1: Find Maximum and Minimum in Array

Aim:
To find both the maximum and minimum elements in an array.

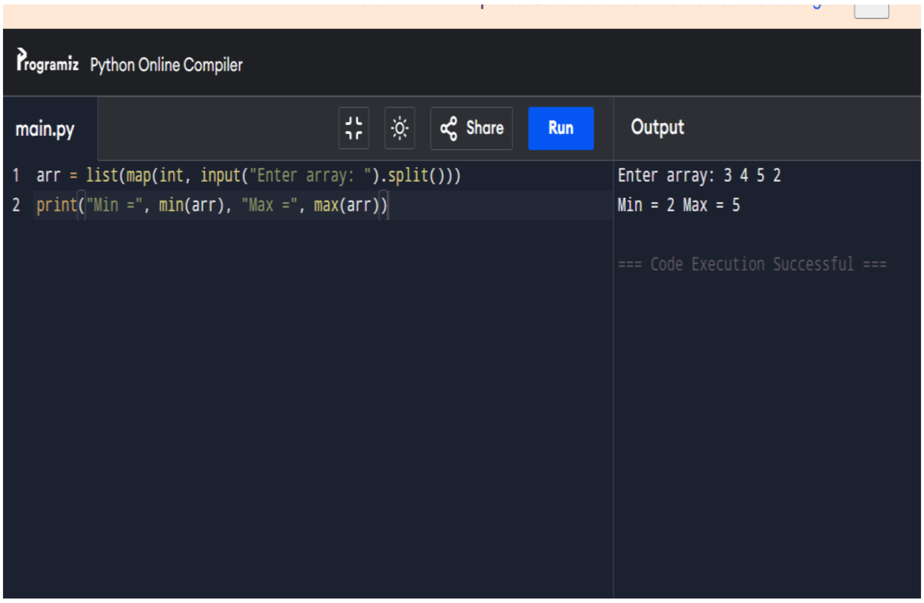
Algorithm:

1. Read size N and array elements.
2. Initialize min and max with first element.
3. Traverse array, update min if smaller and max if larger.
4. Print final values.

Code

```
arr = list(map(int, input("Enter array: ").split()))

 print("Min =", min(arr)) print("Max =", max(arr))
```

Output:



```
arr = list(map(int, input("Enter array: ").split()))
print("Min =", min(arr), "Max =", max(arr))
```

```
Enter array: 3 4 5 2
Min = 2 Max = 5

=== Code Execution Successful ===
```

Result: The program has been successfully executed.

2.Min and Max in Sorted Array

Aim:
To find min and max in a sorted array.

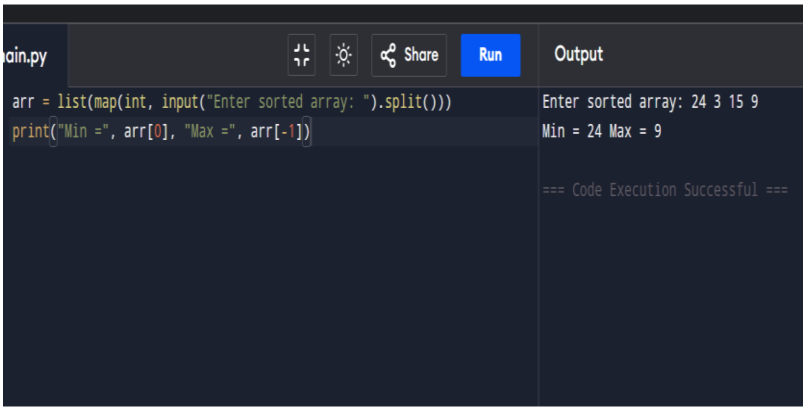
Algorithm:

1. In a sorted array, min = first element, max = last element.
2. Print both values.

Code:

```
arr = list(map(int, input("Enter sorted array: ").split()))

 print("Min =", arr[0]) print("Max =", arr[-1])
```

Output:



```
arr = list(map(int, input("Enter sorted array: ").split()))
print("Min =", arr[0], "Max =", arr[-1])
```

```
Enter sorted array: 24 3 15 9
Min = 24 Max = 9

=== Code Execution Successful ===
```

Result: The program has been successfully executed.

3: Merge Sort

Aim:
To sort an unsorted array using Merge Sort.

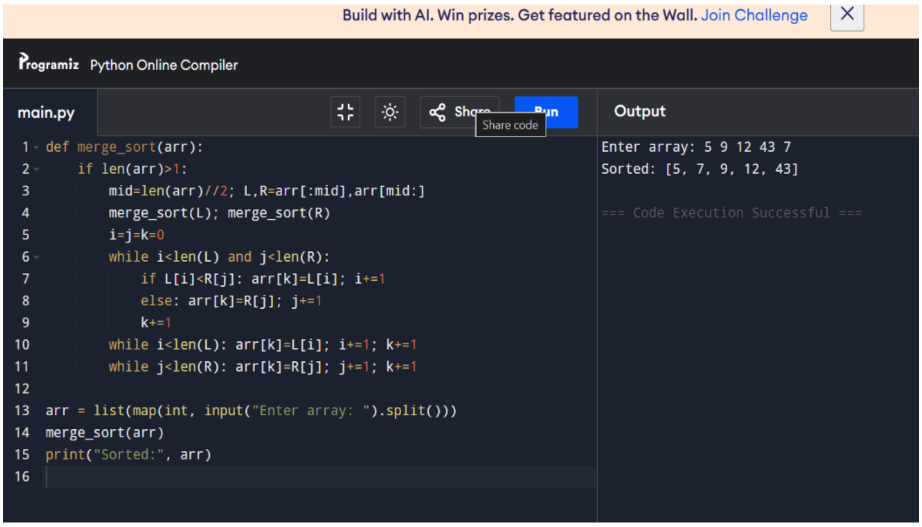
Algorithm:

