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.

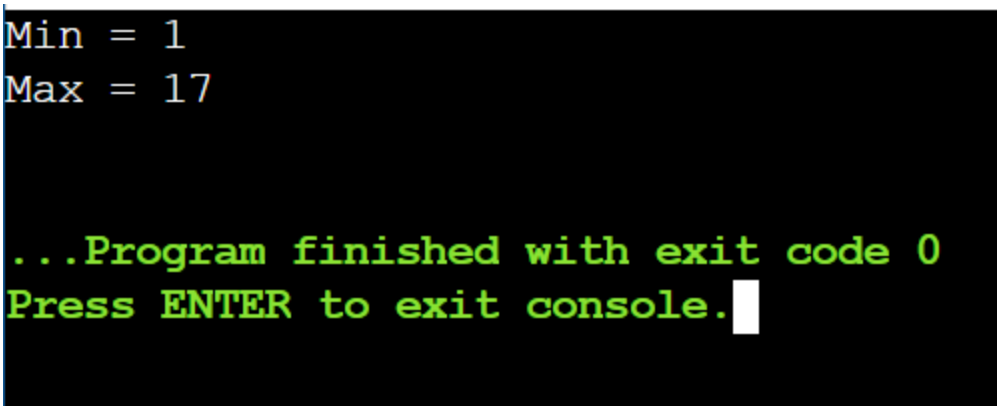
Program:

```
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:



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.

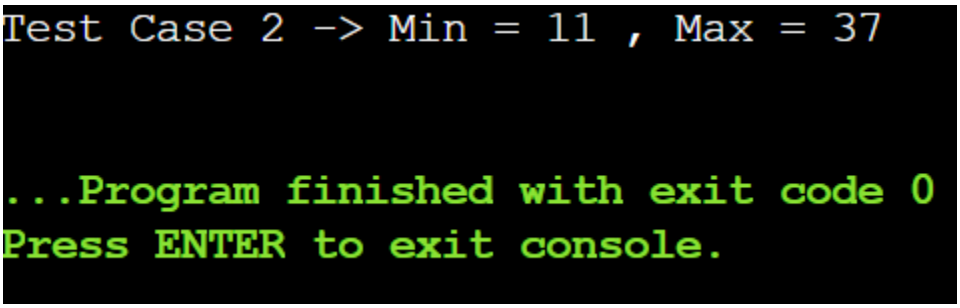
Program:

```
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:



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.
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.
```

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.

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:

```
Test Case 2 -> Sorted Array: [3, 9, 10, 27, 38, 43, 82]
Number of comparisons: 14


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

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.

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.
```

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.
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.
```

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.

