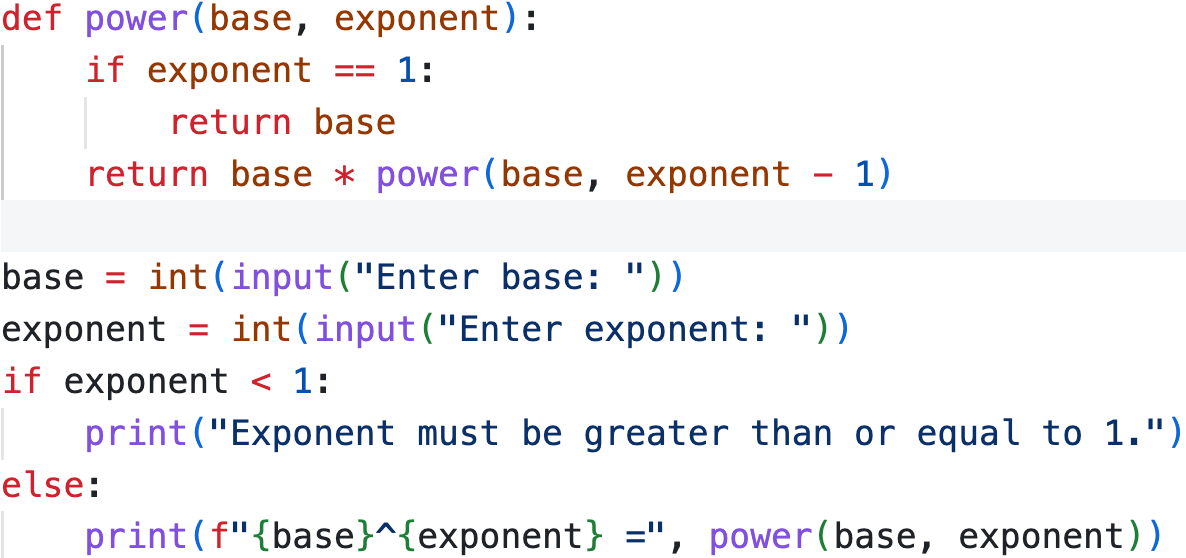
Python for Computer Science and Data Science 2 (CSE 3652)

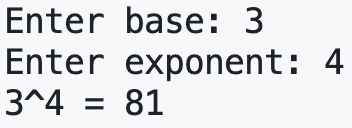
**Minor Assignment-2: COMPUTER SCIENCE THINKING: RECURSION,**

**SEARCHING, SORTING AND BIG O**

1. Write a recursive function power(base, exponent) that, when called, returns *baseexponent* Example: power(3,4) = 3\*3\*3\*3. Both base and exponent are user-defined and assume that the exponent is an integer greater than or equal to 1.

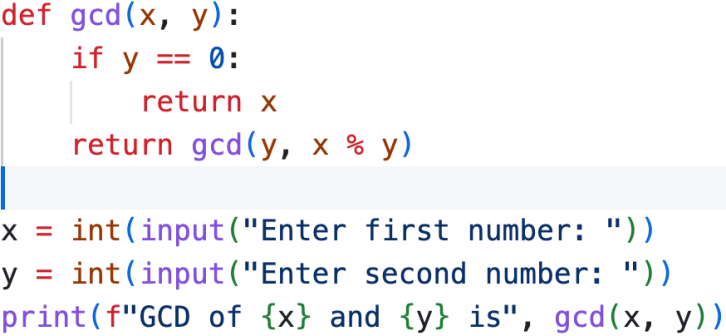
**Ans:-**

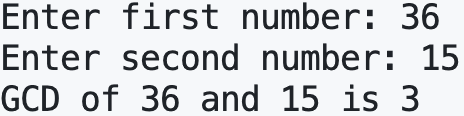


Output:- 

1. The greatest common divisor of integers x and y is the largest integer that evenly divides into both x and y. Write and test a recursive function gcd that returns the greatest common divisor of x and y. The gcd of x and y is defined recursively as follows: If y is equal to 0, then *gcd(x, y)* is x; otherwise, *gcd(x, y)* is *gcd(y, x%y)*?

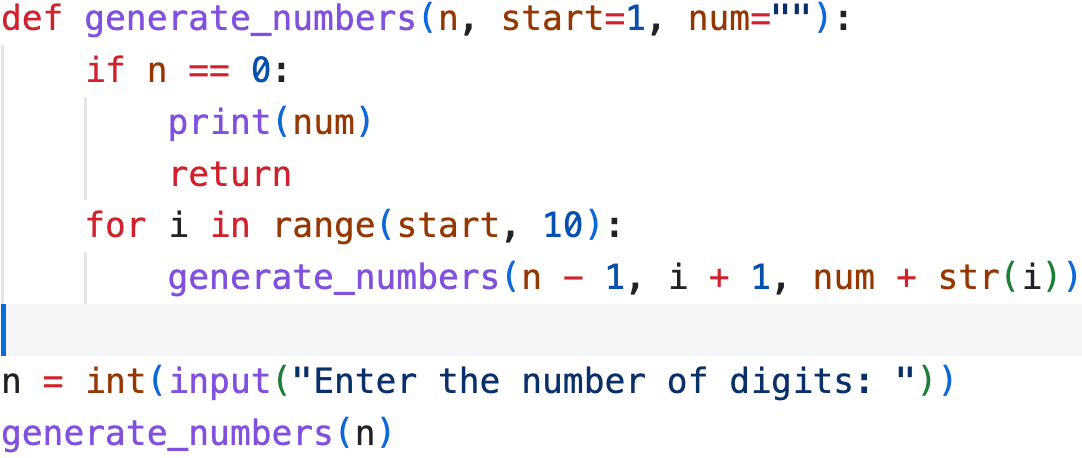
**Ans:-**

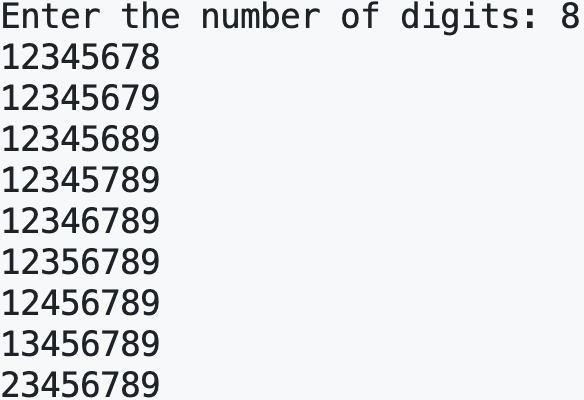


Output:- 

1. Write a recursive function that takes a number n as an input parameter and prints n-digit strictly increasing numbers.