1. Divide array into two halves.
2. Recursively sort both halves.
3. Merge sorted halves.

Code:

```
 def merge_sort(arr):

   if len(arr) > 1:

       mid = len(arr)//2

       L = arr[:mid]

       R = arr[mid:]

       merge_sort(L)

       merge_sort(R)

       i=j=k=0

       while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]; i+=1
            else:
                arr[k] = R[j]; j+=1
            k+=1
        while i < len(L):
            arr[k] = L[i]; i+=1; k+=1
        while j < len(R):
            arr[k] = R[j]; j+=1; k+=1
arr = list(map(int, input("Enter array: ").split()))
merge_sort(arr)
print("Sorted:", arr)
```

Output:


Build with AI. Win prizes. Get featured on the Wall. Join Challenge
Programiz Python Online Compiler
main.py
Share
Share code
Run
Output

```
def merge_sort(arr):
    if len(arr)>1:
        mid=len(arr)//2; L,R=arr[:mid],arr[mid:]
        merge_sort(L); merge_sort(R)
        i=j=k=0
        while i<len(L) and j<len(R):
            if L[i]<R[j]: arr[k]=L[i]; i+=1
            else: arr[k]=R[j]; j+=1
            k+=1
        while i<len(L): arr[k]=L[i]; i+=1; k+=1
        while j<len(R): arr[k]=R[j]; j+=1; k+=1

arr = list(map(int, input("Enter array: ").split()))
merge_sort(arr)
print("Sorted:", arr)
```

```
Enter array: 5 9 12 43 7
Sorted: [5, 7, 9, 12, 43]

=== Code Execution Successful ===
```

Result: The program has been successfully executed.

4: Merge Sort with Comparisons

Aim:

To sort array using Merge Sort and count comparisons.

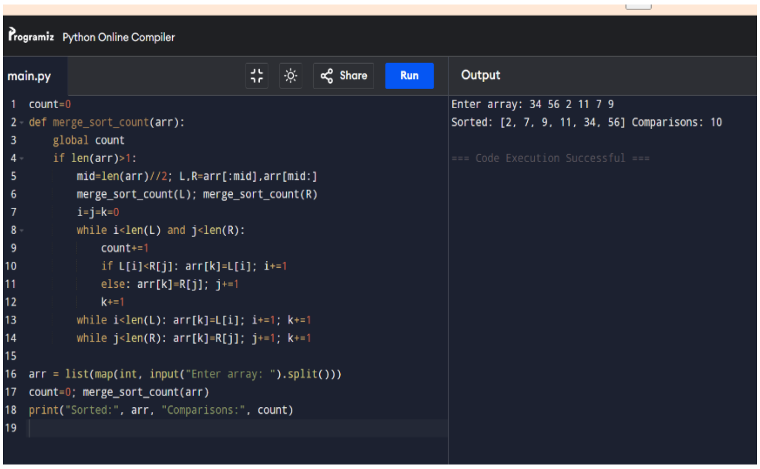
Algorithm:

1. In a sorted array, min = first element, max = last element.
2. Print both values.

Code:

```
comparisons = 0
def merge_sort(arr):
    global comparisons
    if len(arr) > 1:
        mid = len(arr)//2
        L, R = arr[:mid], arr[mid:]
        merge_sort(L)
        merge_sort(R)
        i=j=k=0
        while i < len(L) and j < len(R):
            comparisons += 1
            if L[i] < R[j]:
                arr[k] = L[i]; i+=1
            else:
                arr[k] = R[j]; j+=1
            k+=1
        while i < len(L): arr[k] = L[i]; i+=1; k+=1
        while j < len(R): arr[k] = R[j]; j+=1; k+=1
arr = list(map(int, input("Enter array: ").split()))
merge_sort(arr)
print("Sorted:", arr)
print("Comparisons:", comparisons)
```

Output:



```
count=0
def merge_sort_count(arr):
    global count
    if len(arr)>1:
        mid=len(arr)//2; L,R=arr[:mid],arr[mid:]
        merge_sort_count(L); merge_sort_count(R)
        i=j=k=0
        while i<len(L) and j<len(R):
            count+=1
            if L[i]<R[j]: arr[k]=L[i]; i+=1
            else: arr[k]=R[j]; j+=1
            k+=1
        while i<len(L): arr[k]=L[i]; i+=1; k+=1
        while j<len(R): arr[k]=R[j]; j+=1; k+=1

arr = list(map(int, input("Enter array: ").split()))
count=0; merge_sort_count(arr)
print("Sorted:", arr, "Comparisons:", count)
```

```
Enter array: 34 56 2 11 7 9
Sorted: [2, 7, 9, 11, 34, 56] Comparisons: 10

=== Code Execution Successful ===
```

Result: The program has been successfully executed.

5: Quick Sort (First Element Pivot)

Aim:

To sort array using Quick Sort with first element as pivot.