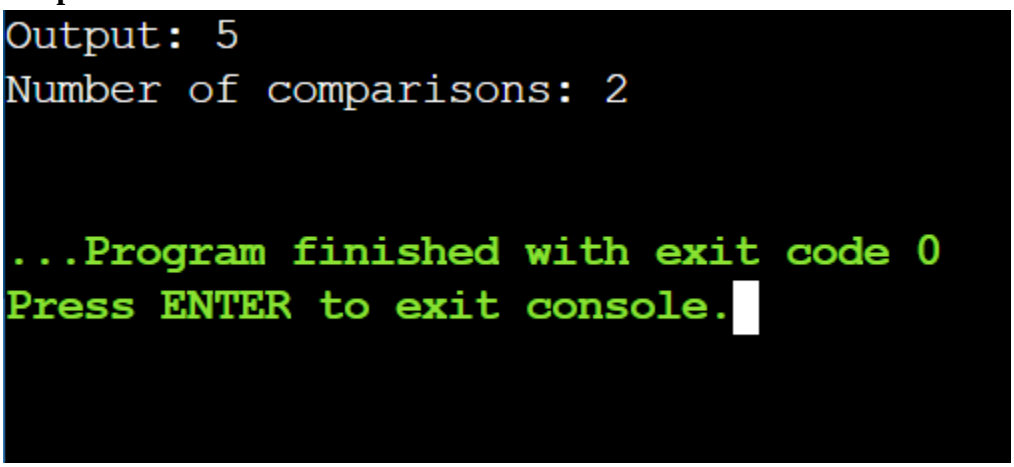
Program:

```
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:

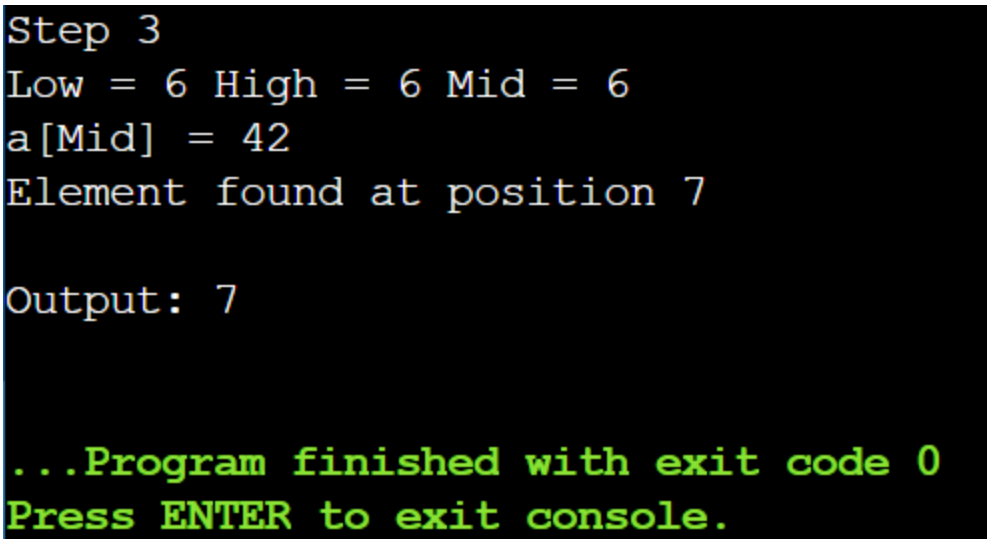


8.You are given a sorted array 3,9,14,19,25,31,42,47,53 and asked to find the position of the element 31 using Binary Search. Show the mid-point calculations and the steps involved in finding the element. Display, what would happen if the array was not sorted, how would this impact the performance and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



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).

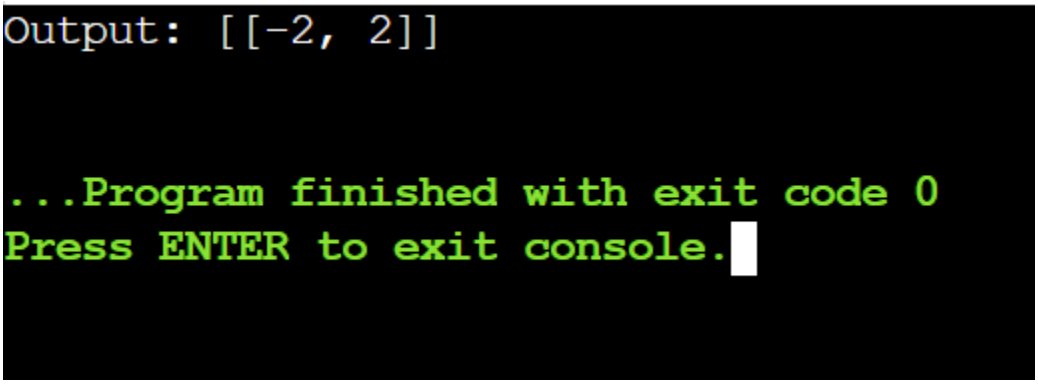
Program:

```
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:



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.

