

main.py



Run

Output

Clear

```
1 import math
2 def floyd_marshall(n, edges):
3
4     dist = [[math.inf] * n for _ in range(n)]
5     for i in range(n):
6         dist[i][i] = 0
7     for u, v, w in edges:
8         dist[u][v] = w
9
10    print("Distance matrix before Floyd-Warshall:")
11    for row in dist:
12        print(row)
13
14    for k in range(n):
15        for i in range(n):
16            for j in range(n):
17                if dist[i][k] + dist[k][j] < dist[i][j]:
18                    dist[i][j] = dist[i][k] + dist[k][j]
19    print("\nDistance matrix after Floyd-Warshall:")
20    for row in dist:
21        print(row)
22    return dist
23 edges_a = [
24     [0, 1, 3], [0, 2, 8], [0, 3, -4],
25     [1, 3, 1], [1, 2, 4],
26     [2, 0, 2],
27     [3, 2, -5]
28 ]
29 n_a = 4
30 dist_a = floyd_marshall(n_a, edges_a)
31 print("\nShortest path from City 1 to City 3:", dist_a[0][2])
32 edges_b = [
33     [0, 1, 1], [0, 2, 5],
34     [1, 2, 2], [1, 3, 1],
35     [2, 4, 3],
36     [3, 4, 1], [3, 5, 6],
37     [4, 5, 2]
38 ]
39 n_b = 6
40 dist_b = floyd_marshall(n_b, edges_b)
```

Distance matrix before Floyd-Warshall:

```
[0, 3, 8, -4]
[inf, 0, 4, 1]
[2, inf, 0, inf]
[inf, inf, -5, 0]
```

Distance matrix after Floyd-Warshall:

```
[-7, -4, -9, -11]
[-2, 0, -4, -6]
[-5, -2, -7, -9]
[-10, -7, -12, -14]
```

Shortest path from City 1 to City 3: -9

Distance matrix before Floyd-Warshall:

```
[0, 1, 5, inf, inf, inf]
[inf, 0, 2, 1, inf, inf]
[inf, inf, 0, inf, 3, inf]
[inf, inf, inf, 0, 1, 6]
[inf, inf, inf, inf, 0, 2]
[inf, inf, inf, inf, inf, 0]
```

Distance matrix after Floyd-Warshall:

```
[0, 1, 3, 2, 3, 5]
[inf, 0, 2, 1, 2, 4]
[inf, inf, 0, inf, 3, 5]
[inf, inf, inf, 0, 1, 3]
[inf, inf, inf, inf, 0, 2]
[inf, inf, inf, inf, inf, 0]
```

==== Code Execution Successful ===



```
1 INF = float('inf')
2 n = 5
3 edges = [[0,1,2],[0,4,8],[1,2,3],[1,4,2],[2,3,1],[3,4,1]]
4 dist = [[INF]*n for _ in range(n)]
5 for i in range(n):
6     dist[i][i] = 0
7 for u,v,w in edges:
8     dist[u][v] = w
9     dist[v][u] = w
10 for k in range(n):
11     for i in range(n):
12         for j in range(n):
13             if dist[i][k] + dist[k][j] < dist[i][j]:
14                 dist[i][j] = dist[i][k] + dist[k][j]
15 print("Shortest Distance Matrix:")
16 for row in dist:
17     print(["INF" if x==INF else x for x in row])
18 print(f"Shortest path from C to A = {dist[2][0]}")
19 print(f"Shortest path from E to C = {dist[4][2]}")
```

Shortest Distance Matrix:

```
[0, 2, 5, 5, 4]
[2, 0, 3, 3, 2]
[5, 3, 0, 1, 2]
[5, 3, 1, 0, 1]
[4, 2, 2, 1, 0]
```

Shortest path from C to A = 5

Shortest path from E to C = 2

== Code Execution Successful ==



main.py



Run

Output

Clear

```
1 def assembly_line(a1, a2, t1, t2, e1, e2, x1, x2, n):
2     f1 = [0] * n
3     f2 = [0] * n
4
5     f1[0] = e1 + a1[0]
6     f2[0] = e2 + a2[0]
7
8     for i in range(1, n):
9         f1[i] = min(f1[i-1] + a1[i], f2[i-1] + t2[i-1] + a1[i])
10        f2[i] = min(f2[i-1] + a2[i], f1[i-1] + t1[i-1] + a2[i])
11
12    return min(f1[n-1] + x1, f2[n-1] + x2)
13
14 a1 = [4, 5, 3, 2]
15 a2 = [2, 10, 1, 4]
16 t1 = [7, 4, 5]
17 t2 = [9, 2, 8]
18 e1 = 10
19 e2 = 12
20 x1 = 18
21 x2 = 7
22 n = 4
23
24 print("Minimum time to process the product:", assembly_line(a1, a2, t1
, t2, e1, e2, x1, x2, n))
```

Minimum time to process the product: 35
== Code Execution Successful ==



main.py



Output

Clear

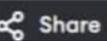
```
1 import heapq
2 class Node:
3     def __init__(self, char, freq):
4         self.char = char
5         self.freq = freq
6         self.left = None
7         self.right = None
8
9     def __lt__(self, other):
10        return self.freq < other.freq
11
12 def huffman_codes(characters, frequencies):
13     heap = [Node(characters[i], frequencies[i]) for i in range(len(characters))]
14     heapq.heapify(heap)
15
16     while len(heap) > 1:
17         left = heapq.heappop(heap)
18         right = heapq.heappop(heap)
19         new_node = Node(None, left.freq + right.freq)
20         new_node.left = left
21         new_node.right = right
22         heapq.heappush(heap, new_node)
23
24     root = heap[0]
25     codes = {}
26
27     def generate_codes(node, code):
28         if node:
29             if node.char is not None:
30                 codes[node.char] = code
31             generate_codes(node.left, code + "0")
32             generate_codes(node.right, code + "1")
33
34     generate_codes(root, "")
35     return sorted(codes.items())
36
37 characters = ['a', 'b', 'c', 'd']
38 frequencies = [5, 9, 12, 13]
39 print(huffman_codes(characters, frequencies))
```

[('a', '00'), ('b', '01'), ('c', '10'), ('d', '11')]

== Code Execution Successful ==



main.py



Run

Output

Clear

```
1 def numIdenticalPairs(nums):
2     count = 0
3     n = len(nums)
4     for i in range(n):
5         for j in range(i+1, n):
6             if nums[i] == nums[j]:
7                 count += 1
8     return count
9 print(numIdenticalPairs([1,2,3,1,1,3]))
10 print(numIdenticalPairs([1,1,1,1]))
```

4

6

--- Code Execution Successful ---

```
1 from itertools import combinations
2 import bisect
3
4 def subset_sums(arr):
5     """Generate all possible subset sums of an array"""
6     sums = []
7     n = len(arr)
8     for r in range(n+1):
9         for comb in combinations(arr, r):
10            sums.append(sum(comb))
11    return sums
12
13 def closest_subset_sum(arr, target):
14     n = len(arr)
15     first_half = arr[:n//2]
16     second_half = arr[n//2:]
17
18     sums1 = subset_sums(first_half)
19     sums2 = sorted(subset_sums(second_half))
20
21     closest_sum = None
22     min_diff = float('inf')
23
24     for s1 in sums1:
25         idx = bisect.bisect_left(sums2, target - s1)
26         for j in [idx, idx-1]:
27             if 0 <= j < len(sums2):
28                 total = s1 + sums2[j]
29                 diff = abs(target - total)
30                 if diff < min_diff:
31                     min_diff = diff
32                     closest_sum = total
33
34     return closest_sum
35
36 set_a = [45, 34, 4, 32, 5, 2]
37 target_a = 42
38 print(closest_subset_sum(set_a, target_a))
39
40 set_b = [1, 3, 2, 7, 4, 6]
41 target_b = 10
42 print(closest_subset_sum(set_b, target_b))
```

41

10

--- Code Execution Successful ---



main.py



Run

Output

Clear

```
1 def greedy_load(weights, max_capacity):
2     weights.sort(reverse=True)
3     total = 0
4     for w in weights:
5         if total + w <= max_capacity:
6             total += w
7     return total
8
9 weights = [10, 20, 30, 40, 50]
10 max_capacity = 50
11 print(greedy_load(weights, max_capacity))
```

50

== Code Execution Successful ==



main.py



Run

Clear

```
1 INF = float('inf')
2 dist = [
3     [0, 2, INF, 1, INF, INF], # A
4     [2, 0, 3, INF, INF, INF], # B
5     [INF, 3, 0, 1, 2, INF], # C
6     [1, INF, 1, 0, INF, 4], # D
7     [INF, INF, 2, INF, 0, 1], # E
8     [INF, INF, INF, 4, 1, 0] # F
9 ]
10
11 V = len(dist)
12 def floyd_marshall(distance):
13     shortest = [row[:] for row in distance]
14     for k in range(V):
15         for i in range(V):
16             for j in range(V):
17                 if shortest[i][k] + shortest[k][j] < shortest[i][j]:
18                     shortest[i][j] = shortest[i][k] + shortest[k][j]
19     return shortest
20 def display_path(shortest):
21     print(f"Router A to Router F = {shortest[0][5]}")
22 shortest_before = floyd_marshall(dist)
23 print("Before link failure:")
24 display_path(shortest_before)
25 dist[1][3] = INF
26 dist[3][1] = INF
27
28 shortest_after = floyd_marshall(dist)
29 print("After link failure:")
30 display_path(shortest_after)
```

Output

Before link failure:

Router A to Router F = 5

After link failure:

Router A to Router F = 5

--- Code Execution Successful ---



main.py



Run

Output

Clear

```
1 def print_board(positions):
2     n = len(positions)
3     for r in range(n):
4         row = ['.' * n
5         row[positions[r]] = 'Q'
6         print(''.join(row))
7     print()
8
9 print_board([1,3,0,2])
```

```
. Q . .
. . . Q
Q . . .
. . Q .
```