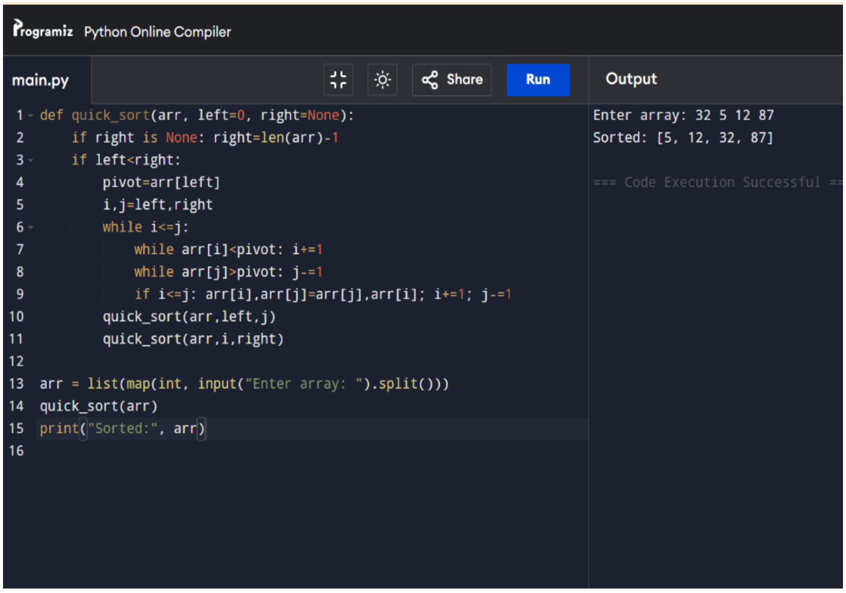
Algorithm:

1. Choose first element as pivot.
2. Partition array into < pivot and > pivot.
3. Recursively quicksort subarrays.

Code:

```
def quick_sort(arr):
  if len(arr) <= 1:
    return arr
  pivot = arr[0]
  left = [x for x in arr[1:] if x <= pivot]
  right = [x for x in arr[1:] if x > pivot]
  return quick_sort(left) + [pivot] + quick_sort(right)
arr = list(map(int, input("Enter array: ").split()))
print("Sorted:", quick_sort(arr))
```

Output:



```python
def quick_sort(arr, left=0, right=None):
    if right is None: right=len(arr)-1
    if left<right:
        pivot=arr[left]
        i,j=left,right
        while i<=j:
            while arr[i]<pivot: i+=1
            while arr[j]>pivot: j-=1
            if i<=j: arr[i],arr[j]=arr[j],arr[i]; i+=1; j-=1
        quick_sort(arr,left,j)
        quick_sort(arr,i,right)

arr = list(map(int, input("Enter array: ").split()))
quick_sort(arr)
print("Sorted:", arr)
```

```
Enter array: 32 5 12 87
Sorted: [5, 12, 32, 87]

=== Code Execution Successful ==
```

Result: The program has been successfully executed.

6: Quick Sort (Middle Pivot)

Aim:

To sort array using Quick Sort with middle element as pivot.

Algorithm:

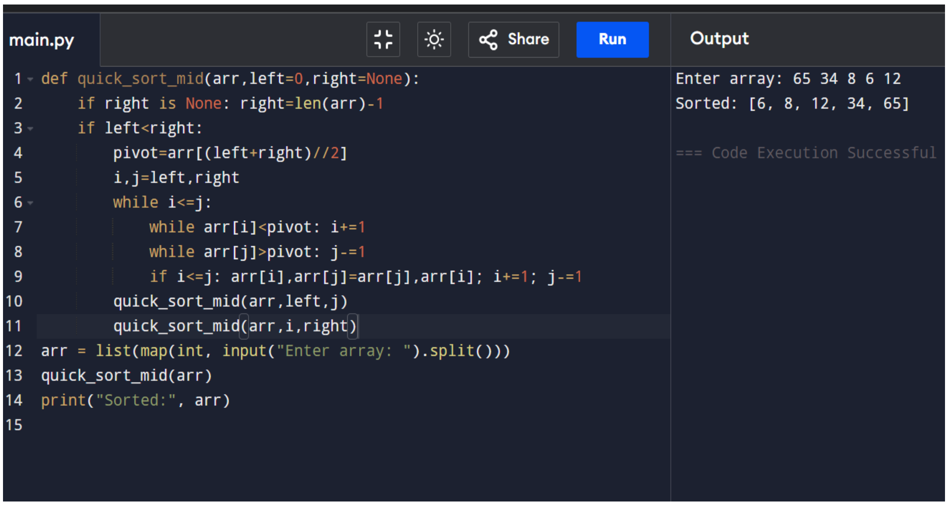
1. Choose middle element as pivot.
2. Partition into < pivot and > pivot.
3. Recursively sort subarrays.

Code:

```python
def quick_sort(arr):
  if len(arr) <= 1:
    return arr
  pivot = arr[len(arr)//2]
  left = [x for x in arr if x < pivot]
  middle = [x for x in arr if x == pivot]
  right = [x for x in arr if x > pivot]
  return quick_sort(left) + middle + quick_sort(right)
arr = list(map(int, input("Enter array: ").split()))
```

```
print("Sorted:", quick_sort(arr))
```

Output:



```
def quick_sort_mid(arr,left=0,right=None):
    if right is None: right=len(arr)-1
    if left<right:
        pivot=arr[(left+right)//2]
        i,j=left,right
        while i<=j:
            while arr[i]<pivot: i+=1
            while arr[j]>pivot: j-=1
            if i<=j: arr[i],arr[j]=arr[j],arr[i]; i+=1; j-=1
        quick_sort_mid(arr,left,j)
        quick_sort_mid(arr,i,right)
arr = list(map(int, input("Enter array: ").split()))
quick_sort_mid(arr)
print("Sorted:", arr)
```

```
Enter array: 65 34 8 6 12
Sorted: [6, 8, 12, 34, 65]

=== Code Execution Successful
```

Result: The program has been successfully executed.

