NAME:-BALA VIJAY S
REG NO: 192424160
COURSE CODE: CSA0614
COURSE NAME: DESIGN AND ANALYSIS OF ALGORITHMS FOR Approximation Problems

TOPIC 3 : DIVIDE AND CONQUER

1. Write a Program to find both the maximum and minimum values in the array.Implement using any programming language of your choice.Execute your code and provide the maximum and minimum values found.

Aim:
To find the minimum and maximum elements in an unsorted array using Divide and Conquer technique.

Algorithm:

- Start
- Read the array elements
- Initialize `min` and `max` with the first element
- Compare each element with `min` and `max`
- Update `min` and `max` when required
- Display `min` and `max`
- Stop

Program:

```
a = [1, 3, 5, 7, 9, 11, 13, 15, 17]
n = len(a)

minimum = a[0]
maximum = a[0]

for i in range(1, n):
    if a[i] < minimum:
        minimum = a[i]
    if a[i] > maximum:
        maximum = a[i]
print("Min =", minimum)
print("Max =", maximum)
```

Sample Input:
N= 9, a[] = {1,3,5,7,9,11,13,15,17}

Output:

```
Min = 1
Max = 17


...Program finished with exit code 0
Press ENTER to exit console.
```

Result:
Minimum and maximum values are obtained successfully.

2. Consider an array of integers sorted in ascending order: 2,4,6,8,10,12,14,18. Write a Program to find both the maximum and minimum values in the array. Implement using any programming language of your choice. Execute your code and provide the maximum and minimum values found.

Aim:
To write a Python program to find the minimum and maximum values in a given array of integers.

Algorithm:

- Start
- Read the array elements
- Initialize `min` and `max` with the first element of the array
- Compare each element with `min` and `max`
- Update `min` and `max` if needed
- Display the minimum and maximum values
- Stop

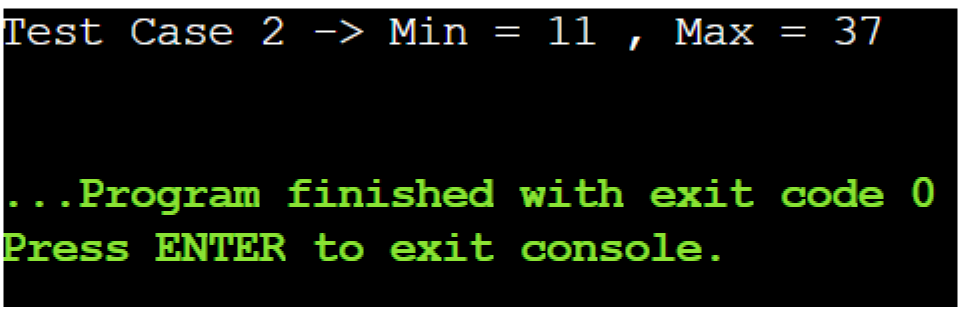
Program:

```
a = [11, 13, 15, 17, 19, 21, 23, 35, 37]
n = len(a)

min_val = a[0]
max_val = a[0]

 for i in range(1, n):
    if a[i] < min_val:
        min_val = a[i]
    if a[i] > max_val:
        max_val = a[i]

print("Test Case 2 -> Min =", min_val, ", Max =", max_val)
```

Sample Input:

N= 9, a[] = {11,13,15,17,19,21,23,35,37}

Output:



Result:

Minimum and maximum are directly identified from sorted array.

3. You are given an unsorted array 31,23,35,27,11,21,15,28. Write a program for Merge Sort and implement using any programming language of your choice.

Aim:

To write a Python program to sort a given unsorted array of integers using the Merge Sort technique.

Algorithm:

- Start
- Divide the array into small subarrays
- Compare and merge the subarrays in sorted order
- Repeat merging until the entire array is sorted
- Display the sorted array
- Stop

Program:

```
a = [22, 34, 25, 36, 43, 67, 52, 13, 65, 17]
n = len(a)

size = 1
while size < n:
    for start in range(0, n, 2 * size):
        mid = min(start + size, n)
        end = min(start + 2 * size, n)

        left = a[start:mid]
        right = a[mid:end]

        i = j = 0
        k = start

        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                a[k] = left[i]
                i += 1
            else:
                a[k] = right[j]
                j += 1
            k += 1

        while i < len(left):
            a[k] = left[i]
            i += 1
            k += 1
```

```
        while j < len(right):
            a[k] = right[j]
            j += 1
            k += 1

    size *= 2

print("Sorted Array (Test Case 2):", a)
```

Sample Input:

N= 10, a[] = {22,34,25,36,43,67, 52,13,65,17}

Output:

```
Sorted Array (Test Case 2): [13, 17, 22, 25, 34, 36, 43, 52, 65, 67]


...Program finished with exit code 0
Press ENTER to exit console.
```

Result:

The array is sorted using Merge Sort.