**Ans:-**

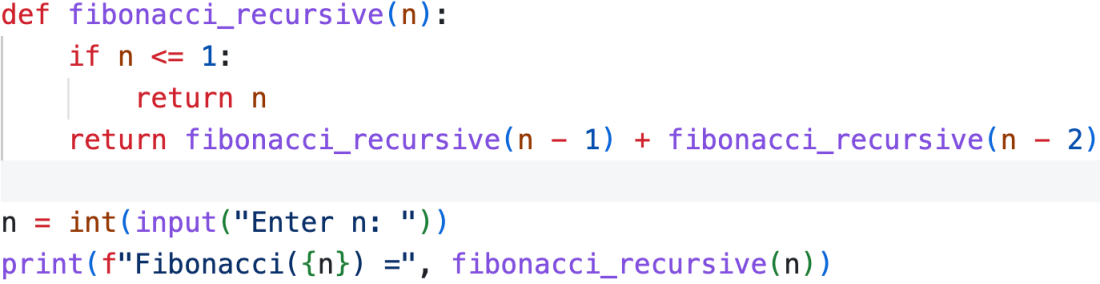


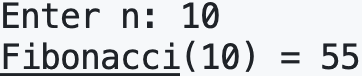
Output:- 

1. Implement a recursive solution for computing the nth Fibonacci number. Then, analyze its time

complexity. Propose a more efficient solution and compare the two approaches.

**Ans:-** Recursive Solution -

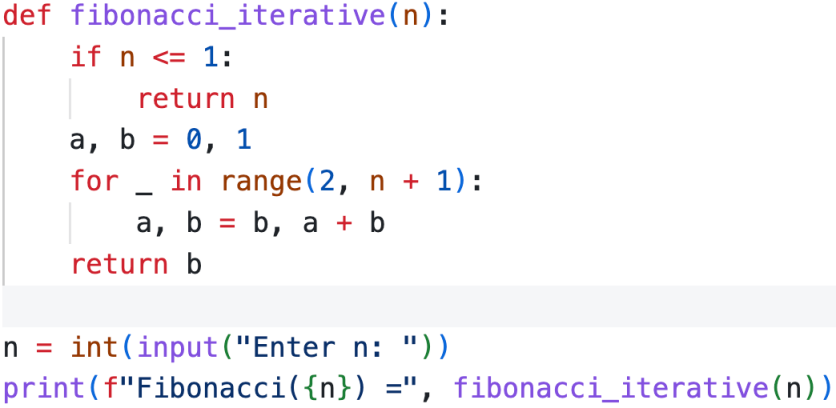


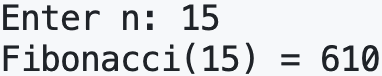
Output:- 

Time Complexity Analysis:

1. Each call to fibonacci\_recursive(n) makes two recursive calls.
2. This results in an exponential time complexity: O(2ⁿ).

Iterative Approach -



Output:- 

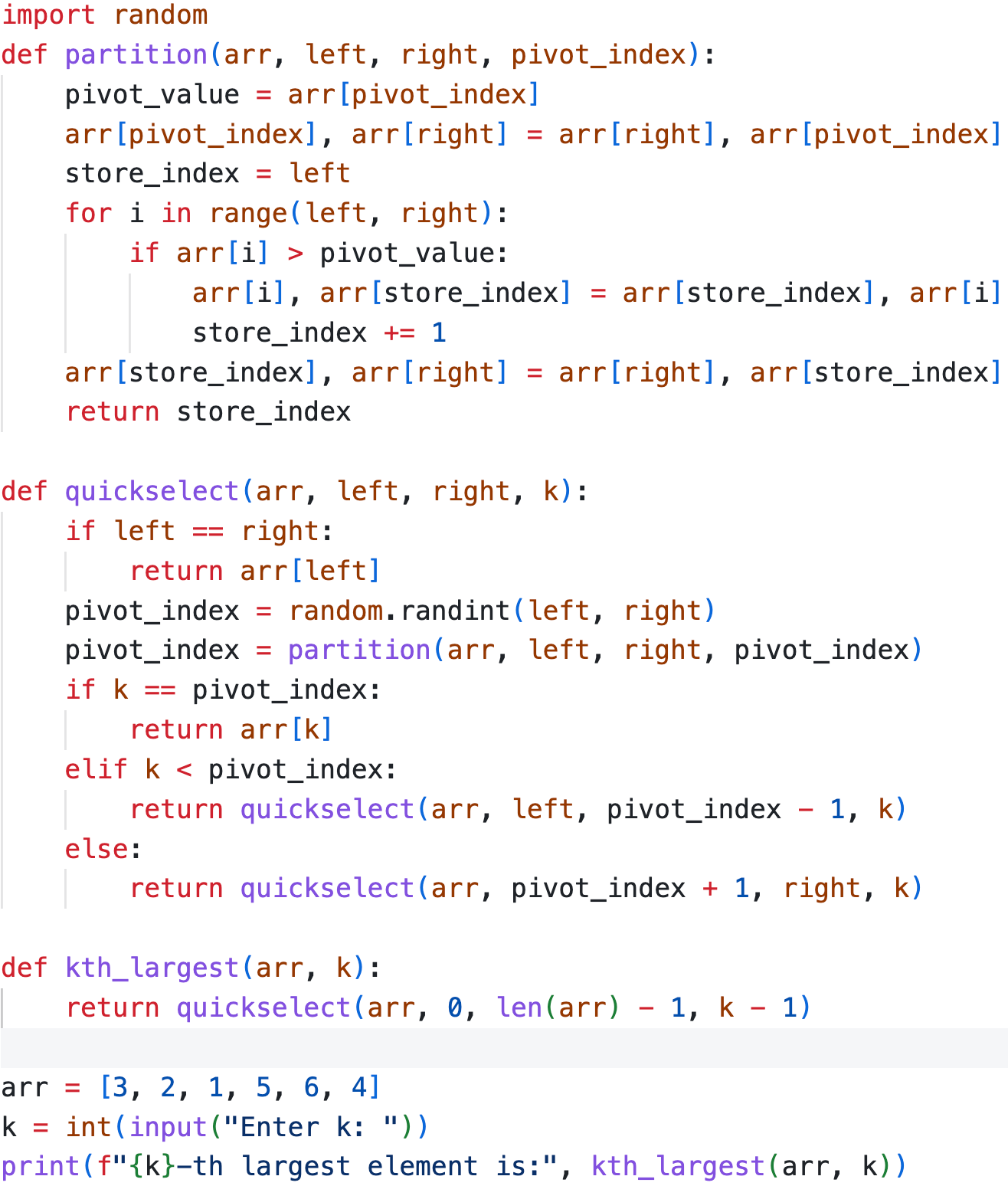
Time Complexity Analysis: O(n), but without the overhead of recursion.