7: Binary Search with Comparisons

Aim:

To implement Binary Search and count comparisons.

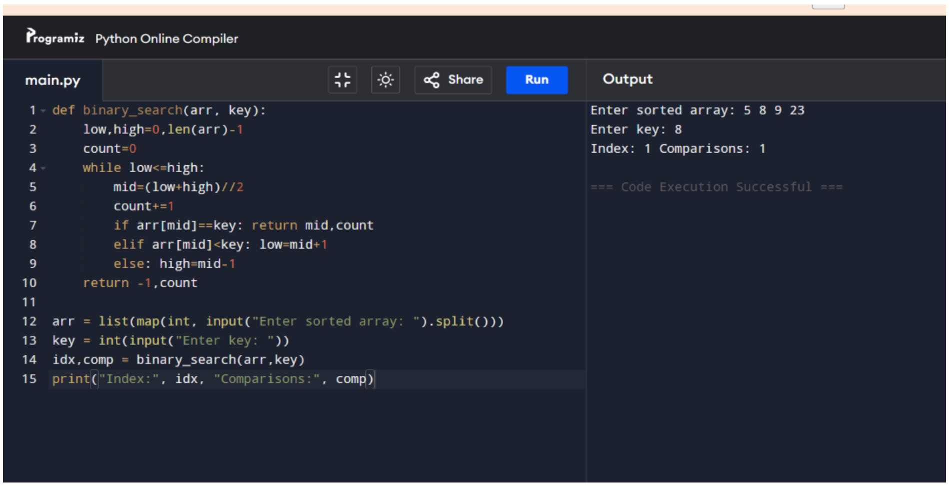
Algorithm:

1. Start with low=0, high=n-1.
2. Find mid, compare with key.
3. Narrow down search space.
4. Count comparisons.

Code:

```
def binary_search(arr, key):
  low, high = 0, len(arr)-1
  comparisons = 0
  while low <= high:
    comparisons += 1
    mid = (low + high)//2
    if arr[mid] == key:
      return mid+1, comparisons
    elif arr[mid] < key:
      low = mid+1
    else:
      high = mid-1
  return -1, comparisons
arr = list(map(int, input("Enter sorted array: ").split()))
key = int(input("Enter search key: "))
pos, comps = binary_search(arr, key)
print("Position:", pos)
print("Comparisons:", comps)
```

Output:



```
def binary_search(arr, key):
    low,high=0,len(arr)-1
    count=0
    while low<=high:
        mid=(low+high)//2
        count+=1
        if arr[mid]==key: return mid,count
        elif arr[mid]<key: low=mid+1
        else: high=mid-1
    return -1,count

arr = list(map(int, input("Enter sorted array: ").split()))
key = int(input("Enter key: "))
idx,comp = binary_search(arr,key)
print("Index:", idx, "Comparisons:", comp)
```

```
Enter sorted array: 5 8 9 23
Enter key: 8
Index: 1 Comparisons: 1

=== Code Execution Successful ===
```

Result: The program has been successfully executed.

8: Binary Search with Steps

Aim:

To perform Binary Search and show mid-point calculations.

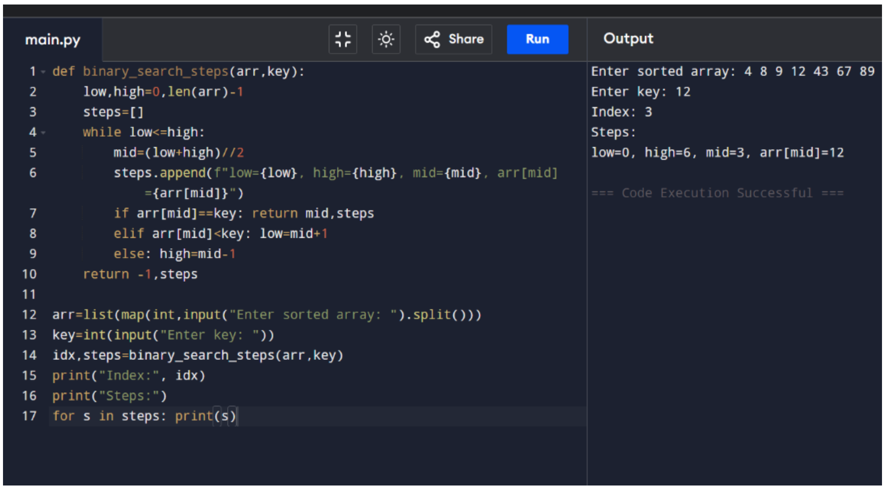
Algorithm:

1. Initialize low=0, high=n-1.
2. Compute mid = (low+high)//2.
3. If arr[mid] == key, stop.
4. If arr[mid] < key, move low=mid+1, else high=mid-1.
5. Print steps.