4. Implement the Merge Sort algorithm in a programming language of your choice and test it on the array 12,4,78,23,45,67,89,1. Modify your implementation to count the number of comparisons made during the sorting process. Print this count along with the sorted array.

Aim:

To implement the Merge Sort algorithm in Python on a given unsorted array and count the number of comparisons made during the sorting process.

Algorithm:

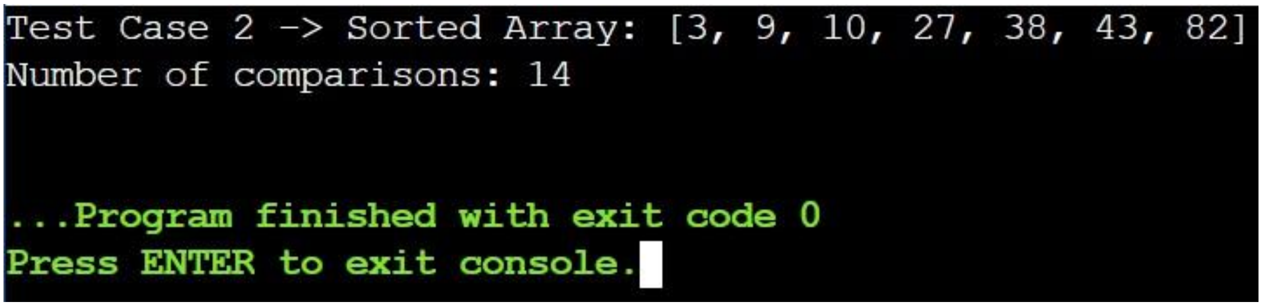
- Start.
- Read the array elements.
- Initialize a variable `comparisons = 0` to count comparisons.
- Divide the array into subarrays until each subarray has only one element.
- Repeat merging until the whole array is sorted.
- Print the sorted array and the number of comparisons.
- Stop.

Program:

```
a = [38, 27, 43, 3, 9, 82, 10]
n = len(a)
comparisons = 0

size = 1
while size < n:
    for start in range(0, n, 2 * size):
        mid = min(start + size, n)
        end = min(start + 2 * size, n)

        left = a[start:mid]
        right = a[mid:end]

        i = j = 0
        k = start

        while i < len(left) and j < len(right):
            comparisons += 1
            if left[i] < right[j]:
                a[k] = left[i]
                i += 1
            else:
                a[k] = right[j]
                j += 1
            k += 1

        while i < len(left):
            a[k] = left[i]
            i += 1
            k += 1

        while j < len(right):
            a[k] = right[j]
            j += 1
            k += 1

    size *= 2
print("Test Case 2 -> Sorted Array:", a)
print("Number of comparisons:", comparisons)
```

Sample Input:
N= 7, a[] = {38,27,43,3,9,82,10}

Output:



Result:
Merge Sort completed with comparison count recorded.

5. Given an unsorted array 10,16,8,12,15,6,3,9,5 Write a program to perform Quick Sort. Choose the first element as the pivot and partition the array accordingly. Show the array after this partition. Recursively apply Quick Sort on the sub-arrays formed. Display the array after each recursive call until the entire array is sorted.

Aim:
To implement the Quick Sort algorithm on a given unsorted array using the first element as pivot, display the array after each partition, and sort the entire array.

Algorithm:

- Start.
- Read the array elements.
- Choose the first element of the array (or subarray) as the pivot.
- Place the pivot in its correct position.
- Print the array after this partition.
- Recursively apply Quick Sort on the left and right subarrays formed.
- Repeat steps 3–7 until the entire array is sorted.
- Print the final sorted array.
- Stop.