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



JS

TS



-GO

php





main.py



Share

Run

```
1 - def maxCoins(piles):
2     piles.sort()
3     n = len(piles)
4     res = 0
5     left, right = 0, n - 1
6
7     while left < right:
8         res += piles[right - 1]
9         right -= 2
10        left += 1
11
12    return res
13
14 piles = [2, 4, 1, 2, 7, 8]
15 print("Maximum coins you can have:", maxCoins(piles))
16 piles = [2, 4, 5]
17 print("Maximum coins you can have:", maxCoins(piles))
```

Output

```
Maximum coins you can have: 9
Maximum coins you can have: 4
```

```
*** Code Execution Successful ***
```

Clear



main.py



Run

Output

Clear

```

1- def optimal_bst(keys, freq):
2     n = len(keys)
3     cost = [[0]*(n+1) for _ in range(n+1)]
4     root = [[0]*(n+1) for _ in range(n+1)]
5
6     def sum_freq(i, j):
7         return sum(freq[i:j+1])
8     for i in range(n):
9         cost[i][i] = freq[i]
10        root[i][i] = i + 1
11    for L in range(2, n+1):
12        for i in range(n-L+1):
13            j = i + L - 1
14            cost[i][j] = float('inf')
15            for r in range(i, j+1):
16                c = (0 if r==i else cost[i][r-1]) + \
17                    (0 if r==j else cost[r+1][j]) + \
18                    sum_freq(i, j)
19                if c < cost[i][j]:
20                    cost[i][j] = c
21                    root[i][j] = r + 1
22
23    return cost, root
24 keys = ['A', 'B', 'C', 'D']
25 freq = [0.1, 0.2, 0.4, 0.3]
26
27 cost_table, root_table = optimal_bst(keys, freq)
28 print("Cost Table:")
29 for row in cost_table[:len(keys)]:
30     print(["{:1f}".format(x) for x in row[:len(keys)+1]])
31     print("\nRoot Table:")
32 for row in root_table[:len(keys)]:
33     print("\nMinimum cost of OBST =", cost_table[0][len(keys)-1])

```

Cost Table:
['0.1', '0.4', '1.1', '1.7', '0.0']

Root Table:
['0.0', '0.2', '0.8', '1.4', '0.0']

Root Table:
['0.0', '0.0', '0.4', '1.0', '0.0']

Root Table:
['0.0', '0.0', '0.0', '0.3', '0.0']

Root Table:
Minimum cost of OBST = 1.7

==== Code Execution Successful ===

```
1 from functools import lru_cache
2 def cat_and_mouse(graph):
3     N = len(graph)
4     MOUSE, CAT, DRAW = 1, 2, 0
5     @lru_cache(None)
6     def dfs(mouse, cat, turn):
7         if turn > 2 * N:
8             return DRAW
9         if mouse == 0:
10            return MOUSE
11        if mouse == cat:
12            return CAT
13        if turn % 2 == 0:
14            result = CAT
15            for nxt in graph[mouse]:
16                res = dfs(nxt, cat, turn+1)
17                if res == MOUSE:
18                    return MOUSE
19                if res == DRAW:
20                    result = DRAW
21            return result
22        else:
23            result = MOUSE
24            for nxt in graph[cat]:
25                if nxt == 0:
26                    continue
27                res = dfs(mouse, nxt, turn+1)
28                if res == CAT:
29                    return CAT
30                if res == DRAW:
31                    result = DRAW
32            return result
33    return dfs(1, 2, 0)
34 graph1 = [[2,5],[3],[0,4,5],[1,4,5],[2,3],[0,2,3]]
35 print("Example 1 Output =", cat_and_mouse(graph1))
36 graph2 = [[1,3],[0],[3],[0,2]]
37 print("Example 2 Output =", cat_and_mouse(graph2))
```

Example 1 Output = 0
Example 2 Output = 1

--- Code Execution Successful ---



main.py



Run

Output

Clear

```
1 def minimumTimeRequired(jobs, k):
2     jobs.sort(reverse=True)
3     workloads = [0] * k
4     ans = sum(jobs)
5
6     def backtrack(i):
7         nonlocal ans
8         if i == len(jobs):
9             ans = min(ans, max(workloads))
10            return
11         seen = set()
12         for w in range(k):
13             if workloads[w] in seen:
14                 continue
15             seen.add(workloads[w])
16             workloads[w] += jobs[i]
17             if max(workloads) < ans:
18                 backtrack(i + 1)
19             workloads[w] -= jobs[i]
20             if workloads[w] == 0:
21                 break
22
23     backtrack(0)
24     return ans
25
26 print(minimumTimeRequired([3, 2, 3], 3))
27 print(minimumTimeRequired([1, 2, 4, 7, 8], 2))
```

3

11

== Code Execution Successful ==

```
1 import heapq
2 class Node:
3     def __init__(self, char, freq):
4         self.char = char
5         self.freq = freq
6         self.left = None
7         self.right = None
8
9     def __lt__(self, other):
10        return self.freq < other.freq
11 def build_huffman_tree(characters, frequencies):
12     heap = [Node(characters[i], frequencies[i]) for i in range(len(characters))]
13     heapq.heapify(heap)
14
15     while len(heap) > 1:
16         left = heapq.heappop(heap)
17         right = heapq.heappop(heap)
18         new_node = Node(None, left.freq + right.freq)
19         new_node.left = left
20         new_node.right = right
21         heapq.heappush(heap, new_node)
22
23     return heap[0]
24 def decode_huffman(root, encoded_string):
25     decoded = ""
26     current = root
27     for bit in encoded_string:
28         if bit == '0':
29             current = current.left
30         else:
31             current = current.right
32         if current.left is None and current.right is None:
33             decoded += current.char
34             current = root
35     return decoded
36
37 characters = ['a', 'b', 'c', 'd']
38 frequencies = [5, 9, 12, 18]
39 encoded_string = '1101100111110'
40
41 root = build_huffman_tree(characters, frequencies)
42 print(decode_huffman(root, encoded_string))
```

dbcbdd

*** Code Execution Successful ***



main.py



Run

Output

Clear

```
1 import heapq
2 def dijkstra(n, edges, source, target):
3     graph = {i: [] for i in range(n)}
4     for u, v, w in edges:
5         graph[u].append((v, w))
6         graph[v].append((u, w))
7
8     dist = [float('inf')] * n
9     dist[source] = 0
10    pq = [(0, source)]
11
12    while pq:
13        d, node = heapq.heappop(pq)
14        if node == target:
15            return d
16        if d > dist[node]:
17            continue
18        for nei, w in graph[node]:
19            new_dist = d + w
20            if new_dist < dist[nei]:
21                dist[nei] = new_dist
22                heapq.heappush(pq, (new_dist, nei))
23
24    return -1
25
26 edges1 = [(0,1,7),(0,2,9),(0,5,14),(1,2,10),(1,3,15),(2,3,11),(2,5,2),(3,4,6),(4,5,9)]
27 print(dijkstra(6, edges1, 0, 4))
28 edges2 = [(0,1,10),(0,4,3),(1,2,2),(1,4,4),(2,3,9),(3,2,7),(4,1,1),(4,2,8),(4,3,2)]
29 print(dijkstra(5, edges2, 0, 3))
```

20

5

--- Code Execution Successful ---



```
1 - def optimal_bst(keys, freq):
2     n = len(keys)
3     cost = [[0]*n for _ in range(n)]
4     root = [[0]*n for _ in range(n)]
5     def sum_freq(i, j):
6         return sum(freq[i:j+1])
7     for i in range(n):
8         cost[i][i] = freq[i]
9         root[i][i] = i
10    for L in range(2, n+1):
11        for i in range(n-L+1):
12            j = i + L - 1
13            cost[i][j] = float('inf')
14            for r in range(i, j+1):
15                c = (0 if r==i else cost[i][r-1]) + \
16                    (0 if r==j else cost[r+1][j]) + \
17                    sum_freq(i, j)
18                if c < cost[i][j]:
19                    cost[i][j] = c
20                    root[i][j] = r
21
22    return cost, root
23 keys = [10, 12, 16, 21]
24 freq = [4, 2, 6, 3]
25
26 cost_table, root_table = optimal_bst(keys, freq)
27 print("Cost Table:")
28 for row in cost_table:
29     print(row)
30 print("\nRoot Table:")
31 for row in root_table:
32     print(row)
33 print("\nMinimum cost of OBST =", cost_table[0][len(keys)-1])
```

Cost Table:

[4, 8, 20, 26]
[0, 2, 10, 16]
[0, 0, 6, 12]
[0, 0, 0, 3]

Root Table:

[0, 0, 2, 2]
[0, 1, 2, 2]
[0, 0, 2, 2]
[0, 0, 0, 3]

Minimum cost of OBST = 26

--- Code Execution Successful ---



```
1 import heapq
2 from collections import defaultdict
3
4 def maxProbability(n, edges, succProb, start, end):
5     graph = defaultdict(list)
6     for (u, v), p in zip(edges, succProb):
7         graph[u].append((v, p))
8         graph[v].append((u, p))
9     heap = [(-1.0, start)]
10    prob = [0.0] * n
11    prob[start] = 1.0
12
13    while heap:
14        curr_prob, node = heapq.heappop(heap)
15        curr_prob *= -1
16        if node == end:
17            return curr_prob
18        if curr_prob < prob[node]:
19            continue
20        for nei, p in graph[node]:
21            new_prob = curr_prob * p
22            if new_prob > prob[nei]:
23                prob[nei] = new_prob
24                heapq.heappush(heap, (-new_prob, nei))
25
26    return 0.0
27 n1 = 3
28 edges1 = [[0,1],[1,2],[0,2]]
29 succProb1 = [0.5,0.5,0.2]
30 start1, end1 = 0, 2
31 print("Example 1 Output -", maxProbability(n1, edges1, succProb1, start1, end1))
32 n2 = 3
33 edges2 = [[0,1],[1,2],[0,2]]
34 succProb2 = [0.5,0.5,0.3]
35 start2, end2 = 0, 2
36 print("Example 2 Output -", maxProbability(n2, edges2, succProb2, start2, end2))
```