Code:

```
def binary_search(arr, key):
    low, high = 0, len(arr)-1
    while low <= high:
        mid = (low+high)//2
        print(f"Checking mid={mid+1}, value={arr[mid]}")
        if arr[mid] == key:
            return mid+1
        elif arr[mid] < key:
            low = mid+1
        else:
            high = mid-1
    return -1
arr = list(map(int, input("Enter sorted array: ").split()))
key = int(input("Enter search key: "))
pos = binary_search(arr, key)
print("Position:", pos)
```

Output:



```
def binary_search_steps(arr,key):
    low,high=0,len(arr)-1
    steps=[]
    while low<=high:
        mid=(low+high)//2
        steps.append(f"low={low}, high={high}, mid={mid}, arr[mid]
            ={arr[mid]}")
        if arr[mid]==key: return mid,steps
        elif arr[mid]<key: low=mid+1
        else: high=mid-1
    return -1,steps

arr=list(map(int,input("Enter sorted array: ").split()))
key=int(input("Enter key: "))
idx,steps=binary_search_steps(arr,key)
print("Index:", idx)
print("Steps:")
for s in steps: print(s)
```

```
Enter sorted array: 4 8 9 12 43 67 89
Enter key: 12
Index: 3
Steps:
low=0, high=6, mid=3, arr[mid]=12

=== Code Execution Successful ===
```

Result: The program has been successfully executed.

9: K Closest Points to Origin

Aim:
To find the k closest points to the origin.

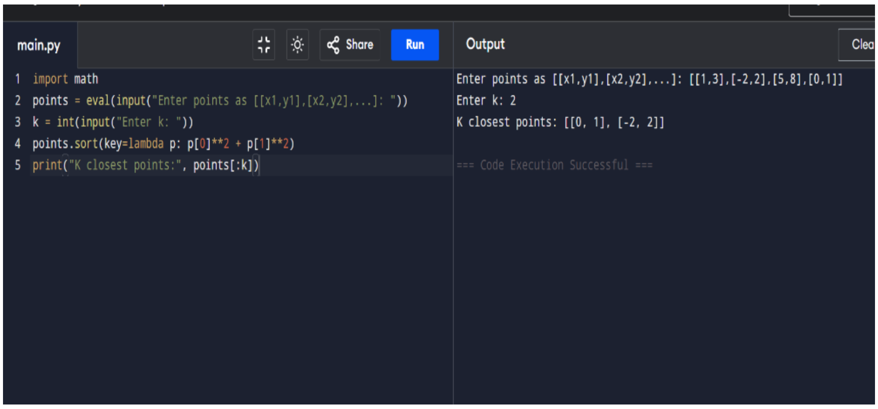
Algorithm:

1. Compute distance = x^2+y^2 for each point.
2. Sort points by distance.
3. Pick first k points.

Code:

```
points = eval(input("Enter points as [[x1,y1],[x2,y2],...]: "))

k = int(input("Enter k: "))

points.sort(key=lambda p: p[0]**2 + p[1]**2)

print("K closest points:", points[:k])
```

Output:



```
import math
points = eval(input("Enter points as [[x1,y1],[x2,y2],...]: "))
k = int(input("Enter k: "))
points.sort(key=lambda p: p[0]**2 + p[1]**2)
print("K closest points:", points[:k])
```

```
Enter points as [[x1,y1],[x2,y2],...]: [[1,3],[-2,2],[5,8],[0,1]]
Enter k: 2
K closest points: [[0, 1], [-2, 2]]

=== Code Execution Successful ===
```

Result: The program has been successfully executed.

10.4-Sum Tuples

Aim:
To count tuples (i,j,k,l) such that A[i]+B[j]+C[k]+D[l]=0.

Algorithm:

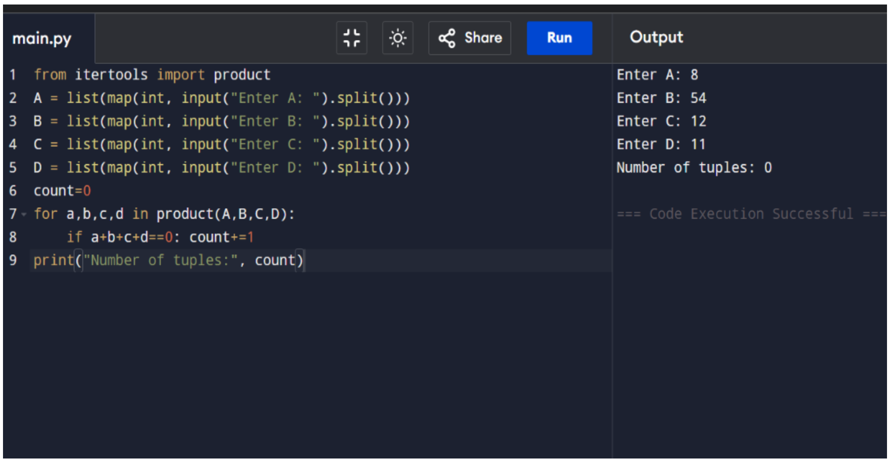
1. Compute all sums of A+B.
2. Compute all sums of C+D.
3. Count matches using hash map.

Code:

```
from collections import Counter
```

```
A = list(map(int, input("Enter A: ").split()))

B = list(map(int, input("Enter B: ").split()))

C = list(map(int, input("Enter C: ").split()))

D = list(map(int, input("Enter D: ").split()))

AB = Counter(a+b for a in A for b in B)

count = 0

for c in C:

  for d in D:

    count += AB.get(-(c+d), 0)

print("Number of tuples:", count)
```

Output:



```
from itertools import product
A = list(map(int, input("Enter A: ").split()))
B = list(map(int, input("Enter B: ").split()))
C = list(map(int, input("Enter C: ").split()))
D = list(map(int, input("Enter D: ").split()))
count=0
for a,b,c,d in product(A,B,C,D):
    if a+b+c+d==0: count+=1
print("Number of tuples:", count)
```

```
Enter A: 8
Enter B: 54
Enter C: 12
Enter D: 11
Number of tuples: 0

=== Code Execution Successful ===
```

Result: The program has been successfully executed.