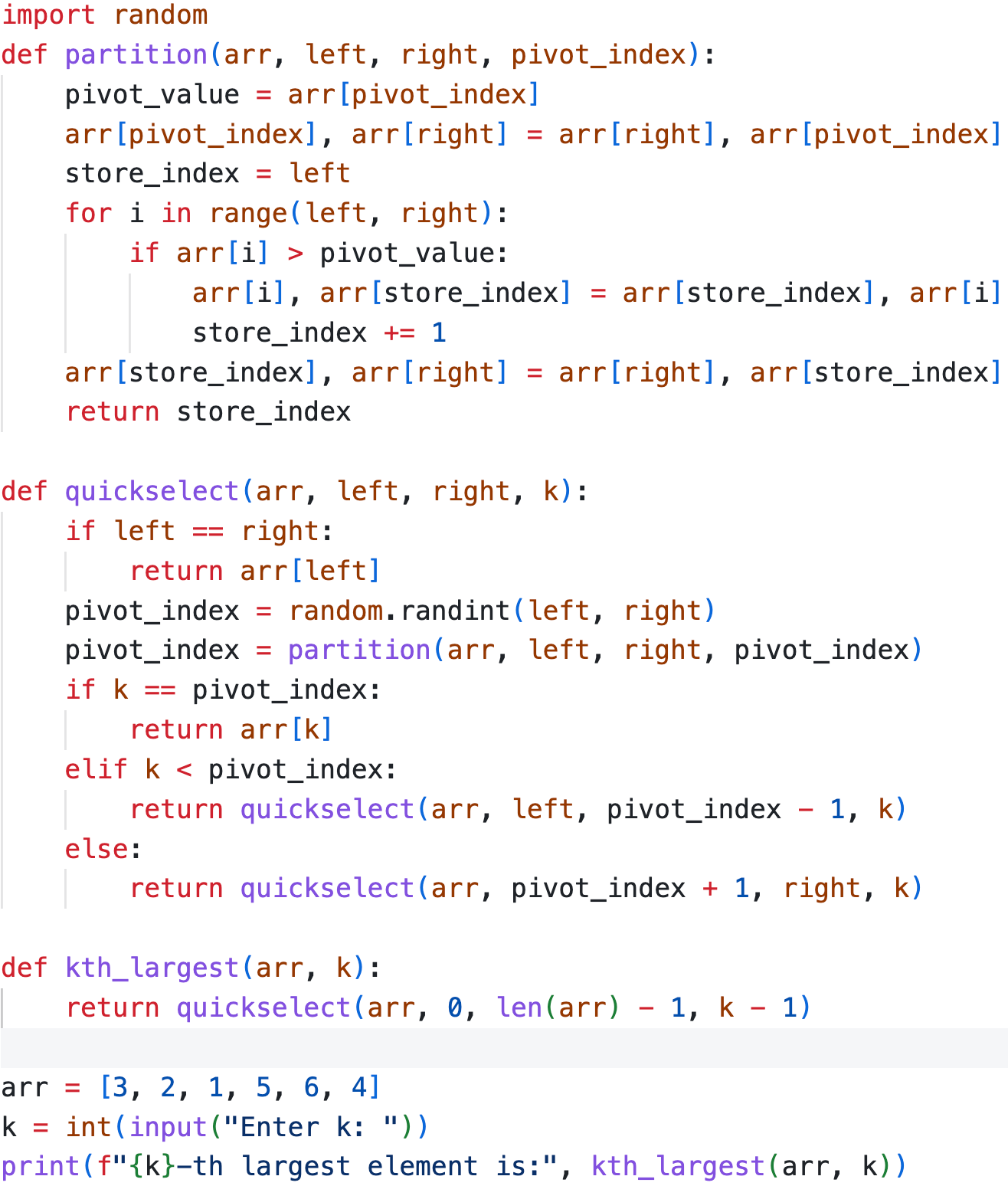
Comparison:

| **Approach** | **Time Complexity** | **Space Complexity** | **Notes** |
| --- | --- | --- | --- |
| Recursive | O(2ⁿ) | O(n) (due to call stack) | Very slow for large n |
| Iterative | O(n) | O(1) | Most efficient in both time & space |

1. Given an array of N elements, not necessarily in ascending order, devised an algorithm to find the kth largest one. It should run in O(N) time on random inputs.

**Ans:-**





Output:- 

1. For each of the following code snippets, determine the time complexity in terms of Big O. Explain your answer.
2. *def example1(n):*

*for i in range(n):*

*for j in range(n):*

*print(i, j)*

**Ans: -** Time Complexity: O(n²)

1. The outer loop runs n times.
2. The inner loop runs n times for each iteration of the outer loop.
3. Total iterations: n × n = n².
4. Big O Complexity: O(n²) (Quadratic time).
5. *for i in range(n):*

*print(i)*

**Ans: -** Time Complexity: O(n)

1. The loop runs n times.
2. Each iteration performs a constant-time operation (O(1)).
3. Big O Complexity: O(n) (Linear time).
4. *def recursive\_function(n):*