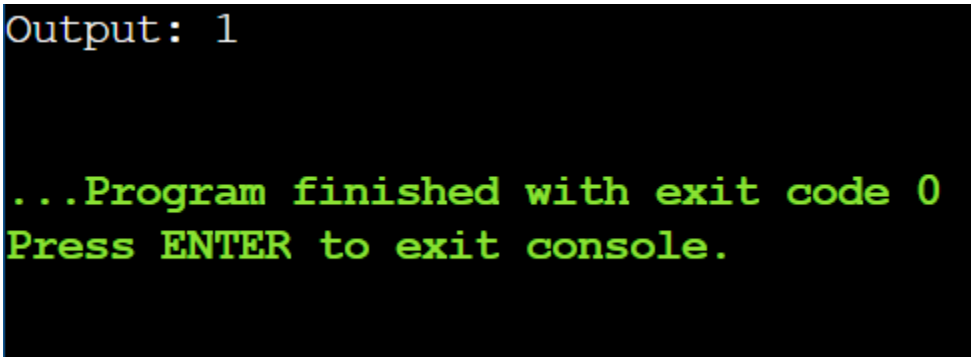
Program:

```
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:



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.

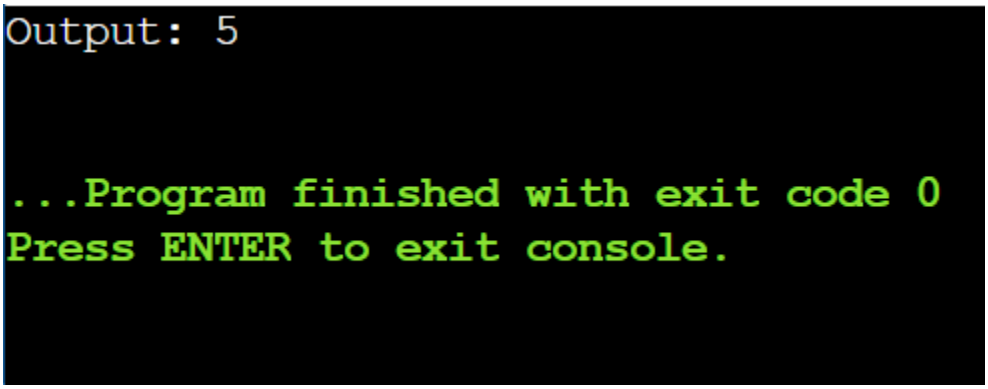
Program:

```
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:



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.

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.
```

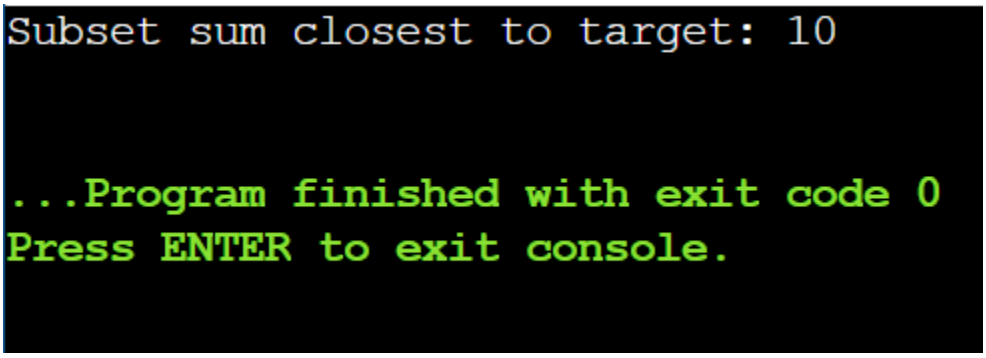
13.Write a program to implement Meet in the Middle Technique. Given an array of integers and a target sum, find the subset whose sum is closest to the target.
You will use the Meet in the Middle technique to efficiently find this subset.
Program:

```
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:



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.