11: Median of Medians (k-th smallest)

Aim:

To find the k-th smallest element using Median of Medians.

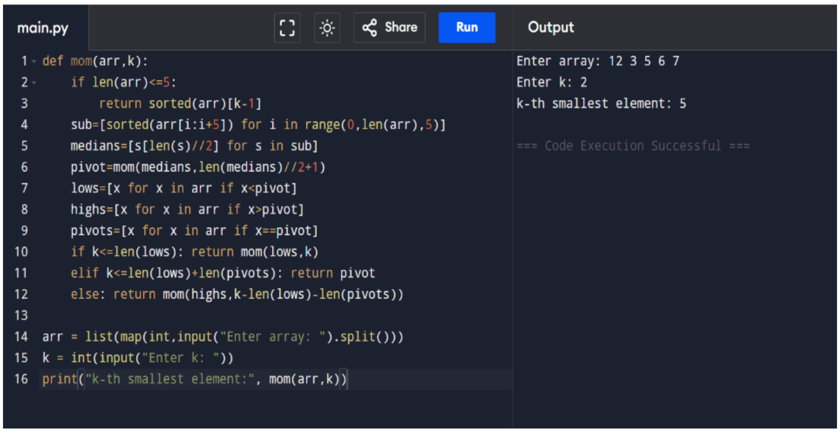
Algorithm:

1. Divide array into groups of 5.
2. Find median of each group.
3. Recursively select median of medians as pivot.
4. Partition and recurse on correct side.

Code:

```
def median_of_medians(arr, k):
   if len(arr) <= 5:
      return sorted(arr)[k-1]
   medians = [sorted(arr[i:i+5])[len(arr[i:i+5])//2] for i in range(0,len(arr),5)]
   pivot = median_of_medians(medians, len(medians)//2+1)
   lows = [x for x in arr if x < pivot]
   highs = [x for x in arr if x > pivot]
   pivots = [x for x in arr if x == pivot]
   if k <= len(lows):
      return median_of_medians(lows, k)
   elif k <= len(lows)+len(pivots):
      return pivot
   else:
      return median_of_medians(highs, k-len(lows)-len(pivots))
arr = list(map(int, input("Enter array: ").split()))
k = int(input("Enter k: "))
print("K-th smallest:", median_of_medians(arr, k))
```

Output:



```
def mom(arr,k):
    if len(arr)<=5:
        return sorted(arr)[k-1]
    sub=[sorted(arr[i:i+5]) for i in range(0,len(arr),5)]
    medians=[s[len(s)//2] for s in sub]
    pivot=mom(medians,len(medians)//2+1)
    lows=[x for x in arr if x<pivot]
    highs=[x for x in arr if x>pivot]
    pivots=[x for x in arr if x==pivot]
    if k<=len(lows): return mom(lows,k)
    elif k<=len(lows)+len(pivots): return pivot
    else: return mom(highs,k-len(lows)-len(pivots))

arr = list(map(int,input("Enter array: ").split()))
k = int(input("Enter k: "))
print("k-th smallest element:", mom(arr,k))
```

```
Enter array: 12 3 5 6 7
Enter k: 2
k-th smallest element: 5

=== Code Execution Successful ===
```

Result: The program has been successfully executed.

12. Median of Medians Function (Reusable)

Aim:

To implement median_of_medians() function and return k-th smallest.

Algorithm:

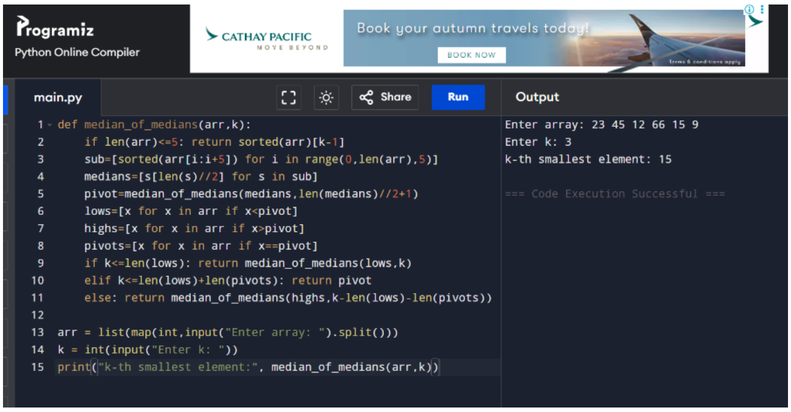
1. Divide array into groups of 5.
2. Find median of each group.
3. Recursively select median of medians as pivot.
4. Partition and recurse on correct side.

Code:

```
def partition(arr, pivot):
    low = [x for x in arr if x < pivot]
    high = [x for x in arr if x > pivot]
    equal = [x for x in arr if x == pivot]
    return low, equal, high
def median_of_medians(arr, k):
    # Base case: small array
    if len(arr) <= 5:
        arr.sort()
        return arr[k]
Split into groups of 5
    groups = [arr[i:i+5] for i in range(0, len(arr), 5)]
    Find median of each group
    medians = [sorted(group)[len(group)//2] for group in groups]
Recursively find pivot
    pivot = median_of_medians(medians, len(medians)//2)
    low, equal, high = partition(arr, pivot)
    # Step 5: Recurse depending on k
    if k < len(low):
        return median_of_medians(low, k)
```

```
  elif k < len(low) + len(equal):
      return pivot
  else:
      return median_of_medians(high, k - len(low) - len(equal))
arr = [12, 3, 5, 7, 4, 19, 26]
k = 3  # Find 3rd smallest (0-based index → 4th element if human count)
print(f"{k+1}th smallest element is:", median_of_medians(arr, k))
```