*if n <= 1:*

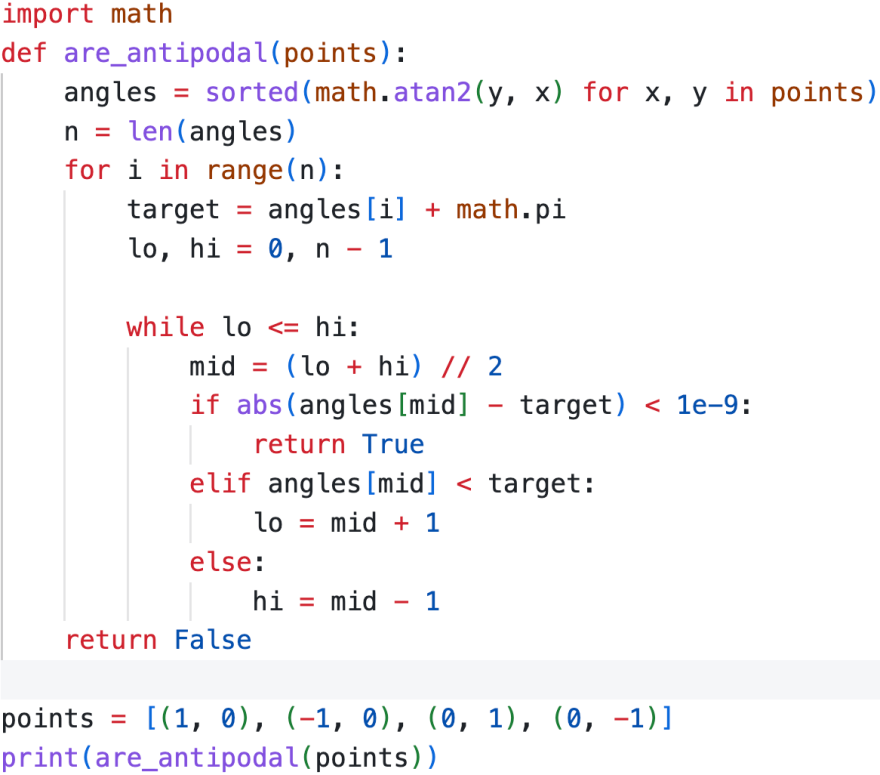
*return 1*

*return recursive\_function(n - 1) + recursive\_function(n - 1)*

**Ans: -** Time Complexity: O(2ⁿ)

1. This is a tree recursion, where each call spawns two recursive calls.
2. The number of recursive calls forms a binary tree of depth n.
3. The total number of calls is approximately 2ⁿ in the worst case.
4. Big O Complexity: O(2ⁿ) (Exponential time).
5. Write Given *N* points on a circle, centered at the origin, design an algorithm that determines whether there are two points that are antipodal, i.e., the line connecting the two points goes through the origin. Your algorithm should run in time proportional to *NlogN*.

**Ans:-**



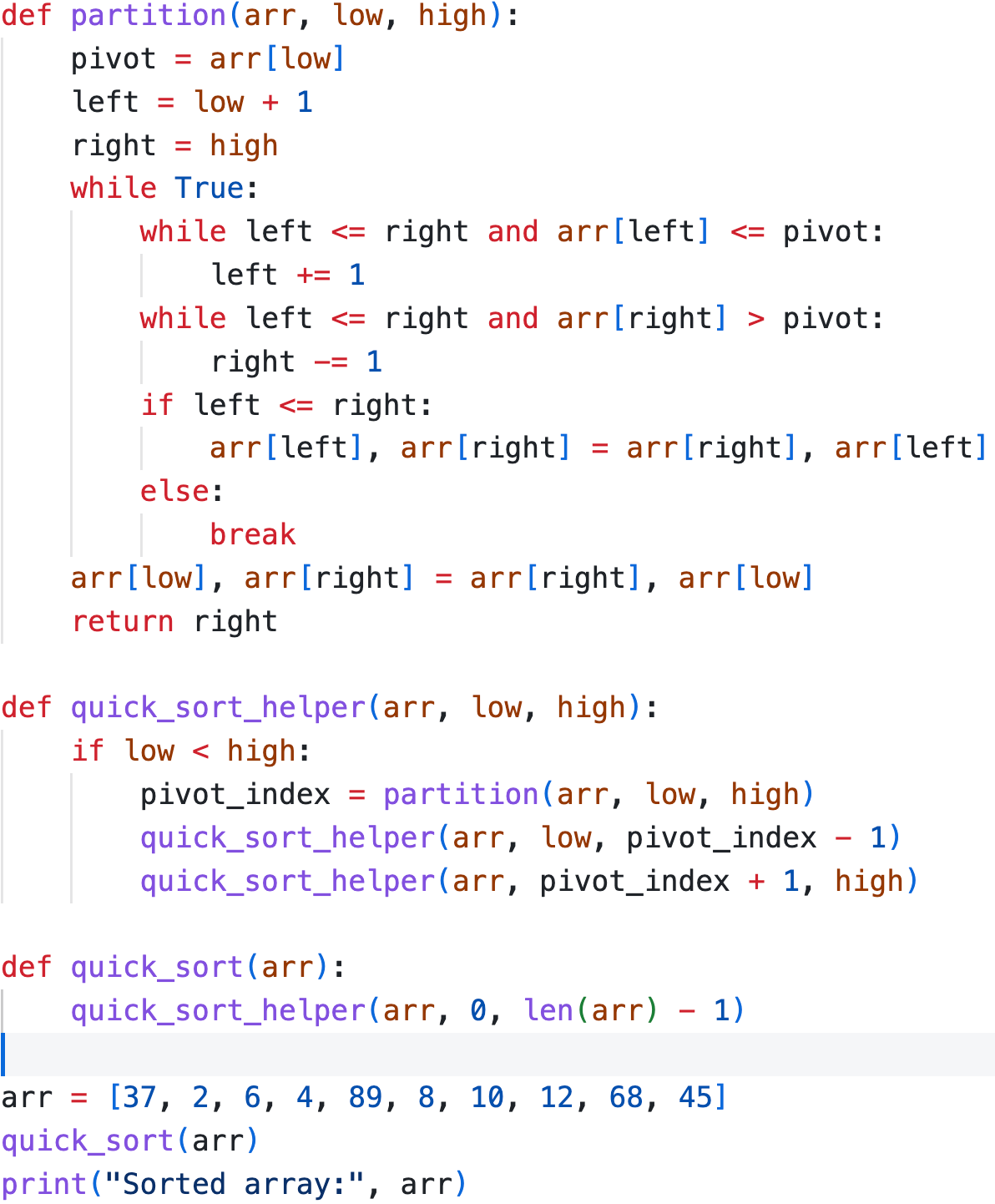
Output:- Screenshot 2025-03-09 at 9.11.34 PM

