Max Product of Three (Recursive)

Pseudo code

```
Function mergeSort(array, left, right)
    If left < right:
        mid = (left + right) / 2
        mergeSort(array, left, mid)
        mergeSort(array, mid + 1, right)
        merge(array, left, mid, right)

Function merge(array, left, mid, right)
    Create temp arrays L and R for two halves
    Fill L with array[left to mid]
    Fill R with array[mid+1 to right]

    While both L and R have elements:
        If L[i] >= R[j]:
            array[k] = L[i]; i++
        Else:
            array[k] = R[j]; j++
        k++

    Copy remaining elements from L (if any)
    Copy remaining elements from R (if any)

Main:
    Read n
    While n < 3:
        Ask for valid n again
    Read n elements into array
    Call mergeSort(array, 0, n-1)

    product1 = array[0] * array[1] * array[2]
    product2 = array[0] * array[n-1] * array[n-2]
    Output max(product1, product2)
```

Implementation

```c
#include <stdio.h>
#include <stdlib.h>

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int *L = (int *)malloc(n1 * sizeof(int));
    int *R = (int *)malloc(n2 * sizeof(int));

    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] >= R[j]) {  // Sort in descending order
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }

    free(L);
    free(R);
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);  //T(n/2)
        mergeSort(arr, mid + 1, right); //T(n/2)

        merge(arr, left, mid, right); //T(n)
    }
}
```

```c

int main() {
    int n;
    do {
        printf("Enter the number of elements in the array (at least three numbers): ");
        scanf("%d", &n);
    } while (n < 3);

    int *arr = (int *)malloc(n * sizeof(int));
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    mergeSort(arr, 0, n - 1);

    int product1 = arr[0] * arr[1] * arr[2];
    int product2 = arr[0] * arr[n - 1] * arr[n - 2];

    int maxProduct = (product1 > product2) ? product1 : product2;
    printf("The maximum product of three numbers is: %d\n", maxProduct);

    free(arr);
    return 0;
}
```

Analysis & Complexity

1. do { … } while (n<3); loop

- **Check n<3: O(1)**
- **Body (printf, scanf): O(1)**
- **Number of iterations: some constant k (depends on how many times the user enters an invalid n, but independent of n)**
- **⇒ Total: O(1)**

---

2. Reading the array

```
for (int i = 0; i < n; i++) {
  scanf("%d", &arr[i]);
}
```

- **Each iteration: a single scanf + loop overhead ⇒ O(1)**
- **Iterations: n**
- **⇒ Total: Θ(n)**

---

3. mergeSort(arr, 0, n-1)

```
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}
```

- **The function divides the array into halves recursively.**
- **Each level of recursion does O(n) work through the merge function.**
- **The depth of recursion tree is log n.**
- **⇒ Total: Θ(n log n)**

---

4. merge(arr, left, mid, right)

```
while (i < n1 && j < n2) {
    if (L[i] >= R[j]) {
        arr[k] = L[i]; i++;
    } else {
arr[k] = R[j]; j++;
    }
    k++;
}
while (i < n1) arr[k++] = L[i++];
while (j < n2) arr[k++] = R[j++];
```

- **Merges two sorted subarrays in linear time: O(n)**
- **Happens at each level of recursion**
- **log n levels total**
- **⇒ Total across all calls: Θ(n log n)**

---

5. Computing the two products & printing

```
int product1 = arr[0] arr[1] arr[2];
int product2 = arr[0] arr[n-1] arr[n-2];
int maxProduct = (product1 > product2) ? product1 : product2;
printf(...);
```

- **Fixed number of multiplications and comparisons: O(1)**
- **⇒ Total: O(1)**

---

6. Memory cleanup

```
free(arr);
```

- **Single call to free dynamic memory**
- **⇒ Total: O(1)**

Phase	Cost
1. do-while input-validation	**O(1)**
2. Reading n elements	**Θ(n)**
3. Merge Sort	**Θ(n log n)**
4. Merging	**Θ(n log n)**
5. Final product/comparison/print	**O(1)**
6. Memory cleanup	**O(1)**
Overall	**Θ(n log n)**