Output:



```
def median_of_medians(arr,k):
    if len(arr)<=5: return sorted(arr)[k-1]
    sub=[sorted(arr[i:i+5]) for i in range(0,len(arr),5)]
    medians=[s[len(s)//2] for s in sub]
    pivot=median_of_medians(medians,len(medians)//2+1)
    lows=[x for x in arr if x<pivot]
    highs=[x for x in arr if x>pivot]
    pivots=[x for x in arr if x==pivot]
    if k<=len(lows): return median_of_medians(lows,k)
    elif k<=len(lows)+len(pivots): return pivot
    else: return median_of_medians(highs,k-len(lows)-len(pivots))

arr = list(map(int,input("Enter array: ").split()))
k = int(input("Enter k: "))
print("k-th smallest element:", median_of_medians(arr,k))
```

```
Enter array: 23 45 12 66 15 9
Enter k: 3
k-th smallest element: 15

=== Code Execution Successful ===
```

Result: The program has been successfully executed.

13.Meet in the Middle – Closest Subset Sum

Aim:

To find subset sum closest to target using Meet in the Middle.

Algorithm:

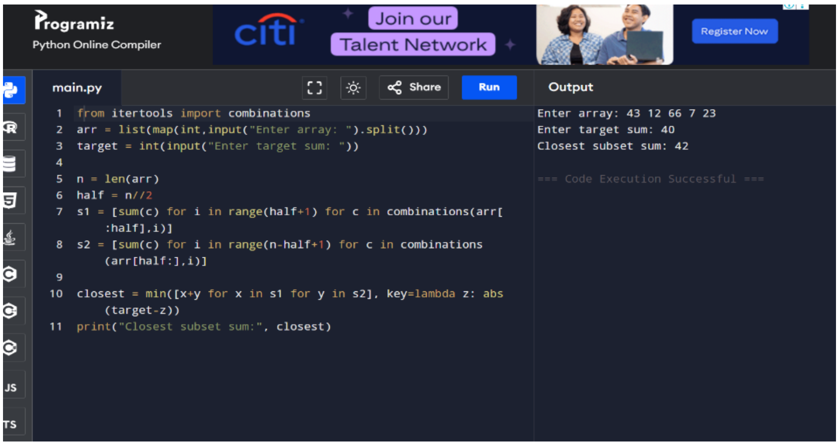
1. **Split array into two halves.**
2. **Generate all subset sums for each half.**
3. **For each sum in left, find closest match in right.**

Code:

```
import itertools, bisect
def meet_in_middle(arr, target):
  n = len(arr)//2
  left, right = arr[:n], arr[n:]
  L = [sum(sub) for r in range(len(left)+1) for sub in itertools.combinations(left,r)]
```

```
    R = [sum(sub) for r in range(len(right)+1) for sub in itertools.combinations(right,r)]
    R.sort()
    best = float('inf')
    for s in L:
        pos = bisect.bisect_left(R, target-s)
        if pos < len(R):
            best = min(best, abs(target-(s+R[pos])))
        if pos > 0:
            best = min(best, abs(target-(s+R[pos-1])))
    return target-best
arr = list(map(int, input("Enter array: ").split()))
target = int(input("Enter target: "))
print("Closest subset sum:", meet_in_middle(arr, target))
```

Output:



```
from itertools import combinations
arr = list(map(int,input("Enter array: ").split()))
target = int(input("Enter target sum: "))

n = len(arr)
half = n//2
s1 = [sum(c) for i in range(half+1) for c in combinations(arr[
    :half],i)]
s2 = [sum(c) for i in range(n-half+1) for c in combinations
    (arr[half:],i)]

closest = min([x+y for x in s1 for y in s2], key=lambda z: abs
    (target-z))
print("Closest subset sum:", closest)
```

```
Enter array: 43 12 66 7 23
Enter target sum: 40
Closest subset sum: 42

=== Code Execution Successful ===
```

Result: The program has been successfully executed.

14: Meet in the Middle – Exact Subset Sum

Aim:

To check if a subset exists with exact sum E.

Algorithm:

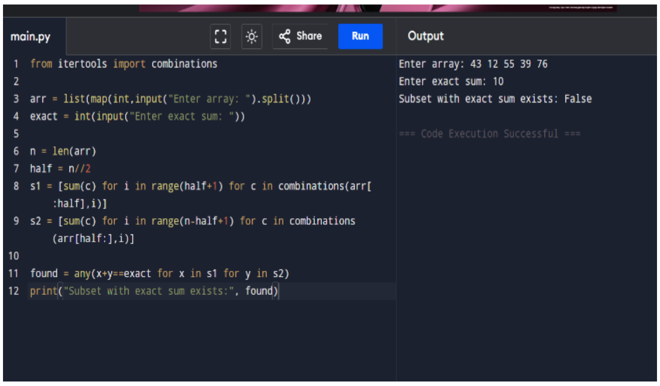
1. Split array in two.
2. Generate all subset sums of both.