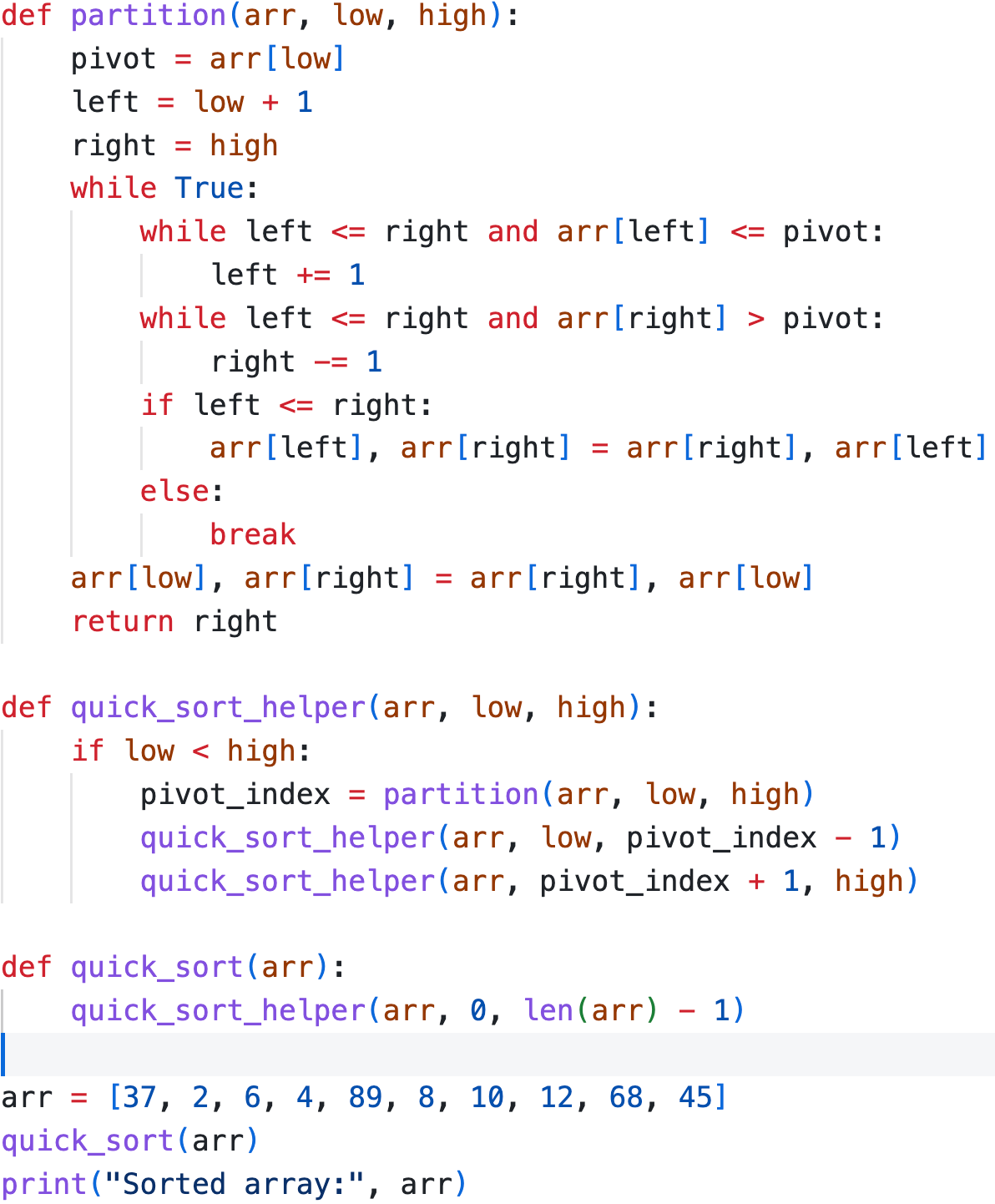
1. The quicksort algorithm is a recursive sorting technique that follows these steps:
2. Partition Step: Choose the first element of the array as the pivot and determine its final position in the sorted array by ensuring all elements to its left are smaller and all elements to its right are larger.
3. Recursive Step: Recursively repeat the partitioning process on the subarrays created on either side of the pivot.

As an example, consider the array [37, 2, 6, 4, 89, 8, 10, 12, 68, 45] with 37 as the pivot. Using the partitioning logic, the pivot eventually moves to its correct position, resulting in two subarrays: [12, 2, 6, 4, 10, 8] and [89, 68, 45]. The algorithm continues recursively until the entire array is sorted.

Write a Python function quick sort that implements the quicksort algorithm. The function should include a helper function quick sort helper to handle recursion. The helper function must take a starting and ending index as arguments and sort the array in-place. Demonstrate the function by sorting the given array and printing the sorted output.

**Ans:-**





Output:- Screenshot 2025-03-09 at 9.34.28 PM

1. You are given the following list of famous personalities with their net worth:

• Elon Musk: $433.9 Billion

• Jeff Bezos: $239.4 Billion

• Mark Zuckerberg: $211.8 Billion

• Larry Ellison: $204.6 Billion

• Bernard Arnault & Family: $181.3 Billion

• Larry Page: $161.4 Billion

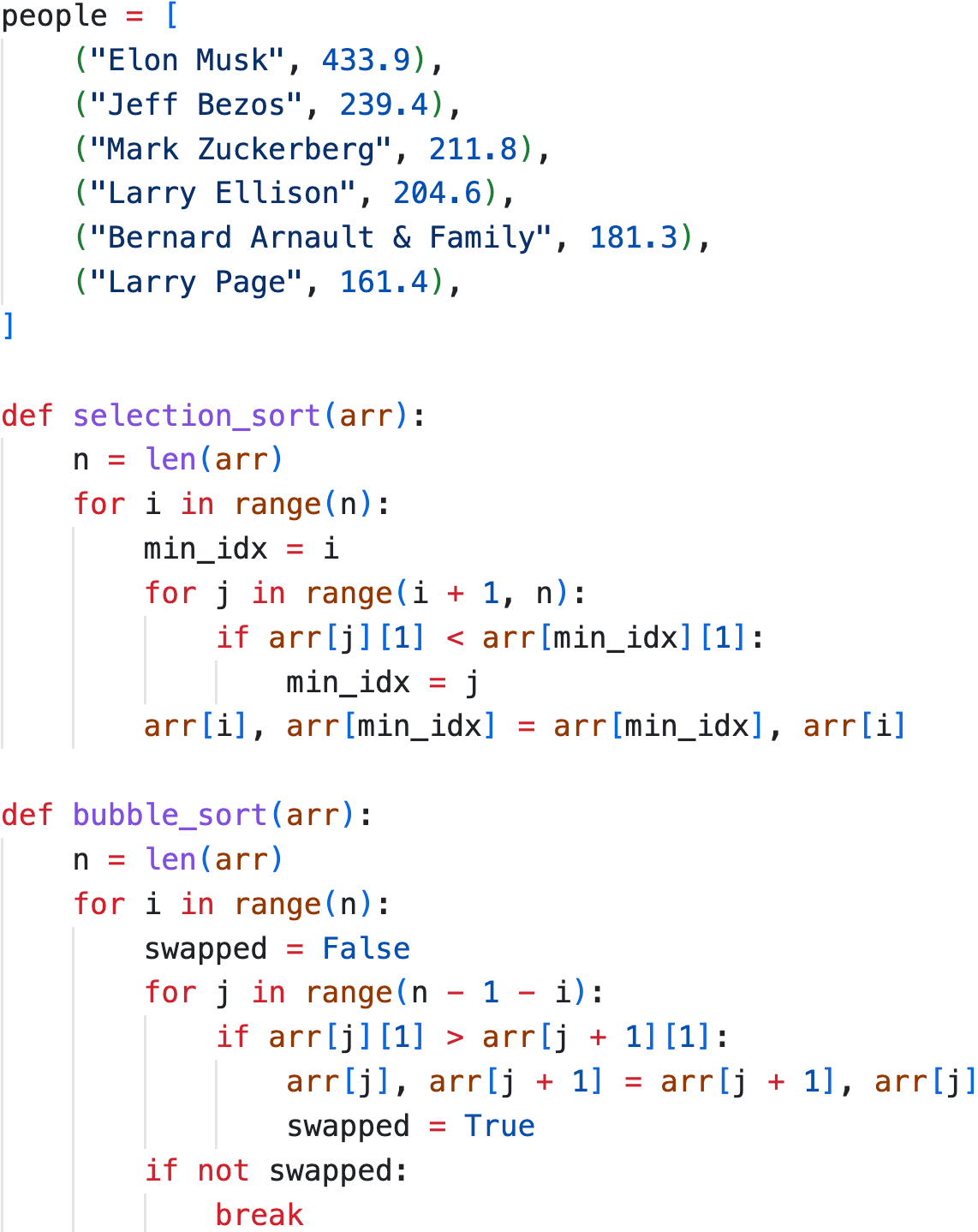
Develop a program to sort the aforementioned details on the basis of net worth using

