3.13 FINDING CLOSEST SUBSET SUM USING MEET IN THE MIDDLE

Question:

Write a program to implement Meet in the Middle Technique. Given an array of integers and a target sum, find the subset whose sum is closest to the target. You will use the Meet in the Middle technique to efficiently find this subset.

AIM

To implement the Meet in the Middle algorithm that finds the subset of an array whose sum is closest to a given target value.

ALGORITHM

1. Split the array into two halves: left and right.
2. Generate all possible subset sums for both halves.
3. Sort one half (say, right_sums) to enable binary search.
4. For each sum in left_sums0, use binary search to find the sum in right_sums that brings the total closest to the target.
5. Track the minimum absolute difference and corresponding sum.

PROGRAM

```
from itertools import combinations

def closest_subset_sum(arr, target):
    n = len(arr)
    left = arr[:n//2]
    right = arr[n//2:]

    def subset_sums(nums):
        return [sum(comb) for r in range(len(nums)+1) for comb in combinations(nums, r)]

    left_sums = subset_sums(left)
    right_sums = subset_sums(right)
    right_sums.sort()

    closest = float('inf')
    for s in left_sums:
        rem = target - s
        lo, hi = 0, len(right_sums)-1
        while lo <= hi:
            mid = (lo + hi) // 2
            if right_sums[mid] < rem:
                lo = mid + 1
            else:
                hi = mid - 1
        for i in [hi, lo]:
            if 0 <= i < len(right_sums):
                total = s + right_sums[i]
                if abs(target - total) < abs(target - closest):
                    closest = total
    return closest

def run_closest_sum():
    arr = list(map(int, input("Enter array: ").split()))
    target = int(input("Enter target sum: "))
    print("Closest subset sum:", closest_subset_sum(arr, target))

run_closest_sum()
```

Input:

[76,34,12,8,64,83] , 41

Output:

```
Enter array: 76 34 12 8 64 83
Enter target sum: 41
Closest subset sum: 42
>>>
```

RESULT:

Thus, program is successfully executed and the output is verified.

PERFORMANCE ANALYSIS:

· Time Complexity: $O(2^{\{n/2\}} \backslash log(2^{\{n/2\}}))$

· Space Complexity: $O(2^{\{n/2\}})$