# 4- Sort Colors

| Name | ID |
| --- | --- |
| عبدالرحمن علاء الدين صلاح هلال | 20210516 |
| كريم مصطفى عبدالرحمن ابراهيم عيد | 202000673 |
| احمد صابر محمد عبد الرازق | 20210062 |
| آسر احمد فهمى حسين | 20210147 |
| محمد شعبان سرور احمد | 202000768 |
| عبدالرحمن حمدي فوزي | 201900414 |

Team number : 123

# The pseudo code for insertion sort algorithm

- InsertionSort(A)
- 1. n ← length[A]
- 2. if  1 > n > 300
- 3. then exit()
- 4. for j ←  2 to n do
- 5. {
- 6.  key ←  A[j]
- 7.  if not (key ←  0 OR key ←  1 OR key ←  2)
- 8.  then exit()
- 9.  // Insert A[j] into the sorted sequence A[1..j-1]
- 10.  i ←  j - 1
- 11.  while i > 0 AND A[i] > key
- 12.  A[i+1] ←  A[i]
- 13.  i ←  i - 1
- 14.  A[i+1] ←  key
- 15.  }

# Analysis of pseudo code for insertion sort algorithm

- InsertionSort(A) ----------> Input size (A)
- 1. n ← length[A]
- 2. if 1 > n > 300
- 3. then exit()
- 4. for j ← 2 to n do ----------> n
- 5. {
- 6. key ← A[j]
- 7. if not (key ← 0 OR key ← 1 OR key ← 2)
- 8. then exit()
- 9. // Insert A[j] into the sorted sequence A[1..j-1]
- 10. i ← j - 1
- 11. while i > 0 AND A[i] > key ----------> n
- 12. A[i+1] ← A[i]
- 13. i ← i - 1 ----------> Basic Operation
- 14. A[i+1] ← key
- 15. }

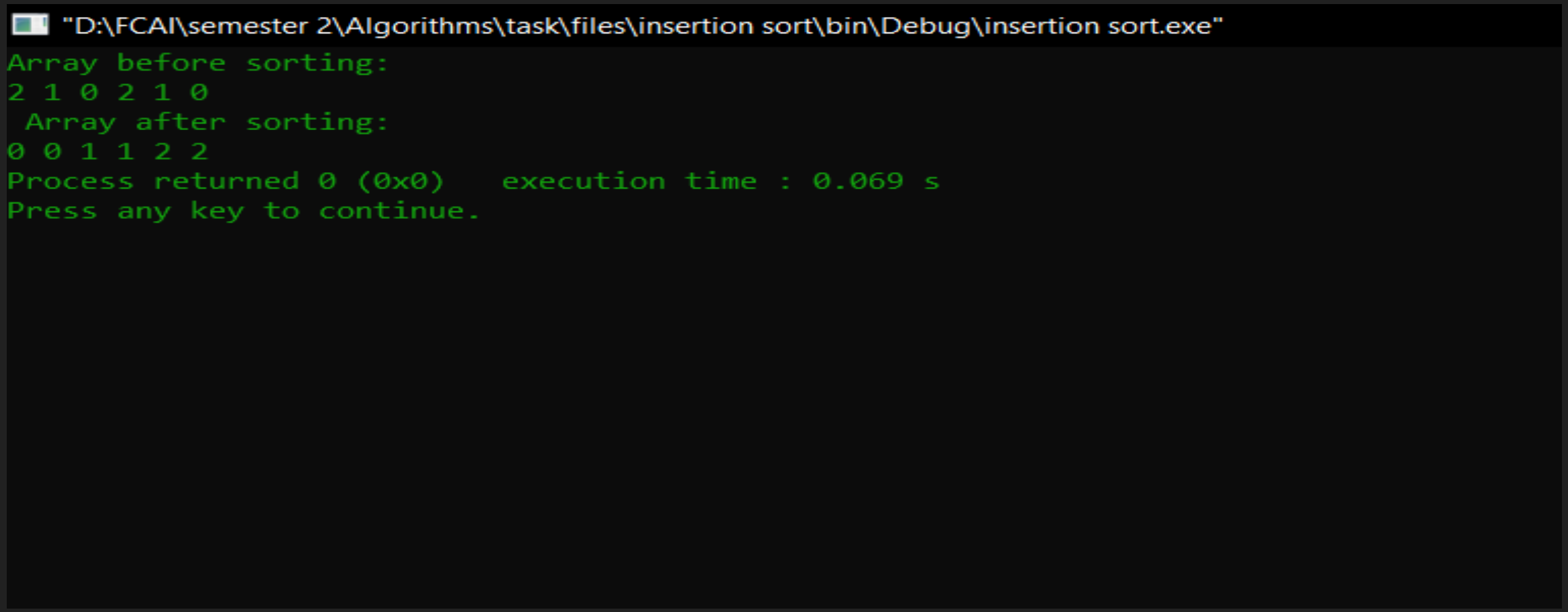
# Cont. Analysis and time complexity for insertion sort algorithm

