the **Minimum Cost Spanning Tree** has a total cost of **15**.

**Ques 04. Apply Dijkastra's Algorithm to find the shortest path from source vertex 'A' to all other vertices for following graph.**




## Solution:

- **Start from the source vertex A.**
- Set distance from A to A as 0: `dist(A) = 0`.
- Set distance from A to all other vertices to infinity: `dist(B) = ∞`, `dist(C) = ∞`, `dist(D) = ∞`, `dist(E) = ∞`, `dist(F) = ∞`.
- Mark all vertices as unvisited.
- Start from A (distance = 0).
- Update distances to its neighbors:
  - B: `dist(B) = 0 + 2 = 2`
  - C: `dist(C) = 0 + 5 = 5`
- Current distances: `A: 0`, `B: 2`, `C: 5`, `D: ∞`, `E: ∞`, `F: ∞`.
- Mark A as visited.

- Choose the vertex with the smallest tentative distance, B (distance = 2).
- Update distances to its neighbors:
  - C: `dist(C) = min(5, 2 + 3) = 5`
  - D: `dist(D) = 2 + 7 = 9`
  - E: `dist(E) = 2 + 6 = 8`
- Current distances: `A: 0`, `B: 2`, `C: 5`, `D: 9`, `E: 8`, `F: ∞`.
- Mark B as visited.
- Choose the vertex with the smallest tentative distance, C (distance = 5).
- Update distances to its neighbors:
  - E: `dist(E) = min(8, 5 + 2) = 7`
- Current distances: `A: 0`, `B: 2`, `C: 5`, `D: 9`, `E: 7`, `F: ∞`.
- Mark C as visited.
- Choose the vertex with the smallest tentative distance, E (distance = 7).
- Update distances to its neighbors:
  - D: `dist(D) = min(9, 7 + 8) = 9`
  - F: `dist(F) = 7 + 4 = 11`
- Current distances: `A: 0`, `B: 2`, `C: 5`, `D: 9`, `E: 7`, `F: 11`.
- Mark E as visited.
- Choose the vertex with the smallest tentative distance, D (distance = 9).
- Update distances to its neighbors:
  - F: `dist(F) = min(11, 9 + 2) = 11`
- Current distances: `A: 0`, `B: 2`, `C: 5`, `D: 9`, `E: 7`, `F: 11`.
- Mark D as visited.

- F is the last unvisited vertex, and no further updates are needed.
- Mark F as visited.

## Final Distances from A:

- `dist(A) = 0`
- `dist(B) = 2`
- `dist(C) = 5`
- `dist(D) = 9`
- `dist(E) = 7`
- `dist(F) = 11`

## Shortest Paths:

- A → B (distance = 2)
- A → C (distance = 5)
- A → B → E (distance = 7)
- A → B → D (distance = 9)
- A → B → E → F (distance = 11)

This is the solution using **Dijkstra's Algorithm**.

**Ques 05. Analyze best case, average case, and worst case time complexities of following algorithms with the help of suitable examples.**

**(a) Insertion sort**

**Ans:** Best Case Time Complexity:

- O(n): Occurs when the input list is already sorted. In this case, each element only needs to be compared once and then inserted into its correct position without shifting other elements.

Average Case Time Complexity:

- O(n^2): The average case occurs when the elements are in random order. Each element may need to be compared and shifted multiple times, leading to a quadratic number of operations.

Worst Case Time Complexity:

- O(n^2): Occurs when the input list is sorted in reverse order. Each element needs to be compared with every other element and shifted to its correct position, leading to the maximum number of operations.

Example:

- Best Case: [1, 2, 3, 4, 5]
- Average Case: [3, 1, 4, 2, 5]
- Worst Case: [5, 4, 3, 2, 1]

**(b) Quick sort**

**Ans:** Best Case Time Complexity:

- O(n log n): Occurs when the pivot divides the array into two equal halves, leading to balanced partitions and a logarithmic number of levels in the recursion tree.

Average Case Time Complexity:

- O(n log n): On average, the pivot divides the array into reasonably balanced partitions, resulting in a logarithmic number of levels in the recursion tree, similar to the best case.

Worst Case Time Complexity:

- O(n^2): Occurs when the pivot results in unbalanced partitions (e.g., always picking the smallest or largest element as the pivot), leading to a linear number of levels in the recursion tree.