Example 1 Output = 0.25
Example 2 Output = 0.3

*** Code Execution Successful ***

```
1 - class DisjointSet:
2 -     def __init__(self, n):
3 -         self.parent = list(range(n))
4 -         self.rank = [0] * n
5 -     def find(self, u):
6 -         if self.parent[u] != u:
7 -             self.parent[u] = self.find(self.parent[u])
8 -         return self.parent[u]
9 -
10 -    def union(self, u, v):
11 -        u_root = self.find(u)
12 -        v_root = self.find(v)
13 -        if u_root == v_root:
14 -            return False
15 -        if self.rank[u_root] < self.rank[v_root]:
16 -            self.parent[u_root] = v_root
17 -        elif self.rank[u_root] > self.rank[v_root]:
18 -            self.parent[v_root] = u_root
19 -        else:
20 -            self.parent[v_root] = u_root
21 -            self.rank[u_root] += 1
22 -        return True
23 -    def kruskal_all_mssts(n, edges):
24 -        edges.sort(key=lambda x: x[2])
25 -        ds = DisjointSet(n)
26 -        mst = []
27 -        for u, v, w in edges:
28 -            if ds.union(u, v):
29 -                mst.append((u, v, w))
30 -        return mst
31 -
32 -    def is_unique_mst(n, edges, given_mst):
33 -        given_mst_sorted = sorted(given_mst, key=lambda x: (x[2], x[0], x[1]))
34 -        mst = kruskal_all_mssts(n, edges)
35 -        mst_sorted = sorted(mst, key=lambda x: (x[2], x[0], x[1]))
36 -        return given_mst_sorted == mst_sorted
37 -n = 4
38 -edges = [(0, 1, 10), (0, 2, 6), (0, 3, 5), (1, 3, 15), (2, 3, 4)]
39 -given_mst = [(2, 3, 4), (0, 3, 5), (0, 1, 10)]
40 -
41 -print("Is the given MST unique?", is_unique_mst(n, edges, given_mst))
```

Is the given MST unique? True

--- Code Execution Successful ---



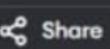
```
1 def is_safe(board, row, col, obstacles, n, m):
2     if (row, col) in obstacles:
3         return False
4
5     for r in range(row):
6         c = board[r]
7         if c == col or abs(c - col) == abs(r - row):
8             return False
9     return True
10
11
12 def solve_nqueens(row, n, m, board, obstacles, restricted_cols, solutions):
13     if row == n:
14         solutions.append(board[:])
15         return
16
17     cols = range(m)
18     if row == 0 and restricted_cols:
19         cols = restricted_cols
20
21     for col in cols:
22         if is_safe(board, row, col, obstacles, n, m):
23             board[row] = col
24             solve_nqueens(row + 1, n, m, board, obstacles, restricted_cols, solutions)
25             board[row] = -1
26
27
28 def generalized_nqueens(n, m=None, obstacles=None, restricted_cols=None):
29     if m is None:
30         m = n
31     if obstacles is None:
32         obstacles = set()
33     if restricted_cols is None:
34         restricted_cols = []
35
36     board = [-1] * n
37     solutions = []
38     solve_nqueens(0, n, m, board, obstacles, restricted_cols, solutions)
39     return solutions
40
41 sol_8x10 = generalized_nqueens(8, 10)
42 print("8x10 Board Solution:", sol_8x10[0] if sol_8x10 else "No solution")
43
44 obstacles_5x5 = {(2, 2), (4, 4)}
45 sol_5x5 = generalized_nqueens(5, 5, obstacles=obstacles_5x5)
46 print("5x5 Board with Obstacles Solution:", sol_5x5[0] if sol_5x5 else "No solution")
47
48 restricted_cols_6x6 = [2, 4]
49 sol_6x6 = generalized_nqueens(6, 6, restricted_cols=restricted_cols_6x6)
```

8x10 Board Solution: [0, 2, 4, 1, 7, 9, 3, 6]

5x5 Board with Obstacles Solution: [0, 2, 4, 1, 3]

6x6 Board with Restricted Columns Solution: [2, 5, 1, 4, 0, 3]

--- Code Execution Successful ---



```
1 def findKthPositive(arr, k):
2     n = len(arr)
3
4     for i in range(n):
5         missing = arr[i] - (i + 1)
6         if missing >= k:
7             return k + i
8
9
10 print(findKthPositive([2, 3, 4, 7, 11], 5))
11 print(findKthPositive([1, 2, 3, 4], 2))
```

9
6
==== Code Execution Successful ===





main.py



Run

Output

Clear

```
1 import heapq
2 from collections import defaultdict
3 def networkDelayTime(times, n, k):
4     graph = defaultdict(list)
5     for u, v, w in times:
6         graph[u].append((v, w))
7
8     heap = [(0, k)]
9     dist = dict()
10
11    while heap:
12        d, node = heapq.heappop(heap)
13        if node in dist:
14            continue
15        dist[node] = d
16        for nei, w in graph[node]:
17            if nei not in dist:
18                heapq.heappush(heap, (d + w, nei))
19    if len(dist) != n:
20        return -1
21    return max(dist.values())
22 print(networkDelayTime([[2,1,1],[2,3,1],[3,4,1]]), 4, 2))
23 print(networkDelayTime([[1,2,1]]), 2, 1))
24 print(networkDelayTime([[1,2,1]]), 2, 2))
```

2
1
-1

==== Code Execution Successful ===



main.py



Run

Output

Clear

```
1 import itertools
2
3 cities = ['A', 'B', 'C', 'D', 'E']
4
5 dist = {
6     ('A', 'B'):10, ('A', 'C'):15, ('A', 'D'):20, ('A', 'E'):25,
7     ('B', 'C'):35, ('B', 'D'):25, ('B', 'E'):30,
8     ('C', 'D'):30, ('C', 'E'):20,
9     ('D', 'E'):15
10 }
11
12 def get_distance(a, b):
13     if (a, b) in dist:
14         return dist[(a, b)]
15     else:
16         return dist[(b, a)]
17
18 min_path = None
19 min_cost = float('inf')
20
21 for perm in itertools.permutations(cities[1:]): # Fix A as start
22     route = ['A'] + list(perm) + ['A']
23     cost = sum(get_distance(route[i], route[i+1]) for i in range(len
24         (route)-1))
25     if cost < min_cost:
26         min_cost = cost
27         min_path = route
28
29 print("Shortest Route:", " -> ".join(min_path))
30 print("Total Distance:", min_cost)
```

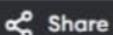
Shortest Route: A -> B -> D -> E -> C -> A

Total Distance: 85

--- Code Execution Successful ---



main.py



Run

Output

Clear

```
1 - def findTheCity(n, edges, distanceThreshold):
2     INF = float('inf')
3     dist = [[INF]*n for _ in range(n)]
4     for i in range(n):
5         dist[i][i] = 0
6     for u, v, w in edges:
7         dist[u][v] = w
8         dist[v][u] = w
9     for k in range(n):
10        for i in range(n):
11            for j in range(n):
12                if dist[i][k] + dist[k][j] < dist[i][j]:
13                    dist[i][j] = dist[i][k] + dist[k][j]
14 result_city = 0
15 min_count = n
16 for i in range(n):
17     count = sum(1 for j in range(n) if i != j and
18                 dist[i][j] <= distanceThreshold)
19     if count <= min_count:
20         min_count = count
21         result_city = i
22 return result_city
23 print(findTheCity(4, [[0,1,3],[1,2,1],[1,3,4],[2,3,1]]), 4))
24 print(findTheCity(5, [[0,1,2],[0,4,8],[1,2,3],[1,4,2],[2,3,1]
25 , [3,4,1]]), 2))
```

3
0

== Code Execution Successful ==



main.py



Run

Output

Clear

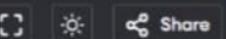
```
1 def selection_sort(arr):
2     n = len(arr)
3     for i in range(n):
4         min_idx = i
5         for j in range(i+1, n):
6             if arr[j] < arr[min_idx]:
7                 min_idx = j
8
9         arr[i], arr[min_idx] = arr[min_idx], arr[i]
10    return arr
11
12 print(selection_sort([5, 2, 9, 1, 5, 6]))
13 print(selection_sort([10, 8, 6, 4, 2]))
14 print(selection_sort([1, 2, 3, 4, 5]))
```

```
[1, 2, 5, 5, 6, 9]
[2, 4, 6, 8, 10]
[1, 2, 3, 4, 5]
```

==== Code Execution Successful ===



main.py



Run

Output

Clear

```
1 from itertools import combinations
2 import bisect
3
4 def subset_sums(arr):
5     """Generate all possible subset sums of an array"""
6     sums = []
7     n = len(arr)
8     for r in range(n+1):
9         for comb in combinations(arr, r):
10            sums.append(sum(comb))
11    return sums
12
13 def subset_sum_exists(arr, target):
14     n = len(arr)
15     first_half = arr[:n//2]
16     second_half = arr[n//2:]
17
18     sums1 = subset_sums(first_half)
19     sums2 = sorted(subset_sums(second_half))
20
21     for s1 in sums1:
22         remaining = target - s1
23         idx = bisect.bisect_left(sums2, remaining)
24         if idx < len(sums2) and sums2[idx] == remaining:
25             return True
26     return False
27
28 arr_a = [1, 3, 9, 2, 7, 12]
29 target_a = 15
30 print(subset_sum_exists(arr_a, target_a))
31
32 arr_b = [3, 34, 4, 12, 5, 2]
33 target_b = 15
34 print(subset_sum_exists(arr_b, target_b))
```