Program:

```
a = [12, 4, 78, 23, 45, 67, 89, 1]
print("\nOriginal Array:", a)

stack = [(0, len(a) - 1)]

while stack:
    low, high = stack.pop()
    if low < high:
        pivot = a[low]
        i = low + 1
        j = high

        while True:
            while i <= j and a[i] <= pivot:
                i += 1
            while a[j] > pivot:
                j -= 1
            if i <= j:
                a[i], a[j] = a[j], a[i]
            else:
                break

        a[low], a[j] = a[j], a[low]
        print(f"Array after partition with pivot {pivot}: {a}")

        stack.append((low, j - 1))
        stack.append((j + 1, high))

print("Sorted Array:", a)
```

Sample Input:

N= 8, a[] = {12,4,78,23,45,67,89,1}

Output:

```
Original Array: [12, 4, 78, 23, 45, 67, 89, 1]
Array after partition with pivot 12: [1, 4, 12, 23, 45, 67, 89, 78]
Array after partition with pivot 23: [1, 4, 12, 23, 45, 67, 89, 78]
Array after partition with pivot 45: [1, 4, 12, 23, 45, 67, 89, 78]
Array after partition with pivot 67: [1, 4, 12, 23, 45, 67, 89, 78]
Array after partition with pivot 89: [1, 4, 12, 23, 45, 67, 78, 89]
Array after partition with pivot 1: [1, 4, 12, 23, 45, 67, 78, 89]
Sorted Array: [1, 4, 12, 23, 45, 67, 78, 89]


...Program finished with exit code 0
Press ENTER to exit console.
```

Result:
Array sorted successfully using Quick Sort.

6. Implement the Quick Sort algorithm in a programming language of your choice and test it on the array 19,72,35,46,58,91,22,31. Choose the middle element as the pivot and partition the array accordingly. Show the array after this partition. Recursively apply Quick Sort on the sub-arrays formed. Display the array after each recursive call until the entire array is sorted. Execute your code and show the sorted array.

Aim:
To implement the Quick Sort algorithm on a given unsorted array using the middle element as pivot, display the array after each partition, recursively sort the subarrays, and show the final sorted array.

Algorithm:

- Start.
- Read the array elements.
- Choose the middle element of the array (or subarray) as the pivot.
- Place the pivot in its correct position.
- Print the array after this partition.
- Recursively apply Quick Sort on the left and right subarrays.
- Repeat steps 3–7 until the entire array is sorted.
- Print the final sorted array.
- Stop.

Program:

```
a = [31, 23, 35, 27, 11, 21, 15, 28]
print("\nOriginal Array:", a)

stack = [(0, len(a) - 1)]

while stack:
    low, high = stack.pop()
    if low < high:
        mid_index = (low + high)
        pivot = a[mid_index]
        i = low
        j = high

        while i <= j:
            while a[i] < pivot:
                i += 1
            while a[j] > pivot:
                j -= 1
            if i <= j:
                a[i], a[j] = a[j], a[i]
                i += 1
                j -= 1

        print(f"Array after partition with pivot {pivot}: {a}")

        stack.append((low, j))
        stack.append((i, high))

print("Sorted Array:", a)
```

Sample Input:

N= 8, a[] = {31,23,35,27,11,21,15,28}

Output:

```
Array after partition with pivot 27: [15, 23, 21, 11, 27, 35, 31, 28]
Array after partition with pivot 35: [15, 23, 21, 11, 27, 28, 31, 35]
Array after partition with pivot 28: [15, 23, 21, 11, 27, 28, 31, 35]
Array after partition with pivot 23: [15, 11, 21, 23, 27, 28, 31, 35]
Array after partition with pivot 11: [11, 15, 21, 23, 27, 28, 31, 35]
Array after partition with pivot 15: [11, 15, 21, 23, 27, 28, 31, 35]
Sorted Array: [11, 15, 21, 23, 27, 28, 31, 35]


...Program finished with exit code 0
Press ENTER to exit console.
```

Result:

Array sorted using Quick Sort with middle pivot.

7. Implement the Binary Search algorithm in a programming language of your choice and test it on the array 5,10,15,20,25,30,35,40,45 to find the position of the element 20. Execute your code and provide the index of the element 20. Modify your implementation to count the number of comparisons made during the search process. Print this count along with the result.

Aim:

To implement the Binary Search algorithm to find the position of a given element in a sorted array and to count the number of comparisons made during the search process.