Example:

- Best Case: Pivot at middle element, e.g., [5, 1, 3, 2, 4]
- Average Case: Random pivot selection, e.g., [3, 2, 5, 4, 1]

- Worst Case: Pivot always at an extreme end, e.g., [1, 2, 3, 4, 5] (if always picking the first element as pivot)

**(c) Binary sort**

**Ans:** Best Case Time Complexity:

- O(n log n): Occurs when the list is already sorted. Binary search is used to find the insertion point efficiently, and no elements need to be moved.

Average Case Time Complexity:

- O(n^2): In practice, this is similar to Insertion Sort, where elements are compared and shifted as they are inserted into the sorted portion of the list.

Worst Case Time Complexity:

- O(n^2): Even with binary search, elements may need to be shifted for insertion, leading to quadratic time complexity for shifting operations.

Example:

- Best Case: [1, 2, 3, 4, 5]
- Average Case: [3, 1, 4, 2, 5]
- Worst Case: [5, 4, 3, 2, 1]

**(d) Selection sort**

**Ans:** Best Case Time Complexity:

- O(n^2): The time complexity remains quadratic even if the array is already sorted, as every element still needs to be compared to find the minimum.

Average Case Time Complexity:

- O(n^2): Each element is compared to all other elements to find the minimum, leading to a quadratic number of comparisons.

Worst Case Time Complexity:

- O(n^2): In the worst case, like the average case, each element must be compared with all others to find the minimum.

Example:

- Best Case: [1, 2, 3, 4, 5] (still performs O(n^2) comparisons)
- Average Case: [3, 1, 4, 2, 5]

- Worst Case: [5, 4, 3, 2, 1]

**Ques 06. Multiply 2146782 x 0422812 using divide and conquer technique(use Karatsuba method).**

## Solution:

Let's split each number into two parts. For simplicity, we'll use:

- $x = 2146782$
- $y = 0422812$

We can divide each number into two parts:

- $x = 2146 \times 10^2 + 782 = 2146 \cdot 100 + 782$
- $y = 0422 \times 10^2 + 812 = 422 \cdot 100 + 812$

So, we have:

- $x_1 = 2146$
- $x_0 = 782$
- $y_1 = 422$
- $y_0 = 812$

Karatsuba's method is based on the formula: $x \times y = (x_1 \times 10^m + x_0) \times (y_1 \times 10^m + y_0) = x_1 \times y_1 \times 10^{2m} + (x_1 \times y_0 + x_0 \times y_1) \times 10^m + x_0 \times y_0$

where $m$ is the number of digits in the split parts (in our case, $m = 2$).

**Compute $x_1 \times y_1$**

$x_1 \times y_1 = 2146 \times 422 = 905,012$

**Compute $x_0 \times y_0$**

$x_0 \times y_0 = 782 \times 812 = 635,464$

**Compute $(x_1 + x_0) \times (y_1 + y_0)$**

$(x_1 + x_0) = 2146 + 782 = 2928$ $(y_1 + y_0) = 422 + 812 = 1234$ $(x_1 + x_0) \times (y_1 + y_0) = 2928 \times 1234 = 3,614,592$

**Compute the middle term**

$\text{Middle term} = (x_1 + x_0) \times (y_1 + y_0) - x_1 \times y_1 - x_0 \times y_0$ $\text{Middle term} = 3,614,592 - 905,012 - 635,464 = 2,074,116$

Using the formula: $x \times y = x_1 \times y_1 \times 10^{2m} + \text{Middle term} \times 10^m + x_0 \times y_0$ $x \times y = 905,012 \times 10^4 + 2,074,116 \times 10^2 + 635,464 = 905,012 \times 10,000 + 2,074,116 \times 100 + 635,464 = 9,050,120,000 + 207,411,600 + 635,464 = 9,258,167,064$

So, the result of $2146782 \times 0422812$ using the Karatsuba method is **9,258,167,064**.

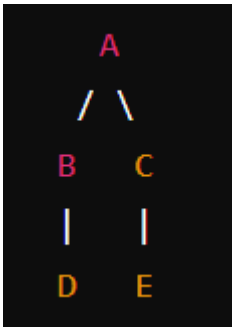
## Ques 07. Explain DFS and BDS graph transversal algorithms with the help of a suitable example.