True

True

--- Code Execution Successful ---



main.py



Run

Output

Clear

```
1 import math
2 def karatsuba(x, y):
3     if x < 10 or y < 10:
4         return x * y
5
6     n = max(len(str(x)), len(str(y)))
7     m = math.ceil(n / 2) # Use ceiling to handle odd digits
8
9     high_x, low_x = divmod(x, 10**m)
10    high_y, low_y = divmod(y, 10**m)
11
12    z0 = karatsuba(low_x, low_y)
13    z1 = karatsuba((low_x + high_x), (low_y + high_y))
14    z2 = karatsuba(high_x, high_y)
15
16    return z2 * 10**(2*m) + (z1 - z2 - z0) * 10**m + z0
17
18 x = 1234
19 y = 5678
20 z = karatsuba(x, y)
21 print(f"{x} × {y} = {z}")
```

1234 × 5678 = 7006652

==== Code Execution Successful ===

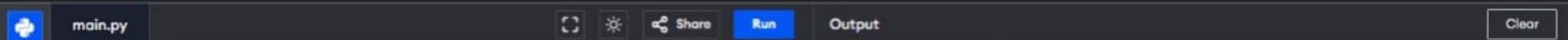
```
1 def binary_search(arr, key):
2     low = 0
3     high = len(arr) - 1
4     step = 1 # Step counter
5
6     while low <= high:
7         mid = (low + high) // 2
8         print(f"Step {step}: Low={low+1}, High={high+1}, Mid={mid+1}, arr[Mid]={arr[mid]}")
9         if arr[mid] == key:
10             print(f"Element {key} found at position {mid+1} (1-based index)\n")
11             return mid
12         elif arr[mid] < key:
13             low = mid + 1
14         else:
15             high = mid - 1
16         step += 1
17
18     print(f"Element {key} not found.\n")
19     return -1
20
21 # Test Case 1
22 a = [3, 9, 14, 19, 25, 31, 42, 47, 53]
23 key = 31
24 print("Test Case 1:")
25 binary_search(a, key)
26
27 # Test Case 2
28 b = [13, 19, 24, 29, 35, 41, 42]
29 key = 42
30 print("Test Case 2:")
31 binary_search(b, key)
32
33 # Test Case 3
34 c = [20, 40, 60, 80, 100, 120]
35 key = 60
36 print("Test Case 3:")
37 binary_search(c, key)
```

Test Case 1:
Step 1: Low=1, High=9, Mid=5, arr[Mid]=25
Step 2: Low=6, High=9, Mid=7, arr[Mid]=42
Step 3: Low=6, High=6, Mid=6, arr[Mid]=31
Element 31 found at position 6 (1-based index)

Test Case 2:
Step 1: Low=1, High=7, Mid=4, arr[Mid]=29
Step 2: Low=5, High=7, Mid=6, arr[Mid]=41
Step 3: Low=7, High=7, Mid=7, arr[Mid]=42
Element 42 found at position 7 (1-based index)

Test Case 3:
Step 1: Low=1, High=6, Mid=3, arr[Mid]=60
Element 60 found at position 3 (1-based index)

--- Code Execution Successful ---



main.py



Output

Clear

```
1 def merge_sort(arr):
2     if len(arr) > 1:
3         mid = len(arr)//2
4         L = arr[:mid]
5         R = arr[mid:]
6
7         merge_sort(L)
8         merge_sort(R)
9
10        i = j = k = 0
11        while i < len(L) and j < len(R):
12            if L[i] < R[j]:
13                arr[k] = L[i]
14                i += 1
15            else:
16                arr[k] = R[j]
17                j += 1
18            k += 1
19
20        while i < len(L):
21            arr[k] = L[i]
22            i += 1
23            k += 1
24
25        while j < len(R):
26            arr[k] = R[j]
27            j += 1
28            k += 1
29
30 a = [12, 4, 78, 23, 45, 67, 89, 1]
31 merge_sort(a)
32 print("Sorted Array:", a)
33
34 b = [38, 27, 43, 3, 9, 82, 10]
35 merge_sort(b)
36 print("Sorted Array:", b)
```

Sorted Array: [11, 15, 21, 23, 27, 28, 31, 35]
Sorted Array: [13, 17, 22, 25, 34, 36, 43, 52, 65, 67]
--- Code Execution Successful ---



main.py



Run

Output

Clear

```
1 def length_of_longest_substring(s: str) -> int:
2     char_index = {}
3     left = 0
4     max_length = 0
5
6     for right, char in enumerate(s):
7         if char in char_index and char_index[char] >= left:
8             left = char_index[char] + 1
9             char_index[char] = right
10            max_length = max(max_length, right - left + 1)
11
12    return max_length
13
14 print(length_of_longest_substring("abcabcbb"))
15 print(length_of_longest_substring("bbbbbb"))
16 print(length_of_longest_substring("pwwkew"))
```

```
3
1
3
```

== Code Execution Successful ==



JS

TS





main.py



Share

Run

Output

Clear

```
1 def find_min_max(arr):
2     print("Min =", min(arr), ", Max =", max(arr))
3
4 a = [2, 4, 6, 8, 10, 12, 14, 18]
5 find_min_max(a)
6
7 a = [11, 13, 15, 17, 19, 21, 23, 35, 37]
8 find_min_max(a)
9
10 a = [22, 34, 35, 36, 43, 67, 12, 13, 15, 17]
11 find_min_max(a)
```

```
Min = 2 , Max = 18
Min = 11 , Max = 37
Min = 12 , Max = 67
==== Code Execution Successful ===
```

Program

Premi

Cours

Progr

Learn



main.py

Run

Clear

```
1 def assembly_line(a1, a2, t1, t2, e1, e2, x1, x2, n):
2     f1 = [0] * n
3     f2 = [0] * n
4
5     f1[0] = e1 + a1[0]
6     f2[0] = e2 + a2[0]
7
8     for i in range(1, n):
9         f1[i] = min(f1[i-1] + a1[i], f2[i-1] + t2[i-1] + a1[i])
10        f2[i] = min(f2[i-1] + a2[i], f1[i-1] + t1[i-1] + a2[i])
11
12    return min(f1[n-1] + x1, f2[n-1] + x2)
13
14
15 # Example Test Case
16 a1 = [4, 5, 3, 2]
17 a2 = [2, 10, 1, 4]
18 t1 = [7, 4, 5]
19 t2 = [9, 2, 8]
20 e1 = 10
21 e2 = 12
22 x1 = 18
23 x2 = 7
24 n = 4
```

Output

Minimum time to process the product: 35

--- Code Execution Successful ---

main.py



Run

Output

Clear

```
1 def merge_sort(arr):
2     if len(arr) > 1:
3         mid = len(arr)//2
4         L = arr[:mid]
5         R = arr[mid:]
6
7         merge_sort(L)
8         merge_sort(R)
9
10        i = j = k = 0
11        # Merge two halves
12        while i < len(L) and j < len(R):
13            if L[i] < R[j]:
14                arr[k] = L[i]
15                i += 1
16            else:
17                arr[k] = R[j]
18                j += 1
19            k += 1
20
21        while i < len(L):
22            arr[k] = L[i]
23            i += 1
24            k += 1
25
26        while j < len(R):
27            arr[k] = R[j]
28            j += 1
29            k += 1
30
31 # Test Case 1
32 a = [31, 23, 35, 27, 11, 21, 15, 28]
33 merge_sort(a)
34 print("Sorted Array:", a)
35
36 # Test Case 2
37 b = [22, 34, 25, 36, 43, 67, 52, 13, 65, 17]
38 merge_sort(b)
39 print("Sorted Array:", b)
```

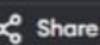
Sorted Array: [11, 15, 21, 23, 27, 28, 31, 35]
Sorted Array: [13, 17, 22, 25, 34, 36, 43, 52, 65, 67]
*** Code Execution Successful ***

The image shows a Jupyter Notebook interface with the following components:

- File Icons:** On the far left, there is a vertical column of icons for different file types: Python (blue), SQL (green), Markdown (yellow), Text (orange), JSON (red), YAML (purple), and others.
- Title Bar:** The title "main.py" is at the top left. To its right are icons for opening, saving, sharing, and running the code. A blue "Run" button is highlighted.
- Output Tab:** The tab "Output" is selected, indicated by a blue border.
- Code Area:** The main area contains the following Python code:

```
1 def champagneTower(poured, query_row, query_glass):
2     dp = [[0] * 101 for _ in range(101)]
3     dp[0][0] = poured
4
5     for i in range(100):
6         for j in range(i + 1):
7             if dp[i][j] > 1:
8                 excess = dp[i][j] - 1
9                 dp[i][j] = 1
10                dp[i+1][j] += excess / 2
11                dp[i+1][j+1] += excess / 2
12
13    return dp[query_row][query_glass]
14
15 poured = 1
16 query_row = 1
17 query_glass = 1
18
19 result = champagneTower(poured, query_row, query_glass)
20 print(result)
```
- Output Panel:** The output panel shows the result of the code execution:

0
--- Code Execution Successful ---
- Clear Button:** A "Clear" button is located in the top right corner of the output panel.



```
1 def stringMatching(words):
2     result = []
3     n = len(words)
4
5     for i in range(n):
6         for j in range(n):
7             if i != j and words[i] in words[j]:
8                 result.append(words[i])
9                 break
10
11    return result
12
13 print(stringMatching(["mass","as","hero","superhero"]))
14 print(stringMatching(["leetcode","et","code"]))
15 print(stringMatching(["blue","green","bu"]))
```

```
['as', 'hero']
['et', 'code']
[]
```

