Data Structure Lab 1

محمد ايهاب مبروك حمادي	20011524
نور خالد محمد	22011319
محمد مصطفى السيد	22011170
عبدالرحمن محمد احمد احمد نصر	22010887
بدر السيد جلال	22010664

Bubble Sort:

- **Time Complexity:**
  - **Average Case: O(n^2)** - check every two adjacent elements, and Swap them if they are in the wrong order, require more than one loop to get the array sorted
  - **Best Case: O(n)** - This occurs when the array is already sorted. (one loop to check if the array is sorted).
  - **Worst Case: O(n^2)** - Similar to the average case, the worst-case scenario occurs when the array is in reverse order.

Space Complexity:

The space complexity of the algorithm is O(1), as there is no need for extra space.

Insertion Sort:

- **Time Complexity:**
  - **Average Case: O(n^2)** - Put each element in the right position in respect to the sorted part of the array by shifting the rest of the array.
  - **Best Case: O(n)** - This occurs when the array is already sorted. (one loop to check if the array is sorted)
  - **Worst Case: O(n^2)** - Similar to the average case, the worst-case scenario occurs when the array is in reverse order.

Space Complexity:

The space complexity of the algorithm is O(1), as there is no need for extra space.

Merge sort:

- **Time Complexity:**
  - **Average Case: O(n log n)** - Merge sort divides the array into halves recursively, and then merges them back together. The merge step takes O(n) time where n is the total number of elements in the subarrays, and the recursive splitting takes O(log n) time.
  - **Best Case: O(n log n)** - No difference in the time complexity if the array is already sorted.
  - **Worst Case: O(n log n)** - No difference in the time complexity if the array is in reverse order.

- **Space Complexity:**
    - The space complexity of the algorithm is O(n) due to the additional space required for the temporary arrays (subArray1 and subArray2) in the merge method. These arrays grow to be as large as the input array in the final step, resulting in O(n) space complexity.

Quick Sort:

- **Time Complexity:**
  - **Average Case: O(n log n)** - Selects a pivot (random element), partitions the array around it, and recursively sorts the partitions.
  - **Best Case: O(n log n)** - If the pivot is always chosen to partition the array into two halves.
  - **Worst Case: O(n^2)** - when the smallest or largest element is always chosen as the pivot. (instead of log n in partition, it becomes n)

- **Space Complexity:**
  - **Average Case: O(n log n)**
  - **Best Case: O(n log n)**
  - **Worst Case: O(n^2)** - due to unbalanced tree

Radix Sort:

- **Time Complexity:**
  - **Best Case: O(nk)** - where n is the number of elements in the input array and k is the number of digits in the largest number. In the best case, all the numbers have the same number of digits, and the algorithm performs linear time operations for each digit.
  - **Average Case: O(nk)** - Radix sort performs O(n) operations for each digit, and since there are k digits in the largest number, the average time complexity is O(nk).
  - **Worst Case: O(nk)** - The worst-case scenario occurs when all the numbers have different digits. In this case, the algorithm performs O(n) operations for each digit, resulting in a time complexity of O(nk).

- **Space Complexity:**
  - The space complexity of the radix sort algorithm is O(n + k), where n is the number of elements in the input array and k is the range of the input (the maximum number - minimum number + 1). This space is used for the counter array in counting sort and the temporary array temp.

  - Overall, the radix sort algorithm has a time complexity of O(nk) and a space complexity of O(n + k). It is often more efficient than comparison-based sorting algorithms for large datasets when k is not very large compared to n.

The Comparison:

- Compare the algorithoms performance on arrays with starting size **= 3**, and increases it by a factor of **3** up to **20000**.

```
Array size =  3
Bubble Sort - Time taken: 2650 ns
Insertion Sort - Time taken: 1975 ns
Merge Sort - Time taken: 7000 ns
Quick Sort - Time taken: 8266 ns
Radix Sort - Time taken: 6699 ns
Array size =  9
Bubble Sort - Time taken: 5316 ns
Insertion Sort - Time taken: 3800 ns
Merge Sort - Time taken: 11091 ns
Quick Sort - Time taken: 11116 ns
Radix Sort - Time taken: 5524 ns
Array size =  27
Bubble Sort - Time taken: 29133 ns
Insertion Sort - Time taken: 16891 ns
Merge Sort - Time taken: 30374 ns
Quick Sort - Time taken: 37316 ns
Radix Sort - Time taken: 8283 ns
Array size =  81
Bubble Sort - Time taken: 161366 ns
Insertion Sort - Time taken: 72583 ns
Merge Sort - Time taken: 66291 ns
Quick Sort - Time taken: 80958 ns
Radix Sort - Time taken: 18408 ns
Array size =  243
Bubble Sort - Time taken: 615833 ns
Insertion Sort - Time taken: 422174 ns
Merge Sort - Time taken: 121358 ns
Quick Sort - Time taken: 107858 ns
Radix Sort - Time taken: 44366 ns
```

```
Array size =  729
Bubble Sort - Time taken: 1234391 ns
Insertion Sort - Time taken: 824716 ns
Merge Sort - Time taken: 412341 ns
Quick Sort - Time taken: 384708 ns
Radix Sort - Time taken: 127933 ns
Array size =  2187
Bubble Sort - Time taken: 6070766 ns
Insertion Sort - Time taken: 7351725 ns
Merge Sort - Time taken: 2782241 ns
Quick Sort - Time taken: 3792233 ns
Radix Sort - Time taken: 414116 ns
Array size =  6561
Bubble Sort - Time taken: 59860158 ns
Insertion Sort - Time taken: 34536533 ns
Merge Sort - Time taken: 53828158 ns
Quick Sort - Time taken: 32310425 ns
Radix Sort - Time taken: 512483 ns
Array size =  19683
Bubble Sort - Time taken: 914256758 ns
Insertion Sort - Time taken: 462122417 ns
Merge Sort - Time taken: 374479350 ns
Quick Sort - Time taken: 425274150 ns
Radix Sort - Time taken: 568433 ns
```