3-Non-Recursive Structure:

Pseudo Code:

FUNCTION MaxProductOfThree(arr):

IF length of arr < 3:

RETURN "Not enough numbers to form a product"

SORT arr in ascending order

# Case 1: Largest three numbers

max\_product1 arr[0] \* arr[1] \* arr[-1]

RETURN max(max\_product1, max\_product2)

Iteration Analysis:

Selection Process:

Instead of exploring all possible subsets of three elements, sorting enables direct selection of numbers that could form the maximum product. Sorting ensures that:

1. The largest three numbers are positioned at the end of the array.

2. The smallest two numbers (potentially negative) are positioned at the beginning.

Thus, only these numbers need to be examined, eliminating unnecessary comparisons.

Number of Computations:

The function avoids \*exponential growth\* seen in exhaustive subset selection.

- Sorting requires O(N log N) operations.

- Selecting elements and computing their product requires O(1) operations.

- Comparing the two cases also requires O(1) operations.

Thus, the total number of computations follows

O(N log N) complexity.

Time Complexity:

- Sorting dominates the execution time at O(N log N).

- All further operations (selection, comparison) occur in O(1).

-Overall time complexity: O(N log N).

Space Complexity:

- Sorting may be performed \*in-place, requiring O(1) extra space.

- No additional memory allocation beyond basic variable storage.

-Overall space complexity: O(1).

Conclusion:

Time Complexity:\* O(N log N)

Space Complexity:\* O(1)