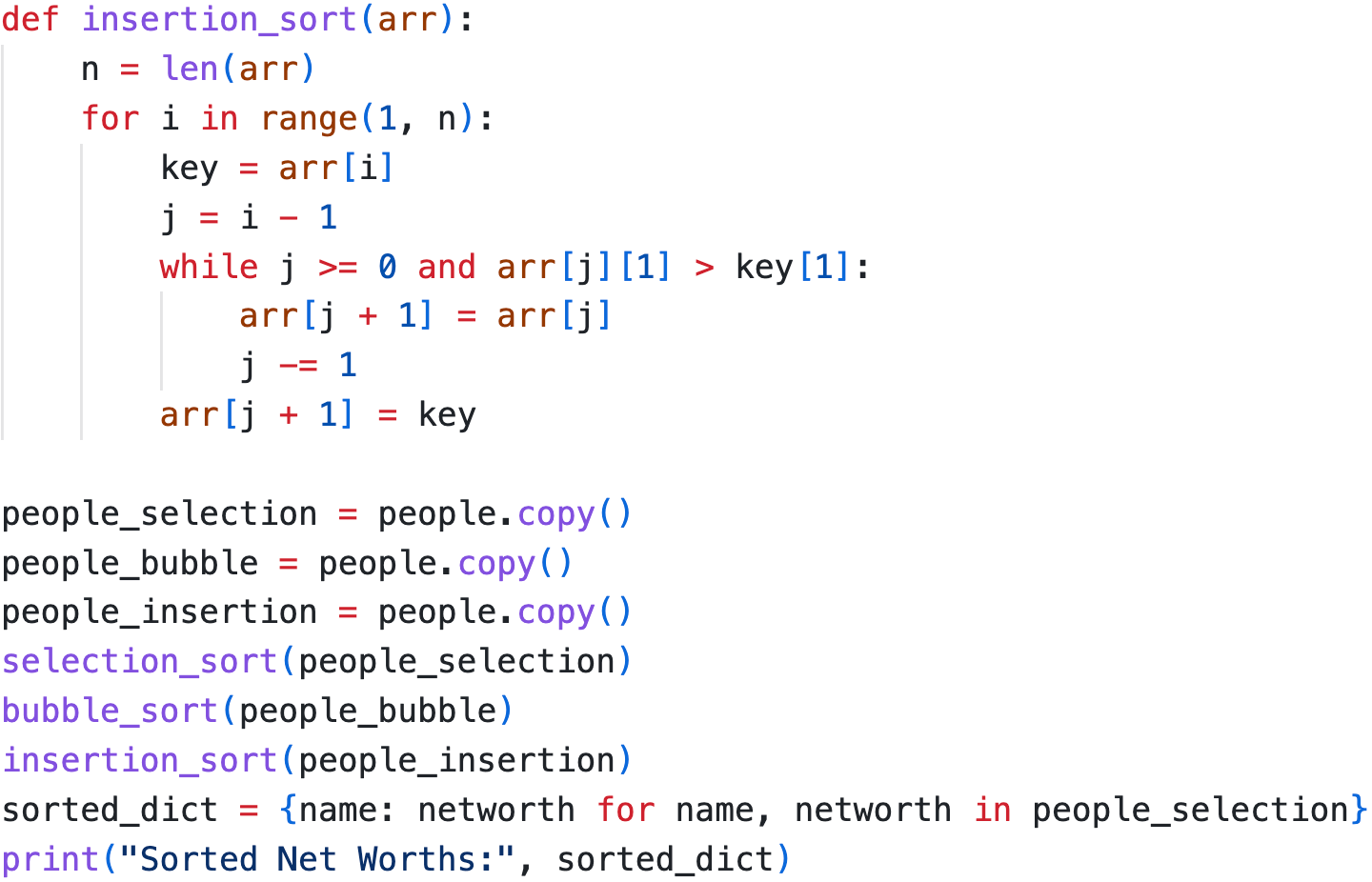
1. Selection sort
2. Bubble sort
3. Insertion sort.