$$\sum_{j=2}^{n} n = n \sum_{j=2}^{n} 1$$

$$n \times (n-2+1) = n^2-2n+n = n^2-n$$

Time complexity for insertion sort → $C(n) \in \Theta(n^2)$

# Screen shot for output



# The pseudo code for merge sort algorithm

```
Algorithm sort_colors(nums):
   merge_sort(nums, 0, len(nums) - 1)
merge_sort(nums, start, end):
            if start < end:
                    mid = (start + end) // 2
                    merge_sort(nums, start, mid)
                    merge_sort(nums, mid+1, end)

                    merge(nums, start, mid, end)

merge(nums, start, mid, end):
        // Create temporary arrays for the left and right subarrays
        left = nums[start:mid+1]
        right = nums[mid+1:end+1]

   // Merge the left and right subarrays while sorting them in-place
        i = j = 0
        k = start
```

```
while i < len(left) and j < len(right):
    if left[i] <= right[j]:
        nums[k] = left[i]
        i += 1
    else:
        nums[k] = right[j]
        j += 1
    k += 1

while i < len(left):
    nums[k] = left[i]
    i += 1
    k += 1

while j < len(right):
    nums[k] = right[j]
    j += 1
    k += 1
```

# Analysis of pseudo code for merge sort algorithm

merge_sort(nums, start, mid) → T(n/2) Recursive
merge_sort(nums, mid+1, end) → T(n/2) Recursive
merge(nums, start, mid, end) θ(n) → non Recursive

# Cont. Analysis and time complexity for merge sort algorithm

$T(n) = 2T(n/2) + \theta(n)$

we will use Master method

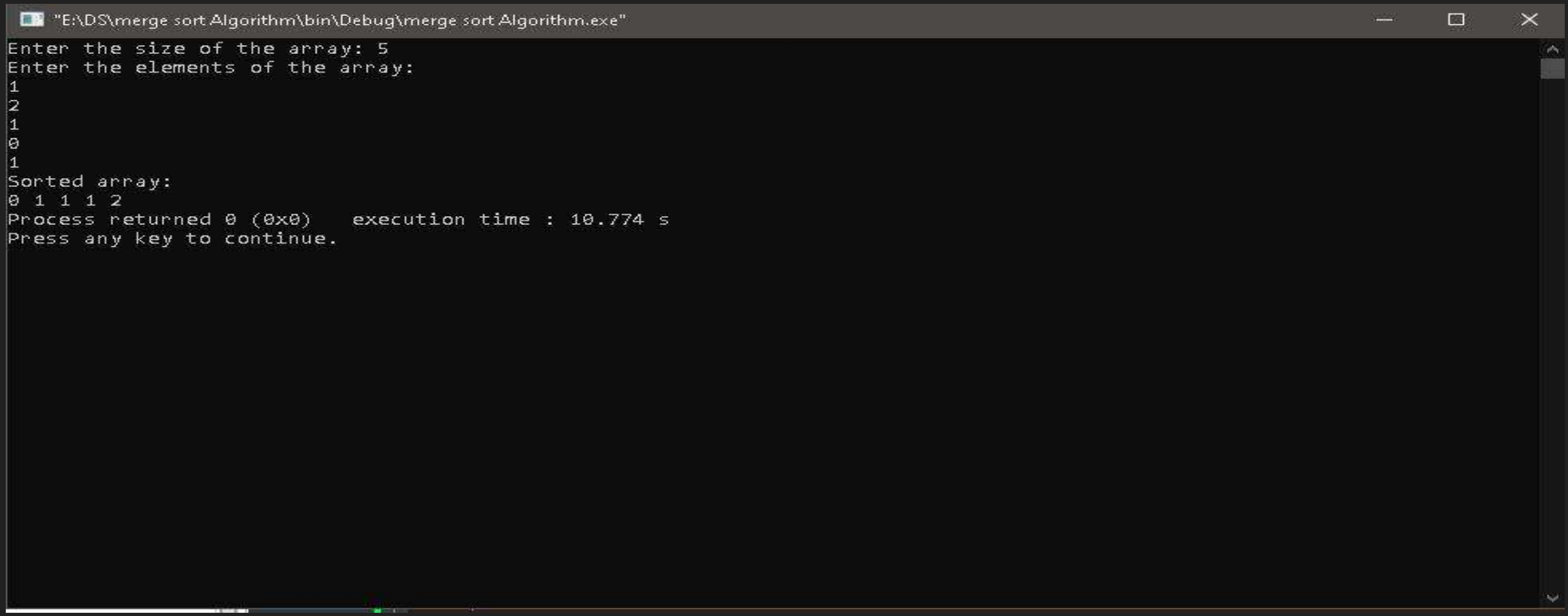
$a = 2$ , $b = 2$ , $f(n) = \theta(n)$

$n^{\log_2 2} = n = \theta(n)$

then $f(n) = n^{\log_2 2} = \theta(n)$ ⟶ case 2

$T(n) = \theta(n^{\log_2 2} \lg n) = \theta(n * \lg n)$

# Screen shot for output



# Complexity compare table

| Algorithms | Average | Worst | Space |
|---|---|---|---|
| Insert | O(n^2) | O(n^2) | **O(1)** |
| Merge | O(nlogn) | O(nlogn) | **O(n)** |

**Complexity Merge is better than Insert**