Time Complexity Comparison: Selection Sort vs Merge Sort

# 1. Purpose of Both Codes

Both programs:  
1. Take n integers as input.  
2. Sort the array in descending order.  
3. Compute the maximum product of any three numbers in the array by evaluating:  
 - Product of the top three largest numbers.  
 - Product of the largest number and the two smallest (possibly negative) numbers.

# 2. First Code – Selection Sort

## Algorithm Used: Selection Sort (descending)

## Time Complexity Analysis:

Selection Sort:  
- Outer loop runs n-1 times, inner loop runs up to n-1 times.  
- Best Case: O(n²)  
- Average Case: O(n²)  
- Worst Case: O(n²)  
Sorting time is always quadratic, regardless of the initial array order.  
  
Other Operations:  
- Input reading: O(n)  
- Max product calculation: O(1)

Overall Time Complexity: O(n²) (dominated by sorting)

# 3. Second Code – Merge Sort

## Algorithm Used: Merge Sort (descending)

## Time Complexity Analysis:

Merge Sort:  
- Divide step takes constant time.  
- Merge step takes linear time.  
- The array is divided recursively into log(n) levels.  
- Best Case: O(n log n)  
- Average Case: O(n log n)  
- Worst Case: O(n log n)  
  
Other Operations:  
- Input reading: O(n)  
- Max product calculation: O(1)

Overall Time Complexity: O(n log n) (dominated by sorting)

# 4. Comparison Table

|  |  |  |
| --- | --- | --- |
| Feature | Selection Sort (Code 1) | Merge Sort (Code 2) |
| Sorting Algorithm | Selection Sort | Merge Sort |
| Time Complexity | O(n²) | O(n log n) |
| Space Complexity | O(1) (in-place) | O(n) (due to extra arrays) |
| Stability | Not stable | Stable |
| Suitability for Large n | Not suitable | Efficient and scalable |
| Implementation Simplicity | Simple | More complex (recursion, dynamic memory) |

# 5. Conclusion

Use Selection Sort only for small arrays or when memory is very limited and performance is not a concern.  
Use Merge Sort for larger arrays where performance matters—its time complexity is much better and scales efficiently.