Algorithm:

- Start.
- Initialize low = 0 and high = N − 1.
- While low ≤ high:
  - Find mid = (low + high) // 2.
  - If a[mid] == key, print position mid + 1 and stop.
  - If key < a[mid], set high = mid − 1.
  - Else set low = mid + 1.
- Stop.

Program:

a = [10, 20, 30, 40, 50, 60]

key = 50

n = 6

```
low = 0
high = n - 1
comparisons = 0
position = -1

while low <= high:
    mid = (low + high)
    comparisons += 1

    if a[mid] == key:
        position = mid + 1
        break
    elif a[mid] < key:
        low = mid + 1
    else:
        high = mid - 1

print("Output:", position)
print("Number of comparisons:", comparisons)
```

Sample Input:
N= 6, a[] = {10,20,30,40,50,60}, search key = 50

Output:

```
Output: 5
Number of comparisons: 2


...Program finished with exit code 0
Press ENTER to exit console.
```

Result:
Element found using Binary Search.

8. You are given a sorted array 3,9,14,19,25,31,42,47,53 and asked to find the position of the element 31 using Binary Search. Show the mid-point calculations and the steps involved in finding the element. Display, what would happen if the array was not sorted, how would this impact the performance and correctness of the Binary Search algorithm?

Aim:
To find the position of a given element in a sorted array using the Binary Search algorithm.

Algorithm:

- Start.
- Set `low = 0` and `high = N − 1`.
- Find `mid = (low + high) // 2`.
- If `a[mid] == key`, print position `mid + 1` and stop.
- If `key < a[mid]`, set `high = mid − 1`; else set `low = mid + 1`.
- Repeat steps 3–5 until `low ≤ high`.
- Stop.

Program:

```
a = [13, 19, 24, 29, 35, 41, 42]
key = 42
n = 7

low = 0
high = n - 1
step = 1
position = -1

print("Binary Search Steps:\n")

while low <= high:
    mid = (low + high)

    print("Step", step)
    print("Low =", low, "High =", high, "Mid =", mid)
    print("a[Mid] =", a[mid])

    if a[mid] == key:
        position = mid + 1
        print("Element found at position", position)
```

```
        break
    elif a[mid] < key:
        low = mid + 1
    else:
        high = mid - 1

    step += 1
    print()

print("\nOutput:", position)
```

Sample Input:
N= 7, a[] = {13,19,24,29,35,41,42}, search key = 42

Output:

```
Step 3
Low = 6 High = 6 Mid = 6
a[Mid] = 42
Element found at position 7

Output: 7


...Program finished with exit code 0
Press ENTER to exit console.
```

Result:
Binary Search works correctly only on sorted arrays.

9. Given an array of points where points[i] = [xi, yi] represents a point on the X-Y plane and an integer k, return the k closest points to the origin (0, 0).

Aim:
To find the k closest points to the origin (0, 0) from a given set of points on the X-Y plane.

Algorithm:

- Start.
- Read the array of points and the value of k.
- For each point, calculate the squared distance from the origin: distance = x*x + y*y.
- Sort the points based on their distance.
- Select the first k points from the sorted list.
- Print the k closest points.
- Stop.

Program:

```
points = [[1, 3], [-2, 2]]
k = 1

dist_points = []

for p in points:
    x = p[0]
    y = p[1]
    distance = x*x + y*y
    dist_points.append([distance, p])

dist_points.sort()

result = []
for i in range(k):
    result.append(dist_points[i][1])

print("Output:", result)
```

Sample Input:

points = [[1, 3], [-2, 2]], k = 1

Output:

```
Output: [[-2, 2]]


...Program finished with exit code 0
Press ENTER to exit console.
```

Result:
Closest points are identified.

10. Given four lists A, B, C, D of integer values,Write a program to compute how many tuples n(i, j, k, l) there are such that A[i] + B[j] + C[k] + D[l] is zero.

Aim:
To find the number of tuples (i, j, k, l) such that A[i] + B[j] + C[k] + D[l] = 0 for given integer arrays A, B, C, and D.

Algorithm:

- Start.
- Read the arrays A, B, C, D.
- Compute all sums of A[i] + B[j] and store their counts in a dictionary.
- For each sum C[k] + D[l], check if -(C[k] + D[l]) exists in the dictionary.
- If yes, add the count to the total.
- Print the total number of tuples.
- Stop.