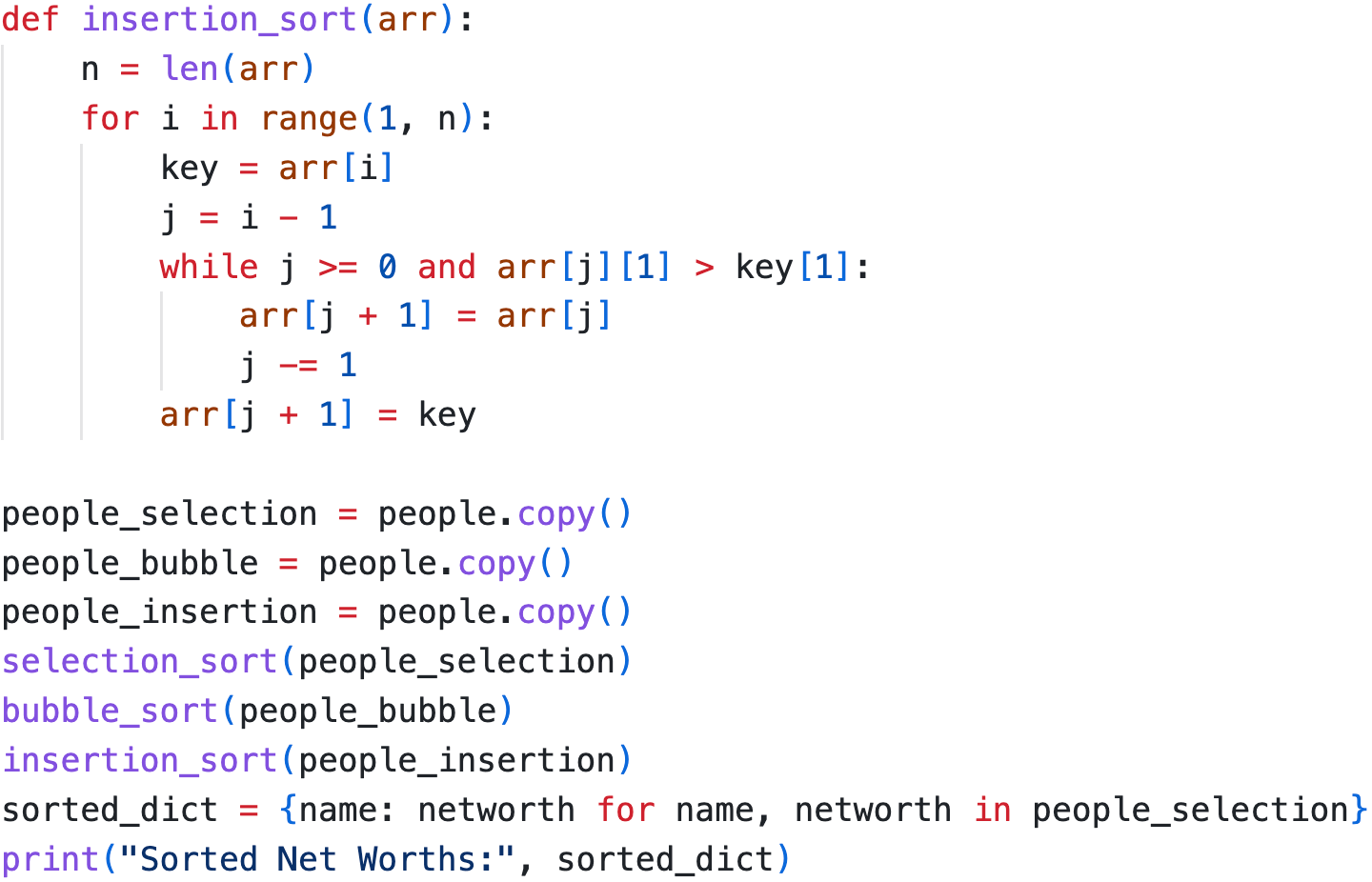
The final sorted data should be the same for all cases. After you obtain the sorted data, present the

result in the form of the following dictionary: {’name1’:networth1, ’name2’:networth2,...}.

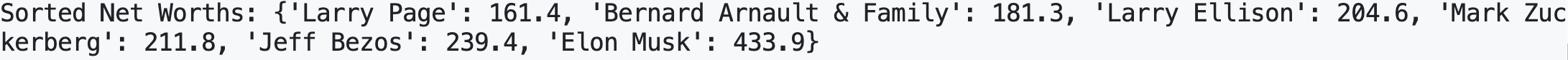
**Ans:-**







Output:-

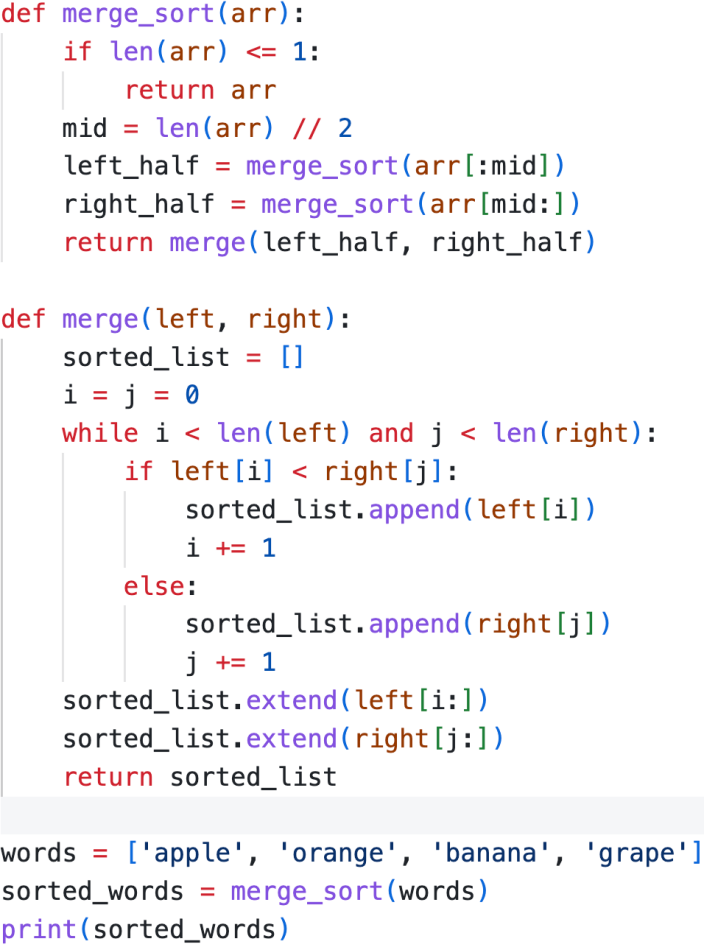


1. Use Merge Sort to sort a list of strings alphabetically. Example:

Input: [’apple’, ’orange’, ’banana’, ’grape’]

Output: [’apple’, ’banana’, ’grape’, ’orange’]