**Solution:** DFS (Direct-First Search) is a graph traversal algorithm that explores as far as possible along each branch before backtracking.

Algorithm:

1. Start at the root (or an arbitrary node).
2. Visit the node and mark it as visited.
3. Recursively visit all unvisited adjacent nodes.
4. Backtrack when no more unvisited adjacent nodes are available.

Example:




Steps for DFS starting from node A:

1. Start at A. Mark A as visited.
2. Move to B (adjacent to A). Mark B as visited.
3. Move to D (adjacent to B). Mark D as visited. D has no unvisited adjacent nodes, so backtrack to B.
4. Backtrack to A. Move to C (adjacent to A). Mark C as visited.
5. Move to E (adjacent to C). Mark E as visited. E has no unvisited adjacent nodes, so backtrack to C.

DFS Traversal Order:

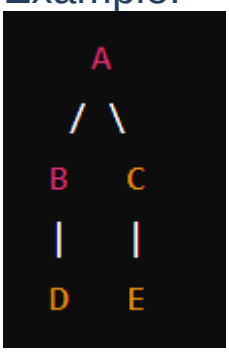
- A → B → D → C → E

BFS (Breadth-First Search) is a graph traversal algorithm that explores all neighbors at the present depth prior to moving on to nodes at the next depth level.

Algorithm:

1. Start at the root (or an arbitrary node).
2. Use a queue to keep track of nodes to visit.
3. Visit the node at the front of the queue and mark it as visited.
4. Enqueue all unvisited adjacent nodes.
5. Dequeue the next node and repeat until the queue is empty.

Example:



Steps for BFS starting from node A:

1. Start at A. Mark A as visited and enqueue it.
2. Dequeue A. Visit its neighbors B and C, mark them as visited, and enqueue them.
3. Dequeue B. Visit its neighbor D, mark D as visited, and enqueue it.
4. Dequeue C. Visit its neighbor E, mark E as visited, and enqueue it.
5. Dequeue D. D has no unvisited adjacent nodes.
6. Dequeue E. E has no unvisited adjacent nodes.

BFS Traversal Order:

- A → B → C → D → E

**Ques 08. Write recurrence relations for matrixmultiplication using Strassen's method and solve it using the Master method.**

## Solution:

Strassen's algorithm improves on the standard matrix multiplication algorithm by reducing the number of multiplications required. For two $n \times n$ matrices, Strassen's algorithm divides each matrix into four submatrices and performs seven recursive multiplications along with some additions and subtractions.

**Recurrence Relation:**

For matrix multiplication using Strassen's method:

- The standard matrix multiplication involves $O(n^3)$ operations.
- Strassen's method reduces the number of multiplications from 8 (as in the standard divide-and-conquer approach) to 7.

Let $T(n)$ be the time complexity of multiplying two $n \times n$ matrices using Strassen's method.

The recurrence relation for Strassen's algorithm is: $T(n) = 7T\left(\frac{n}{2}\right) + O(n^2)$

where:

- $7T\left(\frac{n}{2}\right)$ is due to the seven recursive multiplications of $\frac{n}{2} \times \frac{n}{2}$ matrices.
- $O(n^2)$ accounts for the addition and subtraction operations performed on matrices of size $n \times n$.

To solve the recurrence relation using the Master Method, we use the general form: $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

For Strassen's algorithm:

- $a = 7$
- $b = 2$
- $f(n) = O(n^2)$

The Master Theorem provides a way to determine the asymptotic complexity based on these parameters.

**Master Theorem:**

For a recurrence of the form $T(n) = aT\left(\frac{n}{b}\right) + f(n)$:

- Compare $f(n)$ with $n^{\log_b a}$.

Calculate $\log_b a$: $\log_b a = \log_2 7 \approx 2.81$

Now compare $f(n)$ to $n^{\log_b a}$:

- $f(n) = O(n^2)$
- $n^2$ grows slower than $n^{2.81}$

According to the Master Theorem:

- If $f(n) = O(n^c)$ where $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

In this case:

- $c = 2$ and $\log_b a \approx 2.81$, so $c < \log_b a$.

Thus: $T(n) = \Theta(n^{\log_2 7})$

The time complexity of matrix multiplication using Strassen's method is $\Theta(n^{\log_2 7})$, where $\log_2 7 \approx 2.81$.