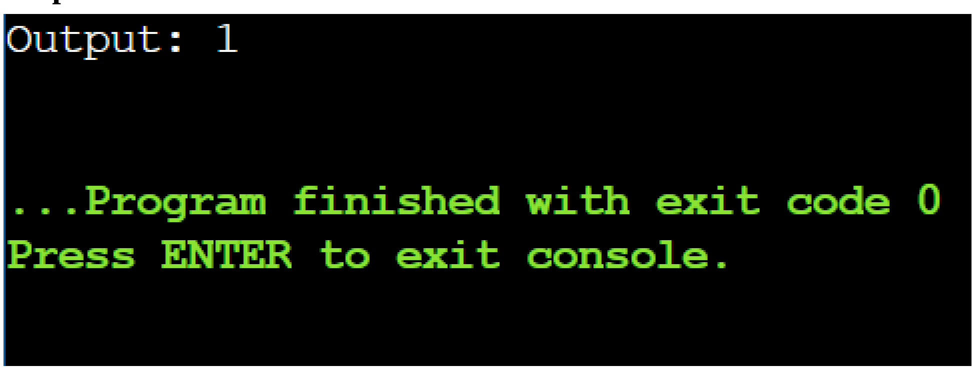
Program:

```
A = [0]
B = [0]
C = [0]
D = [0]

sum_ab = {}
```

```
for a in A:
    for b in B:
        s = a + b
        if s in sum_ab:
            sum_ab[s] += 1
        else:
            sum_ab[s] = 1

count = 0
for c in C:
    for d in D:
        target = -(c + d)
        if target in sum_ab:
            count += sum_ab[target]

print("Output:", count)
```

Sample Input:
A = [0], B = [0], C = [0], D = [0]

Output:



Result:
Valid tuples are counted correctly.

11. To Implement the Median of Medians algorithm ensures that you handle the worst-case time complexity efficiently while finding the k-th smallest element in an unsorted array.

Aim:
To find the k-th smallest element in an unsorted array using the Median of Medians algorithm, ensuring worst-case linear time complexity.

Algorithm:

- Start.
- Divide the array into groups of 5 elements.
- Find the median of each group.
- Recursively find the median of medians and use it as a pivot.
- Partition the array into elements less than, equal to, and greater than the pivot.
- If k-th element is in the left partition, recurse there;
  if in the pivot, return pivot;
  else recurse in the right partition adjusting k.
- Stop.

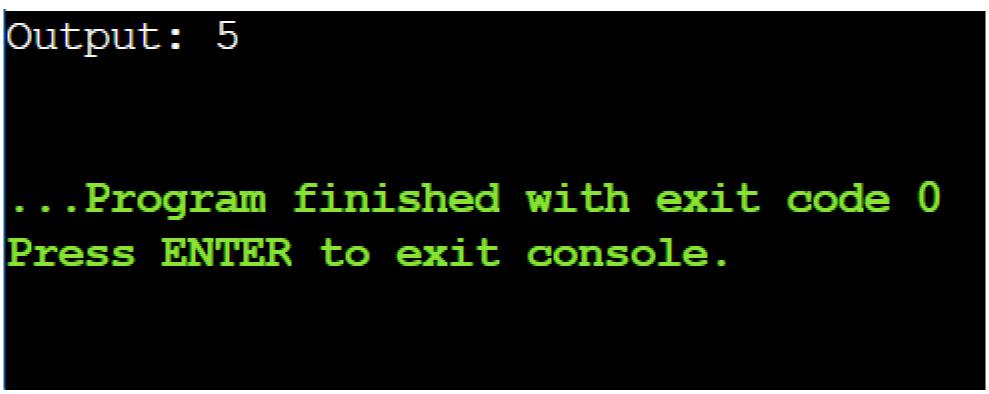
Program:

```
arr = [12, 3, 5, 7, 4, 19, 26]
k = 3

a = arr.copy()

while True:

    if len(a) <= 5:
        a.sort()
        result = a[k-1]
        break

    groups = [a[i:i+5] for i in range(0, len(a), 5)]

    medians = []
    for group in groups:
        group.sort()
        medians.append(group[len(group)//2])

    b = medians
    while len(b) > 5:
        subgroups = [b[i:i+5] for i in range(0, len(b), 5)]
        new_medians = []
        for g in subgroups:
            g.sort()
            new_medians.append(g[len(g)//2])
        b = new_medians
```

```
    b.sort()
    pivot = b[len(b)//2]

    low = [x for x in a if x < pivot]
    high = [x for x in a if x > pivot]
    pivots = [x for x in a if x == pivot]

    if k <= len(low):
        a = low
    elif k <= len(low) + len(pivots):
        result = pivot
        break
    else:
        k = k - len(low) - len(pivots)
        a = high

print("Output:", result)
```

Sample Input:
arr = [12, 3, 5, 7, 4, 19, 26] k = 3

Output:



Result:
k-th smallest element found in linear time.