**Ans:-**



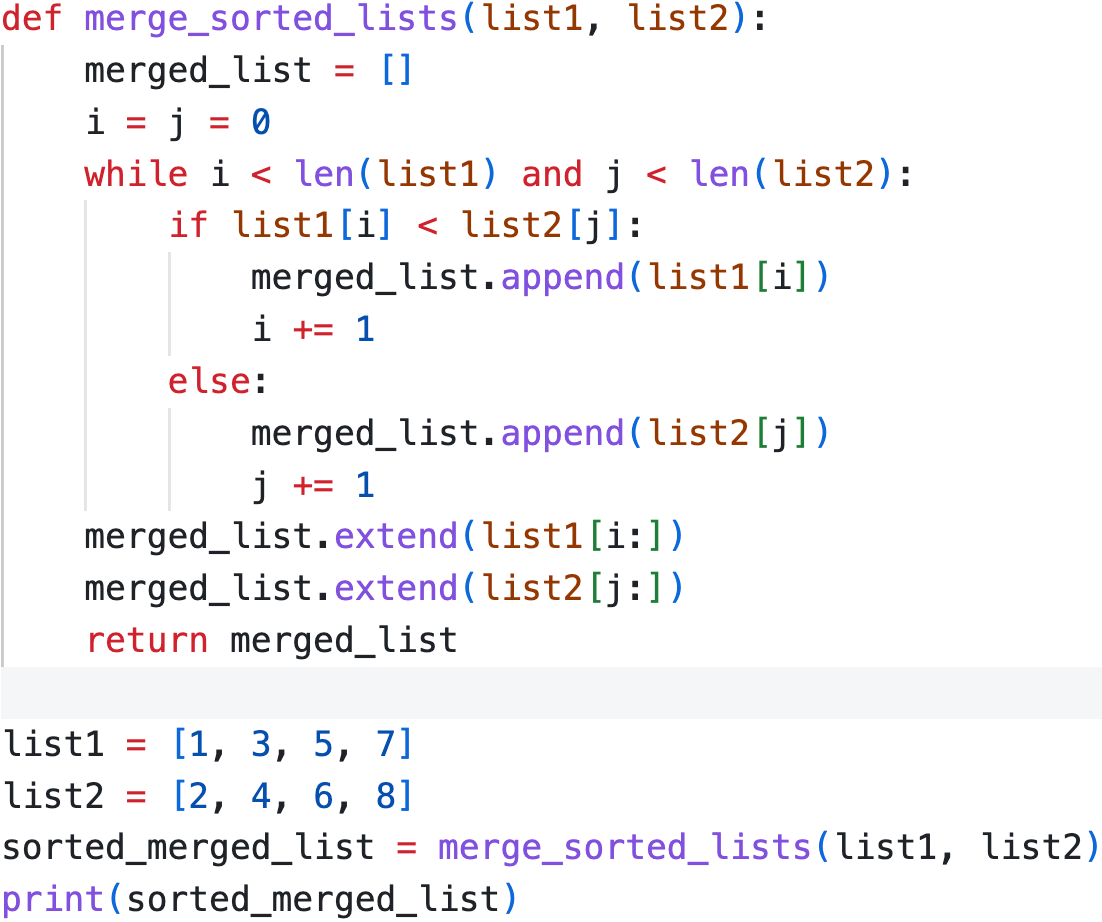
Output:- Screenshot 2025-03-09 at 10.30.07 PM

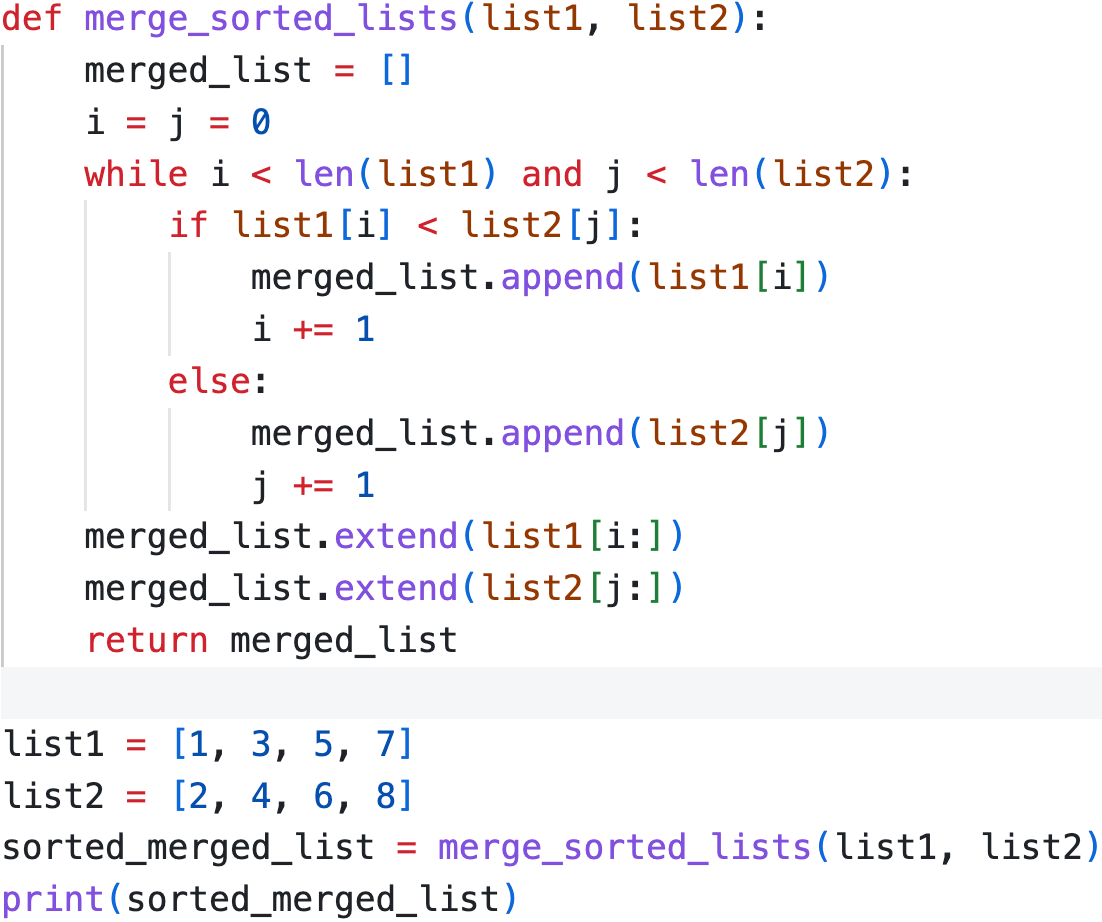
1. What Without using the built-in sorted() function, write a Python program to merge two pre-sorted lists into a single sorted list using the logic of Merge Sort. Example:

Input: [1, 3, 5, 7] and [2, 4, 6, 8]

Output: [1, 2, 3, 4, 5, 6, 7, 8].

**Ans:-**





Output: - Screenshot 2025-03-09 at 10.36.34 PM