== Code Execution Successful ==

```
1 from typing import List
2 def full_justify(words: List[str], maxWidth: int) -> List[str]:
3     res = []
4     n = len(words)
5     i = 0
6
7     while i < n:
8         line_len = len(words[i])
9         j = i + 1
10        while j < n and line_len + 1 + len(words[j]) <= maxWidth:
11            line_len += 1 + len(words[j])
12            j += 1
13
14        line_words = words[i:j]
15        num_words = j - i
16        total_chars = sum(len(word) for word in line_words)
17        spaces_needed = maxWidth - total_chars
18        if j == n or num_words == 1:
19            line = " ".join(line_words)
20            line += " " * (maxWidth - len(line))
21        else:
22            spaces_between_words = spaces_needed // (num_words - 1)
23            extra_spaces = spaces_needed % (num_words - 1)
24
25            line = ""
26            for k in range(num_words - 1):
27                line += line_words[k]
28                line += " " * (spaces_between_words + (1 if k < extra_spaces else 0))
29            line += line_words[-1]
30
31        res.append(line)
32        i = j
33
34    return res
35
36 words1 = ["This", "is", "an", "example", "of", "text", "justification."]
37 maxWidth1 = 16
38 print(full_justify(words1, maxWidth1))
39
40 words2 = ["What", "must", "be", "acknowledgment", "shall", "be"]
41 maxWidth2 = 16
42 print(full_justify(words2, maxWidth2))
```

```
['This is an', 'example of text', 'justification. ']
['What must be', 'acknowledgment', 'shall be be']

--- Code Execution Successful ---
```



main.py



Run

Output

Clear

```
1 - def minPatches(coins, target):
2     coins.sort()
3     miss = 1
4     added = 0
5     i = 0
6     n = len(coins)
7
8     while miss <= target:
9         if i < n and coins[i] <= miss:
10            miss += coins[i]
11            i += 1
12        else:
13            miss += miss
14            added += 1
15
16    return added
17
18 print(minPatches([1, 4, 10], 19))
19 print(minPatches([1, 4, 10, 5, 7, 19], 19))
```

2

1

==== Code Execution Successful ===



main.py



Run

Output

Clear

```
1 def word_break(s: str, wordDict: list[str]) -> bool:
2     word_set = set(wordDict)
3     n = len(s)
4     dp = [False] * (n + 1)
5     dp[0] = True
6
7     for i in range(1, n + 1):
8         for j in range(i):
9             if dp[j] and s[j:i] in word_set:
10                 dp[i] = True
11                 break
12
13     return dp[n]
14
15 print(word_break("leetcode", ["leet", "code"]))
16 print(word_break("applepenapple", ["apple", "pen"]))
17 print(word_break("catsandog", ["cats", "dog", "sand", "and", "cat"]))
```

True
True
False

*** Code Execution Successful ***

main.py

```
1 - def solveSudoku(board):
2 -     def is_valid(r, c, ch):
3 -         for i in range(9):
4 -             if board[r][i] == ch or board[i][c] == ch:
5 -                 return False
6 -         start_row, start_col = 3 * (r // 3), 3 * (c // 3)
7 -         for i in range(start_row, start_row + 3):
8 -             for j in range(start_col, start_col + 3):
9 -                 if board[i][j] == ch:
10 -                     return False
11 -         return True
12
13 -     def solve():
14 -         for r in range(9):
15 -             for c in range(9):
16 -                 if board[r][c] == '.':
17 -                     for ch in '123456789':
18 -                         if is_valid(r, c, ch):
19 -                             board[r][c] = ch
20 -                             if solve():
21 -                                 return True
22 -                             board[r][c] = '.'
23 -                     return False
24 -     return True
25
26 - solve()
27
28 - board = [
29 -     ["5", "3", ".", ".", "7", ".", ".", "5", "6"],
30 -     ["6", ".", "7", "1", "9", "5", "3", "4", "8"],
31 -     [".", "9", "8", ".", "3", "6", "7", "2", "."],
32 -     ["8", "7", "6", "2", "5", "3", "1", "4", "9"],
33 -     ["4", "2", "8", "7", "3", "6", "9", "5", "1"],
34 -     ["7", "5", "2", "1", "8", "4", "3", "6", "9"],
35 -     [".", "6", "1", "4", "9", "2", "8", "7", "."],
36 -     [".", "1", "4", "7", "2", "5", "3", "6", "8"],
37 -     [".", "2", "8", "3", "6", "7", "4", "9", "5"]
38 - ]
39
40 - solveSudoku(board)
41
42 - for row in board:
43 -     print(row)
```



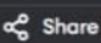
Clear

```
['5', '3', '4', '6', '7', '8', '9', '1', '2'],
['6', '7', '2', '1', '9', '5', '3', '4', '8'],
['1', '9', '8', '3', '4', '2', '5', '6', '7'],
['8', '5', '9', '7', '6', '1', '4', '2', '3'],
['4', '2', '6', '8', '5', '3', '7', '9', '1'],
['7', '1', '3', '9', '2', '4', '8', '5', '6'],
['9', '6', '1', '5', '3', '7', '2', '8', '4'],
['2', '8', '4', '1', '9', '6', '3', '5'],
['3', '4', '5', '2', '8', '6', '1', '7', '9']
```

--- Code Execution Successful ---



main.py



Run

Output

Clear

```
1 def minContainers(weights, maxCapacity):
2     containers = 0
3     currentLoad = 0
4     for weight in weights:
5         if currentLoad + weight <= maxCapacity:
6             currentLoad += weight
7         else:
8             containers += 1
9             currentLoad = weight
10
11    if currentLoad > 0:
12        containers += 1
13
14    return containers
15
16 weights1 = [5, 10, 15, 20, 25, 30, 35]
17 maxCapacity1 = 50
18 print(minContainers(weights1, maxCapacity1))
```

4

== Code Execution Successful ==

main.py

Run Output Clear

```
1- class DisjointSet:
2-     def __init__(self, n):
3-         self.parent = [i for i in range(n)]
4-         self.rank = [0] * n
5-     def find(self, u):
6-         if self.parent[u] != u:
7-             self.parent[u] = self.find(self.parent[u])
8-         return self.parent[u]
9-     def union(self, u, v):
10-        u_root = self.find(u)
11-        v_root = self.find(v)
12-        if u_root == v_root:
13-            return False
14-        if self.rank[u_root] < self.rank[v_root]:
15-            self.parent[u_root] = v_root
16-        elif self.rank[u_root] > self.rank[v_root]:
17-            self.parent[v_root] = u_root
18-        else:
19-            self.parent[v_root] = u_root
20-            self.rank[u_root] += 1
21-        return True
22- def kruskal(n, edges):
23-     edges.sort(key=lambda x: x[2])
24-     ds = DisjointSet(n)
25-     mst = []
26-     total_weight = 0
27-
28-     for u, v, w in edges:
29-         if ds.union(u, v):
30-             mst.append((u, v, w))
31-             total_weight += w
32-
33-     return mst, total_weight
34- n = 4
35- edges = [(0, 1, 10), (0, 2, 6), (0, 3, 5), (1, 3, 15), (2, 3, 4)]
36- mst, total_weight = kruskal(n, edges)
37- print("Edges in MST:", mst)
38- print("Total weight of MST:", total_weight)
```

Edges in MST: [(2, 3, 4), (0, 3, 5), (0, 1, 10)]
Total weight of MST: 19
--- Code Execution Successful ---

main.py



Run

Output

Clear

```
1 - def jobScheduling(startTime, endTime, profit):
2     import bisect
3     jobs = sorted(zip(startTime, endTime, profit), key=lambda x: x[1])
4     n = len(jobs)
5     dp = [0] * (n + 1)
6     end_times = [job[1] for job in jobs]
7
8     for i in range(1, n + 1):
9         s, e, p = jobs[i - 1]
10        j = bisect.bisect_right(end_times, s, 0, i - 1)
11        dp[i] = max(dp[i - 1], dp[j] + p)
12
13    return dp[n]
14
15 print(jobScheduling([1,2,3,3], [3,4,5,6], [50,10,40,70]))
16 print(jobScheduling([1,2,3,4,6], [3,5,10,6,9], [20,20,100,70,60]))
```

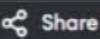
120

150

==== Code Execution Successful ===



main.py



Run

Output

Clear

```
1 import math
2 def dijkstra(graph, src):
3     n = len(graph)
4     dist = [math.inf] * n
5     visited = [False] * n
6     dist[src] = 0
7
8     for _ in range(n):
9         u = min((i for i in range(n) if not visited[i]), key=lambda i:
10             dist[i])
11         visited[u] = True
12
13         for v in range(n):
14             if not visited[v] and graph[u][v] != math.inf and dist[u]
15                 + graph[u][v] < dist[v]:
16                 dist[v] = dist[u] + graph[u][v]
17
18     return dist
19
20 Infinity = math.inf
21 graph = [
22     [0, 10, 3, Infinity, Infinity],
23     [Infinity, 0, 1, 2, Infinity],
24     [Infinity, 4, 0, 8, 2],
25     [Infinity, Infinity, Infinity, 0, 7],
26     [Infinity, Infinity, Infinity, 9, 0]
27 ]
28 print(dijkstra(graph, 0))
```

[0, 7, 3, 9, 5]

== Code Execution Successful ==



main.py



Run

Output

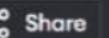
Clear

```
1 class WordFilter:
2     def __init__(self, words: list[str]):
3         self.lookup = {}
4         for index, word in enumerate(words):
5             length = len(word)
6             for i in range(length + 1):
7                 prefix = word[:i]
8                 for j in range(length + 1):
9                     suffix = word[j:]
10                    self.lookup[(prefix, suffix)] = index
11
12    def f(self, pref: str, suff: str) -> int:
13        return self.lookup.get((pref, suff), -1)
14
15 words = ["apple"]
16 wordFilter = WordFilter(words)
17 print(wordFilter.f("a", "e"))
```

