Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def find_min_max(arr):
    if not arr:
        return None, None

    min_element = arr[0]
    max_element = arr[0]

    for num in arr:
        if num < min_element:
            min_element = num
        if num > max_element:
            max_element = num

    return min_element, max_element
if __name__ == "__main__":
    a = [2, 3, 4, 5, 6, 1, 0, 9]
    min_val, max_val = find_min_max(a)
    print(f"Array: {a}")
    print(f"Minimum element: {min_val}")
    print(f"Maximum element: {max_val}")

```

input

```
Array: [2, 3, 4, 5, 6, 1, 0, 9]
Minimum element: 0
Maximum element: 9


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

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def rob_linear(nums):
    if not nums:
        return 0
    if len(nums) == 1:
        return nums[0]
    n = len(nums)
    dp = [0] * n
    dp[0] = nums[0]
    dp[1] = max(nums[0], nums[1])
    for i in range(2, n):
        dp[i] = max(dp[i-1], dp[i-2] + nums[i])
    return dp[-1]
def rob(nums):
    if not nums:
        return 0
    if len(nums) == 1:
        return nums[0]
    max_excluding_last = rob_linear(nums[:-1])
    max_excluding_first = rob_linear(nums[1:])
    return max(max_excluding_last, max_excluding_first)
if __name__ == "__main__":
    nums1 = [2, 3, 2]
    print(f"Maximum money that can be robbed from {nums1} is {rob(nums1)}")

    nums2 = [1, 2, 3, 1]
    print(f"Maximum money that can be robbed from {nums2} is {rob(nums2)}")

```

input

```
Maximum money that can be robbed from [2, 3, 2] is 3
Maximum money that can be robbed from [1, 2, 3, 1] is 4


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

main.py

```python
import heapq
def dijkstra(graph, source):
    n = len(graph)
    distances = [float('inf')] * n
    distances[source] = 0
    priority_queue = [(0, source)]
    while priority_queue:
        current_distance, current_vertex = heapq.heappop(priority_queue)
        if current_distance > distances[current_vertex]:
            continue
        for neighbor, weight in enumerate(graph[current_vertex]):
            if weight != float('inf'):
                distance = current_distance + weight
                if distance < distances[neighbor]:
                    distances[neighbor] = distance
                    heapq.heappush(priority_queue, (distance, neighbor))
    return distances
graph = [
    [0, 10, 3, float('inf'), float('inf')],
    [float('inf'), 0, 1, 2, float('inf')],
    [float('inf'), 4, 0, 8, 2],
    [float('inf'), float('inf'), float('inf'), 0, 7],
    [float('inf'), float('inf'), float('inf'), 9, 0]
]
source = 0
output = dijkstra(graph, source)
print(output)
```

input

```
[0, 7, 3, 9, 5]


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

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def selection_sort(arr):
    n = len(arr)

    for i in range(n):
        min_index = i
        for j in range(i+1, n):
            if arr[j] < arr[min_index]:
                min_index = j

        arr[i], arr[min_index] = arr[min_index], arr[i]

    return arr

if __name__ == "__main__":
    arr = [64, 25, 12, 22, 11]
    sorted_arr = selection_sort(arr)
    print("Sorted array:", sorted_arr)
```

input

```
Sorted array: [11, 12, 22, 25, 64]


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

main.py

```python
def find_element(arr, target):
    for i, num in enumerate(arr):
        if num == target:
            return i

    return -1

if __name__ == "__main__":
    arr = [2, 3, 4, 5, 6, 1, 0, 9]
    target = 8
    index = find_element(arr, target)

    if index != -1:
        print(f"Element {target} found at index {index}.")
    else:
        print(f"Element {target} not found in the array.")

```

input

```
Element 8 not found in the array.


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

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def binary_search(arr, target):
    left, right = 0, len(arr) - 1

    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
if __name__ == "__main__":
    arr = [5, 10, 15, 20, 25, 30, 35, 40, 45]
    target = 20
    result = binary_search(arr, target)

    if result != -1:
        print(f"Element {target} is present at index {result}.")
    else:
        print(f"Element {target} is not present in the array.")

```

input

```
Element 20 is present at index 3.


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

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def combinationSum(candidates, target):
    def backtrack(remaining, start, path, res):
        if remaining == 0:
            res.append(list(path))
            return
        elif remaining < 0:
            return

        for i in range(start, len(candidates)):
            path.append(candidates[i])
            backtrack(remaining - candidates[i], i, path, res)
            path.pop()

    candidates.sort()
    res = []
    backtrack(target, 0, [], res)
    return res

candidates = [2, 3, 6, 7]
target = 7
output = combinationSum(candidates, target)
print(output)
```

input

```
[[2, 2, 3], [7]]


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

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]
        merge_sort(left_half)
        merge_sort(right_half)
        i = j = k = 0
        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1
        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1
        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1
    return arr
if __name__ == "__main__":
    arr = [38, 27, 43, 3, 9, 82, 10]
    print("Original array:", arr)
```

input

```
Original array: [38, 27, 43, 3, 9, 82, 10]
Sorted array: [3, 9, 10, 27, 38, 43, 82]


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

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
import heapq

def kClosest(points, k):
    heap = []
    for (x, y) in points:
        distance = x*x + y*y
        heapq.heappush(heap, (distance, [x, y]))

    result = []
    for _ in range(k):
        result.append(heapq.heappop(heap)[1])

    return result
points = [[1, 3], [-2, 2], [5, 8], [0, 1]]
k = 2
output = kClosest(points, k)
print(output)
```

input

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


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

Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def graph_coloring(edges, n):
    from collections import defaultdict, deque
    adjacency_list = defaultdict(list)
    for u, v in edges:
        adjacency_list[u].append(v)
        adjacency_list[v].append(u)
    colors = [-1] * n
    max_colors = 3
    your_turn = True
    your_color_count = 0

    def is_safe(vertex, color):
        for neighbor in adjacency_list[vertex]:
            if colors[neighbor] == color:
                return False
        return True
    def color_vertex(vertex, color):
        nonlocal your_color_count, your_turn
        colors[vertex] = color
        if your_turn:
            your_color_count += 1
        your_turn = not your_turn
    queue = deque(range(n))
    while queue:
        vertex = queue.popleft()
        for color in range(max_colors):
            if is_safe(vertex, color):
                color_vertex(vertex, color)
```

input

```
Maximum number of regions you can color: 2


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