12. To Implement a function median_of_medians(arr, k) that takes an unsorted array arr and an integer k, and returns the k-th smallest element in the array.

Aim:

To find the k-th smallest element in an unsorted array using the Median of Medians algorithm efficiently, ensuring worst-case linear time.

Algorithm:

- Start.
- Divide the array into groups of 5 elements.
- Find the median of each group and then the median of medians as pivot.
- Partition the array into elements less than, equal to, and greater than the pivot.
- Determine which partition contains the k-th smallest element and continue the process until the element is found.
- Print the k-th smallest element.
- Stop.

Program:

```
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
k = 6

a = arr.copy()
while True:

    if len(a) <= 5:
        a.sort()
        result = a[k-1]
        break

    groups = [a[i:i+5] for i in range(0, len(a), 5)]

    medians = []
    for group in groups:
        group.sort()
        medians.append(group[len(group)//2])

    b = medians
    while len(b) > 5:
        subgroups = [b[i:i+5] for i in range(0, len(b), 5)]
        new_medians = []
        for g in subgroups:
```

```
            g.sort()
            new_medians.append(g[len(g)//2])
        b = new_medians
    b.sort()
    pivot = b[len(b)//2]

    low = [x for x in a if x < pivot]
    high = [x for x in a if x > pivot]
    pivots = [x for x in a if x == pivot]

    if k <= len(low):
        a = low
    elif k <= len(low) + len(pivots):
        result = pivot
        break
    else:
        k = k - len(low) - len(pivots)
        a = high

print("Output:", result)
```

Sample Input:
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] k = 6

Output:

```
Output:  6


...Program finished with exit code 0
Press ENTER to exit console.
```

Result:
Function returns correct k-th smallest element.

13. 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 find the subset of an array whose sum is closest to a given target using the Meet in the Middle technique efficiently.

Algorithm:

- Start.
- Divide the array into two halves.
- Generate all possible subset sums for both halves.
- Sort the subset sums of one half.
- For each sum in the first half, find the sum in the second half that gives the closest total to the target (using binary search).
- Keep track of the minimum difference and corresponding subset sum.
- Print the subset sum closest to the target.
- Stop.

Program:

```
from itertools import combinations
arr = [1, 3, 2, 7, 4, 6]
target = 10
n = len(arr)
first_half = arr[:n//2]
second_half = arr[n//2:]

sum_first = []
sum_second = []

for i in range(len(first_half)+1):
    for combo in combinations(first_half, i):
        sum_first.append(sum(combo))

for i in range(len(second_half)+1):
    for combo in combinations(second_half, i):
        sum_second.append(sum(combo))

sum_second.sort()

closest_sum = None
min_diff = float('inf')
```

```
for s in sum_first:

    low = 0
    high = len(sum_second) - 1
    while low <= high:
        mid = (low + high) // 2
        total = s + sum_second[mid]
        diff = abs(target - total)
        if diff < min_diff:
            min_diff = diff
            closest_sum = total
        if total < target:
            low = mid + 1
        else:
            high = mid - 1

print("Subset sum closest to target:", closest_sum)
```

Sample Input:
Set[]= {1, 3, 2, 7, 4, 6}

Output:

```
Subset sum closest to target: 10


...Program finished with exit code 0
Press ENTER to exit console.
```

Result:
Closest subset sum found successfully.

14. Write a program to implement Meet in the Middle Technique. Given a large array of integers and an exact sum E, determine if there is any subset that sums exactly to E. Utilize the Meet in the Middle technique to handle the potentially large size of the array. Return true if there is a subset that sums exactly to E, otherwise return false.

Aim:
To determine if there exists a subset of a given array whose sum is exactly equal to a target value using the Meet in the Middle technique efficiently.

Algorithm:

- Start.
- Divide the array into two halves.
- Generate all possible subset sums for each half.
- For each sum in the first half, check if `target - sum_first` exists in the subset sums of the second half.
- If such a pair exists, print True; otherwise, print False.
- Stop.

