1. Computing Binomial Coefficient

Here is a simple recursive implementation of the binomial coefficient:

```
python
Copy code
def binomial_coefficient(n, k):
    if k == 0 or k == n:
        return 1
    return binomial_coefficient(n-1, k-1) + binomial_coefficient(n-1, k)

# Example usage:
n = 5
k = 2
print(f"Binomial Coefficient C({n},{k}) = {binomial_coefficient(n, k)}")
```

2. Bellman-Ford Algorithm

Here's a straightforward implementation of the Bellman-Ford algorithm:

```
python
Copy code
def bellman_ford(graph, V, src):
    dist = [float('inf')] * V
    dist[src] = 0

    for _ in range(V - 1):
        for u, v, w in graph:
            if dist[u] != float('inf') and dist[u] + w < dist[v]:
                dist[v] = dist[u] + w

    # Check for negative-weight cycles
    for u, v, w in graph:
        if dist[u] != float('inf') and dist[u] + w < dist[v]:
            print("Graph contains negative weight cycle")
            return

    print("Vertex Distance from Source")
    for i in range(V):
        print(f"{i}\t\t{dist[i]}")

# Example usage:
graph = [
    (0, 1, 1),
    (1, 2, 3),
    (2, 3, 2),
    (3, 1, -6)
]
V = 4
src = 0
bellman_ford(graph, V, src)
```

3. Floyd-Warshall Algorithm

Here's a simple implementation of the Floyd-Warshall algorithm:

```
python
```

```
Copy code
def floyd_warshall(graph):
    V = len(graph)
    dist = list(map(lambda i: list(map(lambda j: j, i)), graph))

    for k in range(V):
        for i in range(V):
            for j in range(V):
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])

    print_solution(dist)

def print_solution(dist):
    V = len(dist)
    print("Shortest distances between every pair of vertices")
    for i in range(V):
        for j in range(V):
            if dist[i][j] == float('inf'):
                print("INF", end=" ")
            else:
                print(dist[i][j], end=" ")
            if j == V - 1:
                print("")

# Example usage:
graph = [
    [0, 3, float('inf'), 7],
    [8, 0, 2, float('inf')],
    [5, float('inf'), 0, 1],
    [2, float('inf'), float('inf'), 0]
]
floyd_warshall(graph)
```

4. Meet in the Middle Technique

Here's a simple implementation of the Meet in the Middle technique for finding a subset that sums to a target value:

```
python
Copy code
def meet_in_the_middle(arr, target):
    from itertools import combinations

    def subset_sums(arr):
        sums = []
        for i in range(len(arr) + 1):
            for comb in combinations(arr, i):
                sums.append(sum(comb))
        return sums

    n = len(arr)
    left_sums = subset_sums(arr[:n//2])
    right_sums = subset_sums(arr[n//2:])

    left_sums_set = set(left_sums)

    for s in right_sums:
        if (target - s) in left_sums_set:
            return True
    return False
```

```
# Example usage:
arr = [1, 2, 3, 4, 5]
target = 9
print(f"Subset with sum {target} exists: {meet_in_the_middle(arr,
target)}")
```