Program:

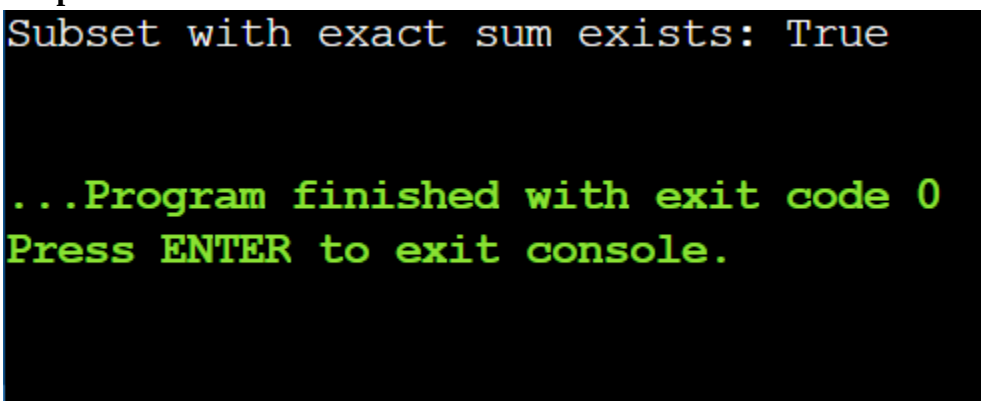
```
from itertools import combinations
arr = [3, 34, 4, 12, 5, 2]
exact_sum = 15
```

```
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:



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.

Program:

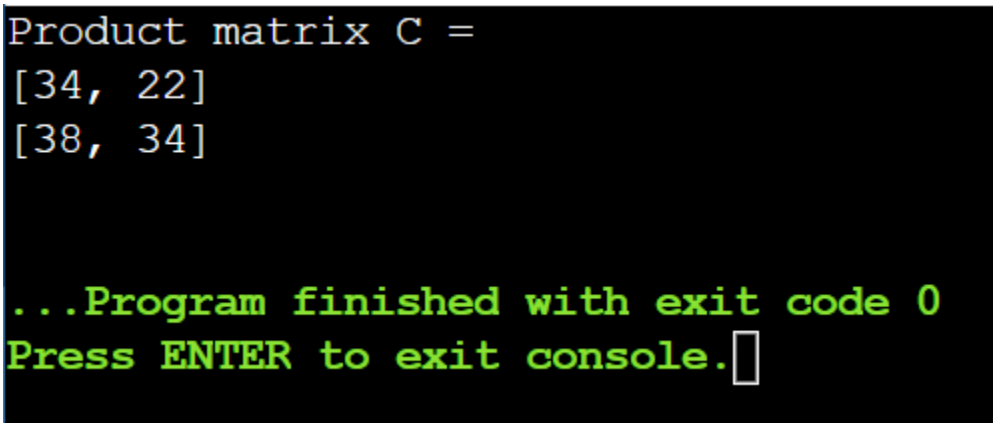
```
A = [[1, 7],
```

```
    [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:



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

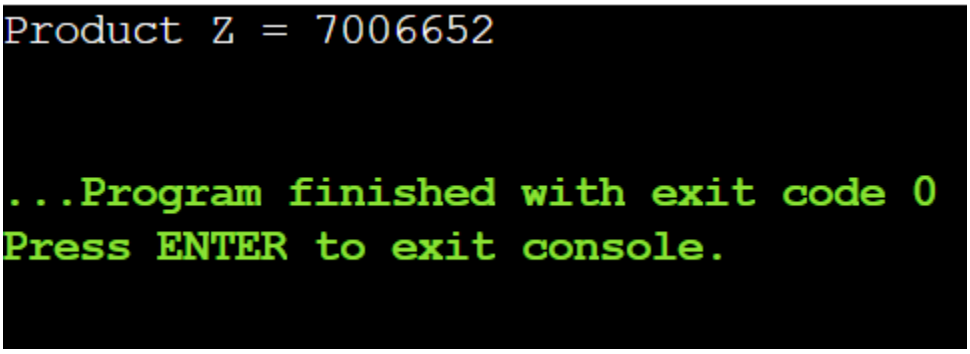
Program:

```
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.