3. Check if target - sum_left exists in right.

Code:

```
import itertools

def subset_sum(arr, target):

  n = len(arr)//2

  left, right = arr[:n], arr[n:]

  L = [sum(sub) for r in range(len(left)+1) for sub in itertools.combinations(left,r)]

  R = [sum(sub) for r in range(len(right)+1) for sub in itertools.combinations(right,r)]

  R = set(R)

  for s in L:

    if target-s in R:

      return True

  return False

arr = list(map(int, input("Enter array: ").split()))

target = int(input("Enter target sum: "))

print("Subset exists?", subset_sum(arr, target))
```

Output:



```
from itertools import combinations

arr = list(map(int,input("Enter array: ").split()))
exact = int(input("Enter exact sum: "))

n = len(arr)
half = n//2
s1 = [sum(c) for i in range(half+1) for c in combinations(arr[:half],i)]
s2 = [sum(c) for i in range(n-half+1) for c in combinations(arr[half:],i)]

found = any(x+y==exact for x in s1 for y in s2)
print("Subset with exact sum exists:", found)
```

```
Enter array: 43 12 55 39 76
Enter exact sum: 10
Subset with exact sum exists: False

=== Code Execution Successful ===
```

Result: The program has been successfully executed.

15: Strassen’s Matrix Multiplication (2×2)

Aim:

To multiply two 2×2 matrices using Strassen’s algorithm.

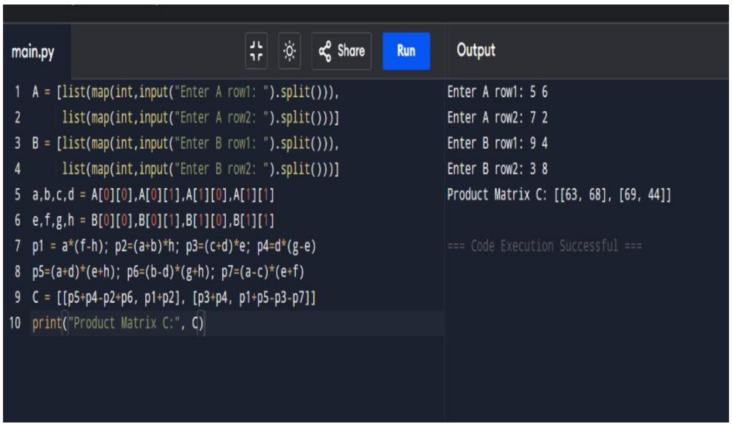
Algorithm:

1. Compute 7 products P1…P7.
2. Combine them into result matrix.

Code:

```
def strassen(A, B):
  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)
  return [[p5+p4-p2+p6, p1+p2],
      [p3+p4, p1+p5-p3-p7]]
A = [[1,7],[3,5]]
B = [[1,3],[7,5]]
print("Result:", strassen(A,B))
```

Output:



```
A = [list(map(int,input("Enter A row1: ").split())),
     list(map(int,input("Enter A row2: ").split()))]
B = [list(map(int,input("Enter B row1: ").split())),
     list(map(int,input("Enter B row2: ").split()))]
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)
C = [[p5+p4-p2+p6, p1+p2], [p3+p4, p1+p5-p3-p7]]
print("Product Matrix C:", C)
```

```
Enter A row1: 5 6
Enter A row2: 7 2
Enter B row1: 9 4
Enter B row2: 3 8
Product Matrix C: [[63, 68], [69, 44]]


=== Code Execution Successful ===
```

Result: The program has been successfully executed.

16: Karatsuba Multiplication

Aim:

To multiply two large integers using Karatsuba algorithm.

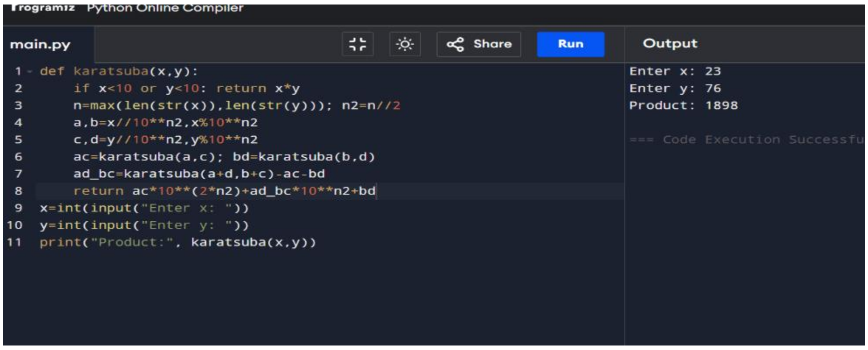
Algorithm:

1. Split numbers into halves.
2. Recursively compute three multiplications.
3. Combine results.

Code:

```
def karatsuba(x, y):
  if x < 10 or y < 10:
    return x*y
  m = max(len(str(x)), len(str(y)))
  m2 = m//2
  high1, low1 = divmod(x, 10**m2)
  high2, low2 = divmod(y, 10**m2)
  z0 = karatsuba(low1, low2)
  z1 = karatsuba((low1+high1), (low2+high2))
  z2 = karatsuba(high1, high2)
  return (z2*10**(2*m2)) + ((z1-z2-z0)*10**m2) + z0
x = int(input("Enter x: "))
y = int(input("Enter y: "))
print("Product:", karatsuba(x,y))
```

Output:



```
def karatsuba(x,y):
    if x<10 or y<10: return x*y
    n=max(len(str(x)),len(str(y))); n2=n//2
    a,b=x//10**n2,x%10**n2
    c,d=y//10**n2,y%10**n2
    ac=karatsuba(a,c); bd=karatsuba(b,d)
    ad_bc=karatsuba(a+d,b+c)-ac-bd
    return ac*10**(2*n2)+ad_bc*10**n2+bd
x=int(input("Enter x: "))
y=int(input("Enter y: "))
print("Product:", karatsuba(x,y))
```

```
Enter x: 23
Enter y: 76
Product: 1898

=== Code Execution Successfu
```

Result: The program has been successfully executed.