0

==== Code Execution Successful ====





```
1 def bubble_sort(arr):  
2     n = len(arr)  
3     for i in range(n):  
4         swapped = False  
5         for j in range(0, n - i - 1):  
6             if arr[j] > arr[j + 1]:  
7                 arr[j], arr[j + 1] = arr[j + 1], arr[j]  
8                 swapped = True  
9         if not swapped:  
10            break  
11     return arr  
12  
13 print(bubble_sort([64, 25, 12, 22, 11]))  
14 print(bubble_sort([29, 10, 14, 37, 13]))
```

```
[11, 12, 22, 25, 64]  
[10, 13, 14, 29, 37]
```

== Code Execution Successful ==



main.py



Run

Output

Clear

```
6
7 def median_of_medians(arr, k):
8     if len(arr) <= 5:
9         return sorted(arr)[k-1]
10
11    groups = [arr[i:i+5] for i in range(0, len(arr), 5)]
12    medians = [sorted(group)[len(group)//2] for group in groups]
13
14    pivot = median_of_medians(medians, (len(medians)+1)//2)
15
16    low, equal, high = partition(arr, pivot)
17
18    if k <= len(low):
19        return median_of_medians(low, k)
20    elif k <= len(low) + len(equal):
21        return pivot
22    else:
23        return median_of_medians(high, k - len(low) - len(equal))
24
25 arr1 = [12, 3, 5, 7, 19]
26 k1 = 2
27 print(median_of_medians(arr1, k1))
28
29 arr2 = [12, 3, 5, 7, 4, 19, 26]
30 k2 = 3
31 print(median_of_medians(arr2, k2))
32
33 arr3 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
34 k3 = 6
35 print(median_of_medians(arr3, k3))
```

```
5
5
6
```

== Code Execution Successful ==



main.py



Run

Output

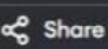
Clear

```
1 def binary_search(arr, key):
2     arr = sorted(arr)
3     low, high = 0, len(arr) - 1
4
5     while low <= high:
6         mid = (low + high) // 2
7         if arr[mid] == key:
8             return mid
9         elif arr[mid] < key:
10            low = mid + 1
11        else:
12            high = mid - 1
13    return -1
14
15 X = [3, 4, 6, -9, 10, 8, 9, 30]
16 KEY = 10
17 pos = binary_search(X, KEY)
18 if pos != -1:
19     print(f"Element {KEY} is found at position {pos+1}")
20 else:
21     print(f"Element {KEY} is not found")
22
23 KEY = 100
24 pos = binary_search(X, KEY)
25 if pos != -1:
26     print(f"Element {KEY} is found at position {pos+1}")
27 else:
28     print(f"Element {KEY} is not found")
```

Element 10 is found at position 7
Element 100 is not found
== Code Execution Successful ==



main.py



Run

Output

Clear

```
1 def count_common_indices(nums1, nums2):  
2     set_nums2 = set(nums2)  
3     set_nums1 = set(nums1)  
4  
5     answer1 = sum(1 for x in nums1 if x in set_nums2)  
6     answer2 = sum(1 for x in nums2 if x in set_nums1)  
7  
8     return [answer1, answer2]  
9  
10 nums1 = [2, 3, 2]  
11 nums2 = [1, 2]  
12 print(count_common_indices(nums1, nums2))
```

[2, 1]

--- Code Execution Successful ---



main.py



Run

Output

Clear

```
1 def dice_throw(num_sides, num_dice, target):
2     dp = [[0] * (target + 1) for _ in range(num_dice + 1)]
3     dp[0][0] = 1
4
5     for dice in range(1, num_dice + 1):
6         for sum_value in range(1, target + 1):
7             for face in range(1, num_sides + 1):
8                 if sum_value - face >= 0:
9                     dp[dice][sum_value] += dp[dice - 1][sum_value - face]
10
11     return dp[num_dice][target]
12
13 print("Test Case 1:")
14 print("Number of ways to reach sum 7:", dice_throw(6, 2, 7))
15
16 print("\nTest Case 2:")
17 print("Number of ways to reach sum 10:", dice_throw(4, 3, 10))
```

Test Case 1:

Number of ways to reach sum 7: 6

Test Case 2:

Number of ways to reach sum 10: 6

== Code Execution Successful ==



main.py



Run

Output

Clear

Toggle light mode

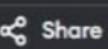
```
1 import math
2 def closest_pair(points):
3     n = len(points)
4     min_dist = float('inf')
5     pair = (None, None)
6
7     for i in range(n):
8         for j in range(i + 1, n):
9             dist = math.sqrt((points[i][0] - points[j][0])**2 +
10                               (points[i][1] - points[j][1])**2)
11
12         if dist < min_dist:
13             min_dist = dist
14             pair = (points[i], points[j])
15
16     return pair, min_dist
17
18 points = [(1, 2), (4, 5), (7, 8), (3, 1)]
19 result, distance = closest_pair(points)
20
21 print("Closest pair:", result[0], "-", result[1])
22 print("Minimum distance:", distance)
```

Closest pair: (1, 2) - (3, 1)
Minimum distance: 2.23606797749979

--- Code Execution Successful ---



main.py



Run

Output

Clear

```
1 def find_max_element(arr):  
2     max_element = arr[0]  
3     for num in arr[1:]:  
4         if num > max_element:  
5             max_element = num  
6     return max_element  
7  
8 print(find_max_element([1, 2, 3, 4, 5]))  
9 print(find_max_element([7, 7, 7, 7, 7]))  
10 print(find_max_element([-10, 2, 3, -4, 5]))
```

5
7
5

==== Code Execution Successful ===



main.py



Run

Output

Clear

```
1 def heapify(a,n,i):
2     l,r=2*i+1,2*i+2; m=i
3     if l<n and a[l]>a[m]: m=l
4     if r<n and a[r]>a[m]: m=r
5     if m!=i: a[i],a[m]=a[m],a[i]; heapify(a,n,m)
6
7 def heap_sort(a):
8     n=len(a)
9     for i in range(n//2-1,-1,-1): heapify(a,n,i)
10    for i in range(n-1,0,-1):
11        a[0],a[i]=a[i],a[0]; heapify(a,i,0)
12    return a
13
14 print(heap_sort([5,2,3,1]))
15 print(heap_sort([5,1,1,2,0,0]))
```

```
[1, 2, 3, 5]
[0, 0, 1, 1, 2, 5]
```

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

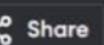


JS

TS



-go



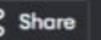
```
1 import itertools
2
3 def total_cost(assignment, cost_matrix):
4     return sum(cost_matrix[i][assignment[i]] for i in range
5                 (len(assignment)))
6
7 def assignment_problem(cost_matrix):
8     n = len(cost_matrix)
9     min_cost = float('inf')
10    best_assignment = None
11
12    for perm in itertools.permutations(range(n)):
13        cost = total_cost(perm, cost_matrix)
14        if cost < min_cost:
15            min_cost = cost
16            best_assignment = perm
17
18    assignment_pairs = [(f"worker {i+1}", f"task
19                        {best_assignment[i]+1}") for i in range(n)]
20
21    return assignment_pairs, min_cost
22
23 cost_matrix2 = [[15, 9, 4],
24                  [8, 7, 18],
25                  [6, 12, 11]]
26
27 assignment2, cost2 = assignment_problem(cost_matrix2)
28 print("\nTest Case 2:")
29 print("Optimal Assignment:", assignment2)
30 print("Total Cost:", cost2)
```

Test Case 2:

Optimal Assignment: [('worker 1', 'task 3'), ('worker 2', 'task 2'),
('worker 3', 'task 1')]

Total Cost: 17

== Code Execution Successful ==



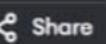
```
1 from collections import Counter
2 def four_sum_count(A, B, C, D):
3
4     AB_sum = Counter(a + b for a in A for b in B)
5
6     count = 0
7     for c in C:
8         for d in D:
9             count += AB_sum.get(-(c + d), 0)
10    return count
11
12 A = [1, 2]
13 B = [-2, -1]
14 C = [-1, 2]
15 D = [0, 2]
16 print(four_sum_count(A, B, C, D))
17
18 A = [0]
19 B = [0]
20 C = [0]
21 D = [0]
22 print(four_sum_count(A, B, C, D))
```

2
1

==== Code Execution Successful ===



main.py



Run

Clear

```
1 def findPeakElement(nums):
2     left, right = 0, len(nums) - 1
3
4     while left < right:
5         mid = (left + right) // 2
6         if nums[mid] < nums[mid + 1]:
7             left = mid + 1
8         else:
9             right = mid
10    return left
11 print(findPeakElement([1, 2, 3, 1]))
12 print(findPeakElement([1, 2, 1, 3, 5, 6, 4]))
```

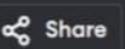
Toggle light mode

2
5

== Code Execution Successful ==



main.py



Run

Output

Clear

```
1 from itertools import combinations
2
3 def knapsack(w, v, C):
4     best = ([], 0)
5     for r in range(len(w)+1):
6         for s in combinations(range(len(w)), r):
7             if sum(w[i] for i in s) <= C:
8                 val = sum(v[i] for i in s)
9                 if val > best[1]: best = (s, val)
10    return best
11
12 # Test Case 1
13 print("Test 1:", knapsack([2,3,1],[4,5,3],4))
14 # Test Case 2
15 print("Test 2:", knapsack([1,2,3,4],[2,4,6,3],6))
```

Test 1: ((1, 2), 8)

Test 2: ((0, 1, 2), 12)

== Code Execution Successful ==



main.py



Run

Output

Clear

```
1- def process_list(lst):
2-     if not lst:
3-         return []
4-     return sorted(lst)
5-
6- test_cases = [
7-     [],
8-     [1],
9-     [7, 7, 7, 7],
10-    [-5, -1, -3, -2, -4]
11- ]
12-
13- for i, lst in enumerate(test_cases, 1):
14-     result = process_list(lst)
15-     print(f"Test Case {i}: Input: {lst} -> Output: {result}")
```