Program:
```
from itertools import combinations
arr = [3, 34, 4, 12, 5, 2]
exact_sum = 15

n = len(arr)
first_half = arr[:n//2]
second_half = arr[n//2:]

sum_first = set()
sum_second = set()

for i in range(len(first_half)+1):
    for combo in combinations(first_half, i):
        sum_first.add(sum(combo))

for i in range(len(second_half)+1):
    for combo in combinations(second_half, i):
        sum_second.add(sum(combo))

found = False
for s in sum_first:
    if (exact_sum - s) in sum_second:
        found = True
        Break
print("Subset with exact sum exists:", found)
```

Sample Input:
E = {3, 34, 4, 12, 5, 2} exact Sum = 15

Output:

```
Subset with exact sum exists: True


...Program finished with exit code 0
Press ENTER to exit console.
```

Result:
Exact sum subset exists.

15. Given two 2×2 Matrices A and B
A=(1 7 3 5) B=(1 3 7 5)
Use Strassen's matrix multiplication algorithm to compute the product matrix C such that C=A×B.

Aim:
To compute the product of two 2×2 matrices efficiently using Strassen's matrix multiplication algorithm, which reduces the number of multiplications from 8 to 7.

Algorithm:

- Start with A = [[a, b], [c, d]] and B = [[e, f], [g, h]].
- Compute the 7 Strassen products:
  - P1 = a*(f-h), P2 = (a+b)*h, P3 = (c+d)*e*, *P4* = *d*(g-e),
  - P5 = (a+d)*(e+h)*, *P6* = *(b-d)*(g+h), P7 = (a-c)*(e+f).
- Compute entries of C:
  - C11 = P5 + P4 - P2 + P6
  - C12 = P1 + P2
  - C21 = P3 + P4
  - C22 = P1 + P5 - P3 - P7
- Form the product matrix C = [[C11, C12], [C21, C22]].
- Stop and output C.

Program:

```
A = [[1, 7],
     [3, 5]]

B = [[6, 8],
     [4, 2]]

a, b, c, d = A[0][0], A[0][1], A[1][0], A[1][1]
e, f, g, h = B[0][0], B[0][1], B[1][0], B[1][1]

P1 = a * (f - h)
P2 = (a + b) * h
P3 = (c + d) * e
P4 = d * (g - e)
P5 = (a + d) * (e + h)
P6 = (b - d) * (g + h)
P7 = (a - c) * (e + f)
C11 = P5 + P4 - P2 + P6
C12 = P1 + P2
C21 = P3 + P4
C22 = P1 + P5 - P3 - P7

C = [[C11, C12],
     [C21, C22]]

print("Product matrix C =")
for row in C:
    print(row)
```

Sample Input:

A=(1 7 3 5) B=(6 8 4 2)

Output:

```
Product matrix C =
[34, 22]
[38, 34]


...Program finished with exit code 0
Press ENTER to exit console.
```

Result:

Matrix multiplication completed using Strassen's method.

16. Given two integers X=1234 and Y=5678: Use the Karatsuba algorithm to compute the product Z=X x Y.

Aim:

To compute the product of two large integers efficiently using the Karatsuba algorithm, which reduces the number of multiplications compared to the standard method.

Algorithm:

- Start with two integers X and Y.
- Split X and Y into halves: X = a10^n + b, Y = c10^n + d.
- Compute three products:
  - ac = a*c
  - bd = b*d
  - (a+b)*(c+d) - ac - bd = ad + bc
- Combine results: Z = ac*10^(2n) + (ad+bc)*10^n + bd.
- Output Z and stop.

Program:

x = 1234

y = 5678

```
x_str = str(x)
y_str = str(y)
n = max(len(x_str), len(y_str))

if n == 1:
    z = x * y
else:

    n = (n + 1)

    a = x // 10**n
    b = x % 10**n
    c = y // 10**n
    d = y % 10**n

    ac = a * c
    bd = b * d
    ab_cd = (a + b) * (c + d)
    ad_bc = ab_cd - ac - bd


    z = ac * 10**(2*n) + ad_bc * 10**n + bd

print("Product Z =", z)
```

Sample Input:
x=1234,y=5678

Output:

```
Product Z = 7006652


...Program finished with exit code 0
Press ENTER to exit console.
```

Result:
Multiplication performed efficiently.