Test Case 1: Input: [] -> Output: []

Test Case 2: Input: [1] -> Output: [1]

Test Case 3: Input: [7, 7, 7, 7] -> Output: [7, 7, 7, 7]

Test Case 4: Input: [-5, -1, -3, -2, -4] -> Output: [-5, -4, -3, -2, -1]

--- Code Execution Successful ---



main.py



Share

Run

Output

Clear

```
1 def findPaths(m, n, N, i, j):
2     MOD = 10**9 + 7
3     memo = {}
4
5     def dfs(x, y, steps):
6         if x < 0 or x >= m or y < 0 or y >= n:
7             return 1
8         if steps == 0:
9             return 0
10        if (x, y, steps) in memo:
11            return memo[(x, y, steps)]
12        count = (
13            dfs(x + 1, y, steps - 1) +
14            dfs(x - 1, y, steps - 1) +
15            dfs(x, y + 1, steps - 1) +
16            dfs(x, y - 1, steps - 1)
17        ) % MOD
18        memo[(x, y, steps)] = count
19        return count
20
21    return dfs(i, j, N)
22
23 print(findPaths(2, 2, 2, 0, 0))
```

6

--- Code Execution Successful ---



main.py



Run

Output

Clear

```
1 def insertion_sort(arr):
2     n = len(arr)
3     for i in range(1, n):
4         key = arr[i]
5         j = i - 1
6         while j >= 0 and arr[j] > key:
7             arr[j + 1] = arr[j]
8             j -= 1
9         arr[j + 1] = key
10    return arr
11
12 test_cases = [
13     [3, 1, 4, 1, 5, 9, 2, 6, 5, 3],
14     [5, 5, 5, 5, 5],
15 ]
16
17 for i, lst in enumerate(test_cases, 1):
18     result = insertion_sort(lst.copy())
19     print(f"Test Case {i}:\n Input: {lst}\n -> Output: {result}")
```

Test Case 1:

Input: [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]
-> Output: [1, 1, 2, 3, 3, 4, 5, 5, 6, 9]

Test Case 2:

Input: [5, 5, 5, 5, 5]
-> Output: [5, 5, 5, 5, 5]

==== Code Execution Successful ===

```
1 def is_on_same_side(p1, p2, a, b):
2     val_a = (p2[0] - p1[0]) * (a[1] - p1[1]) - (p2[1] - p1[1]) * (a[0] - p1[0])
3     val_b = (p2[0] - p1[0]) * (b[1] - p1[1]) - (p2[1] - p1[1]) * (b[0] - p1[0])
4     return val_a * val_b >= 0
5
6 def convex_hull_brute_force(points):
7     hull_points = []
8     n = len(points)
9
10    for i in range(n):
11        for j in range(i+1, n):
12            p1 = points[i]
13            p2 = points[j]
14            all_on_same_side = True
15
16            for k in range(n):
17                if k != i and k != j:
18                    if not is_on_same_side(p1, p2, points[k], points[(k+1) % n]):
19                        all_on_same_side = False
20                        break
21
22            if all_on_same_side:
23                if p1 not in hull_points:
24                    hull_points.append(p1)
25                if p2 not in hull_points:
26                    hull_points.append(p2)
27
28    cx = sum(x for x, y in hull_points) / len(hull_points)
29    cy = sum(y for x, y in hull_points) / len(hull_points)
30    hull_points.sort(key=lambda p: (180 + math.degrees(math.atan2(p[1]-cy, p[0]-cx))) % 360
31    )
32
33    return hull_points
34
35 import math
36 points = [(1, 1), (4, 6), (8, 1), (0, 0), (3, 3)]
37 hull = convex_hull_brute_force(points)
38 print("Convex Hull:", hull)
```

Convex Hull: [(0, 0), (8, 1), (4, 6)]

--- Code Execution Successful ---



```
1 def word_break_segmentation(s: str, wordDict: set) -> str:
2     n = len(s)
3     dp = [False] * (n + 1)
4     dp[0] = True
5     prev = [-1] * (n + 1)
6
7     for i in range(1, n + 1):
8         for j in range(i):
9             if dp[j] and s[j:i] in wordDict:
10                 dp[i] = True
11                 prev[i] = j
12                 break
13
14     if not dp[n]:
15         return "No"
16
17     segments = []
18     idx = n
19     while idx > 0:
20         segments.append(s[prev[idx]:idx])
21         idx = prev[idx]
22     segments.reverse()
23     return "Yes: " + " ".join(segments)
24
25 wordDict = {"i", "like", "sam", "sung", "samsung", "mobile", "ice",
26             "cream", "icecream", "man", "go", "mango"}
27 print(word_break_segmentation("ilike", wordDict))
28 print(word_break_segmentation("ilikesamsung", wordDict))
```

Yes: i like
Yes: i like samsung

== Code Execution Successful ==



main.py



Run

Output

Clear

```
1 import math
2
3 def unique_paths(m, n):
4     return math.comb(m + n - 2, m - 1)
5
6 print(unique_paths(7, 3))
7 print(unique_paths(3, 2))
```

28

3

--- Code Execution Successful ---



main.py



Share

Run

Clear

```
1 a = [22,34,35,36,43,67,12,13,15,17]
2 print("Min =", min(a), ", Max =", max(a))
```

Min = 12 , Max = 67

==== Code Execution Successful ====



JS

TS





main.py



Run

Output

Clear

```
1 def longest_palindrome(s: str) -> str:
2     if not s or len(s) == 0:
3         return ""
4
5     start, end = 0, 0
6
7     def expand_around_center(left: int, right: int) -> int:
8         while left >= 0 and right < len(s) and s[left] == s[right]:
9             left -= 1
10            right += 1
11        return right - left - 1
12
13    for i in range(len(s)):
14        len1 = expand_around_center(i, i)
15        len2 = expand_around_center(i, i + 1)
16        max_len = max(len1, len2)
17        if max_len > (end - start):
18            start = i - (max_len - 1) // 2
19            end = i + max_len // 2
20
21    return s[start:end + 1]
22
23 print(longest_palindrome("babad"))
24 print(longest_palindrome("cbbd"))
```

aba
bb

--- Code Execution Successful ---



main.py



Run

Output

Clear

```
1 import heapq
2
3 def k_closest_points(points, k):
4     # Use a min-heap based on distance squared
5     heap = []
6     for x, y in points:
7         dist = x**2 + y**2 # distance squared
8         heapq.heappush(heap, (dist, [x, y]))
9
10    result = []
11    for _ in range(k):
12        result.append(heapq.heappop(heap)[1])
13    return result
14
15 # Test Case 1
16 points1 = [[1,3],[-2,2],[5,8],[0,1]]
17 k1 = 2
18 print(k_closest_points(points1, k1)) # Output: [[-2, 2], [0, 1]]
19
20 # Test Case 2
21 points2 = [[1,3],[-2,2]]
22 k2 = 1
23 print(k_closest_points(points2, k2)) # Output: [[-2, 2]]
24
25 # Test Case 3
26 points3 = [[3,3],[5,-1],[-2,4]]
27 k3 = 2
28 print(k_closest_points(points3, k3)) # Output: [[3, 3], [-2, 4]]
```

```
[[0, 1], [-2, 2]]  
[[-2, 2]]  
[[3, 3], [-2, 4]]
```

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



main.py



Run

Output

Clear

```
1 - def strassen_2x2(A, B):
2     a, b, c, d = A[0][0], A[0][1], A[1][0], A[1][1]
3     e, f, g, h = B[0][0], B[0][1], B[1][0], B[1][1]
4
5     P1 = a * (f - h)
6     P2 = (a + b) * h
7     P3 = (c + d) * e
8     P4 = d * (g - e)
9     P5 = (a + d) * (e + h)
10    P6 = (b - d) * (g + h)
11    P7 = (a - c) * (e + f)
12
13    C11 = P5 + P4 - P2 + P6
14    C12 = P1 + P2
15    C21 = P3 + P4
16    C22 = P1 + P5 - P3 - P7
17
18    C = [[C11, C12],
19          [C21, C22]]
20
21    return C
22
23 A1 = [[1, 7],
24        [3, 5]]
25 B1 = [[1, 3],
26        [7, 5]]
27 C1 = strassen_2x2(A1, B1)
28 print("Product Matrix C:")
29 for row in C1:
30     print(row)
```

Product Matrix C:
[50, 38]
[38, 34]

== Code Execution Successful ==



main.py



Run

Output

Clear

```
1 def gameOfLife(board):
2     m, n = len(board), len(board[0])
3     directions = [(-1,-1), (-1,0), (-1,1), (0,-1), (0,1), (1,-1), (1,0)
4                   ), (1,1)]
5     for i in range(m):
6         for j in range(n):
7             live_neighbors = sum(
8                 0 <= i+di < m and 0 <= j+dj < n and abs(board[i+di][j
9                     +dj]) == 1
10                for di, dj in directions
11            )
12            if board[i][j] == 1 and (live_neighbors < 2 or
13                live_neighbors > 3):
14                board[i][j] = -1
15            if board[i][j] == 0 and live_neighbors == 3:
16                board[i][j] = 2
17
18        for i in range(m):
19            for j in range(n):
20                board[i][j] = 1 if board[i][j] > 0 else 0
21
22    board = [[0,1,0],[0,0,1],[1,1,1],[0,0,0]]
23    result = gameOfLife(board)
24    print(result)
```

[[0, 0, 0], [1, 0, 1], [0, 1, 1], [0, 1, 0]]

== Code Execution Successful ==



main.py



Run

Output

Clear

```
1 - def binary_search(arr, key):
2     low = 0
3     high = len(arr) - 1
4     comparisons = 0
5
6     while low <= high:
7         comparisons += 1
8         mid = (low + high) // 2
9         if arr[mid] == key:
10            return mid, comparisons
11        elif arr[mid] < key:
12            low = mid + 1
13        else:
14            high = mid - 1
15    return -1, comparisons # Key not found
16
17 # Test Case 1
18 a = [5, 10, 15, 20, 25, 30, 35, 40, 45]
19 key = 20
20 index, comp = binary_search(a, key)
21 print(f"Index of {key}: {index}, Comparisons: {comp}")
22
23 # Test Case 2
24 b = [10, 20, 30, 40, 50, 60]
25 key = 50
26 index, comp = binary_search(b, key)
27 print(f"Index of {key}: {index}, Comparisons: {comp}")
28
29 # Test Case 3
30 c = [21, 32, 40, 54, 65, 76, 87]
31 key = 32
32 index, comp = binary_search(c, key)
33 print(f"Index of {key}: {index}, Comparisons: {comp}")
```



```
1 - def quick_sort(arr, low, high):
2 -     if low < high:
3 -         pi = partition(arr, low, high)
4 -         print(f"After partition with pivot {arr[pi]}: {arr}")
5 -         quick_sort(arr, low, pi - 1)
6 -         quick_sort(arr, pi + 1, high)
7 -
8 - def partition(arr, low, high):
9 -     mid = (low + high) // 2
10 -    arr[low], arr[mid] = arr[mid], arr[low] # Move middle element to start
11 -    pivot = arr[low]
12 -    i = low + 1
13 -    j = high
14 -
15 -    while True:
16 -        while i <= j and arr[i] <= pivot:
17 -            i += 1
18 -        while arr[j] > pivot and j >= i:
19 -            j -= 1
20 -        if i <= j:
21 -            arr[i], arr[j] = arr[j], arr[i]
22 -        else:
23 -            break
24 -    arr[low], arr[j] = arr[j], arr[low]
25 -    return j
26 -
27 - # Test Case 1
28 - a = [19, 72, 35, 46, 58, 91, 22, 31]
29 - print("Original Array:", a)
30 - quick_sort(a, 0, len(a)-1)
31 - print("Sorted Array:", a)
32 -
33 - # Test Case 2
34 - b = [31, 23, 35, 27, 11, 21, 15, 28]
35 - print("\nOriginal Array:", b)
36 - quick_sort(b, 0, len(b)-1)
37 - print("Sorted Array:", b)
38 -
39 - # Test Case 3
40 - c = [22, 34, 25, 36, 43, 67, 52, 13, 65, 17]
41 - print("\nOriginal Array:", c)
42 - quick_sort(c, 0, len(c)-1)
43 - print("Sorted Array:", c)
```

```
Original Array: [19, 72, 35, 46, 58, 91, 22, 31]
After partition with pivot 46: [22, 31, 35, 19, 46, 91, 58, 72]
After partition with pivot 31: [19, 22, 31, 35, 46, 91, 58, 72]
After partition with pivot 19: [19, 22, 31, 35, 46, 91, 58, 72]
After partition with pivot 58: [19, 22, 31, 35, 46, 58, 91, 72]
After partition with pivot 91: [19, 22, 31, 35, 46, 58, 72, 91]
Sorted Array: [19, 22, 31, 35, 46, 58, 72, 91]
```

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

```
Original Array: [22, 34, 25, 36, 43, 67, 52, 13, 65, 17]
After partition with pivot 43: [13, 34, 25, 36, 22, 17, 43, 52, 65, 67]
After partition with pivot 25: [22, 17, 13, 25, 36, 34, 43, 52, 65, 67]
After partition with pivot 17: [13, 17, 22, 25, 36, 34, 43, 52, 65, 67]
After partition with pivot 36: [13, 17, 22, 25, 34, 36, 43, 52, 65, 67]
After partition with pivot 65: [13, 17, 22, 25, 34, 36, 43, 52, 65, 67]
Sorted Array: [13, 17, 22, 25, 34, 36, 43, 52, 65, 67]
```

---- Code Execution Successful ----



main.py



Run

Output

Clear

```
1 def large_group_positions(s):
2     res = []
3     i = 0
4     while i < len(s):
5         j = i
6
7         while j < len(s) and s[j] == s[i]:
8             j += 1
9
10        if j - i >= 3:
11            res.append([i, j - 1])
12        i = j
13    return res
14
15 print(large_group_positions("abxxxxxzy"))
```

[[3, 6]]

== Code Execution Successful ==



main.py



Run

Output

Clear

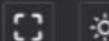
```
1 def bubble_sort(arr):
2     n = len(arr)
3     for i in range(n):
4         for j in range(0, n - i - 1):
5             if arr[j] > arr[j + 1]:
6                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
7     return arr
8
9 print(bubble_sort([64, 34, 25, 12, 22, 11, 90]))
10 print(bubble_sort([5, 1, 4, 2, 8]))
11 print(bubble_sort([3, 3, 3]))
```

```
[11, 12, 22, 25, 34, 64, 90]
```

```
[1, 2, 4, 5, 8]
```

```
[3, 3, 3]
```

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



```
1 def strStr(haystack: str, needle: str) -> int:
2     if not needle:
3         return 0
4
5     for i in range(len(haystack) - len(needle) + 1):
6         if haystack[i:i+len(needle)] == needle:
7             return i
8     return -1
9
10 print(strStr("sadbutsad", "sad"))
11 print(strStr("leetcode", "leeto"))
```

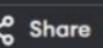
0

-1

==== Code Execution Successful ====



main.py



Run

Output

Clear

```
1 import math
2 def uniquePaths(m, n):
3     return math.comb(m + n - 2, m - 1)
4 print(uniquePaths(3, 7))
5 print(uniquePaths(3, 2))
```

28

3

--- Code Execution Successful ---



main.py



Run

Output

Clear

```
1 - def count_pairs(nums, k):
2     count = 0
3     n = len(nums)
4     for i in range(n):
5         for j in range(i + 1, n):
6             if nums[i] == nums[j] and (i * j) % k == 0:
7                 count += 1
8     return count
9
10 nums = [3, 1, 2, 2, 2, 1, 3]
11 k = 2
12 print(count_pairs(nums, k))
```

4

== Code Execution Successful ==



main.py



Run

Output

Clear

```
1 import itertools
2 import math
3
4 def distance(city1, city2):
5     return math.sqrt((city1[0]-city2[0])**2 + (city1[1]
6                     -city2[1])**2)
7
8 def tsp(cities):
9     start = cities[0]
10    min_dist = float('inf')
11    best_path = []
12
13    for perm in itertools.permutations(cities[1:]):
14        path = [start] + list(perm) + [start]
15        dist = sum(distance(path[i], path[i+1]) for i in range
16                    (len(path)-1))
17        if dist < min_dist:
18            min_dist = dist
19            best_path = path
20
21    cities1 = [(1, 2), (4, 5), (7, 1), (3, 6)]
22    dist1, path1 = tsp(cities1)
23    print("Test Case 1:")
24    print("Shortest Distance:", dist1)
25    print("Shortest Path:", path1)
```

Test Case 1:

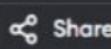
Shortest Distance: 16.969112047670894

Shortest Path: [(1, 2), (7, 1), (4, 5), (3, 6), (1, 2)]

==== Code Execution Successful ===



main.py



Run

Output

Clear

```
1 def find_max_after_sorting(nums):  
2     return None if not nums else sorted(nums)[-1]  
3  
4 print(find_max_after_sorting([]))  
5 print(find_max_after_sorting([5]))  
6 print(find_max_after_sorting([3,2,3,3,3]))
```

None

5

3

--- Code Execution Successful ---



main.py



Run

Output

Clear

```
1 def rob_linear(nums):
2     prev, curr = 0, 0
3     for num in nums:
4         prev, curr = curr, max(curr, prev + num)
5     return curr
6
7 def rob(nums):
8     n = len(nums)
9     if n == 0: return 0
10    if n == 1: return nums[0]
11    return max(rob_linear(nums[:-1]), rob_linear(nums[1:]))
12
13 nums1 = [2, 3, 2]
14 print(f"The maximum money you can rob is {rob(nums1)}")
15 nums2 = [1, 2, 3, 1]
16 print(f"The maximum money you can rob is {rob(nums2)}")
```

The maximum money you can rob is 3
The maximum money you can rob is 4

== Code Execution Successful ==



main.py



Run

Output

Clear

```
1 def climb_stairs(n):
2     if n <= 2:
3         return n
4     a, b = 1, 2
5     for _ in range(3, n + 1):
6         a, b = b, a + b
7     return b
8
9 print(climb_stairs(4))
10 print(climb_stairs(3))
```

5
3

== Code Execution Successful ==



main.py



Run

Clear

```
1 def first_palindrome(words):
2     for word in words:
3         if word == word[::-1]:
4             return word
5     return ""
6
7 words = ["abc", "car", "cool", "malayalam"]
8 print(first_palindrome(words))
```

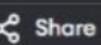
Output

malayalam

== Code Execution Successful ==



main.py



Run

Clear

```
1 def unique_elements(nums):
2     return list(set(nums))
3
4 print(unique_elements([3, 7, 3, 5, 2, 5, 9, 2]))
5 print(unique_elements([-1, 2, -1, 3, 2, -2]))
6 print(unique_elements([1000000, 999999, 1000000]))
```

[2, 3, 5, 7, 9]

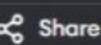
[2, 3, -1, -2]

[1000000, 999999]

== Code Execution Successful ==



main.py



Run

Output

Clear

```
1 def sum_of_squares_distinct_counts(nums):
2     n = len(nums)
3     total = 0
4
5     for i in range(n):
6         seen = set()
7         for j in range(i, n):
8             seen.add(nums[j])
9             distinct_count = len(seen)
10            total += distinct_count ** 2
11
12    return total
13
14 nums = [1, 2, 1]
15 print(sum_of_squares_distinct_counts(nums))
```

15

== Code Execution Successful ==