**Question 1: Implement Quicksort:  
(i) First Element as Pivot:**

* The problem is solved by making use of arrays
* Program is written on C with these basic functions.
* (i) **partition(int arr[],int beg,int end):** takes the start and end positions and the array of numbers. It selects the first element as the pivot, and places all the smaller on the lft and the larger numbers on the right, this way the pivot reaches it’s correct position and returns the index of it’s position.
* (ii) **quicksort(int arr[],int p, int r):** this function takes input as the arr input, divides into halves and recursively follows this partitioning done on the index value returned by the partition function to sort the array.
* (ix) **main()**: main driver function that helps the user make use of all the functionalities provided by the program. It takes input from the file into the array and calls quicksort function to sort the array.
* The input is checked against any exception, by using if and else to check for expected input.  
  If unexpected input is received, The program returns Invalid Input and exits.

**(ii) Random Element as Pivot:**

* The problem is solved by making use of arrays
* Program is written on C with these basic functions.
* (i) **partition(int arr[],int beg,int end):** takes the start and end positions and the array of numbers. It selects some element as the pivot, and places all the smaller on the lft and the larger numbers on the right, this way the pivot reaches it’s correct position and returns the index of it’s position.
* (ii) **piv\_create(int arr[], int p, int r):** Picks up a random element from my array to be used as the pivot element.
* (ii) **quicksort(int arr[],int p, int r):** this function takes input as the arr input, divides into halves and recursively follows this partitioning done on the index value returned by the partition function to sort the array.
* (ix) **main()**: main driver function that helps the user make use of all the functionalities provided by the program. It takes input from the file into the array and calls quicksort function to sort the array.
* The input is checked against any exception, by using if and else to check for expected input.  
  If unexpected input is received, The program returns Invalid Input and exits.

**(iii) Median of {first, middle and last element} as a pivot:**

* The problem is solved by making use of arrays
* Program is written on C with these basic functions.
* (i) **partition(int arr[],int beg,int end):** takes the start and end positions and the array of numbers. It selects some element as the pivot, and places all the smaller on the lft and the larger numbers on the right, this way the pivot reaches it’s correct position and returns the index of it’s position.
* (ii) **piv\_create(int arr[], int p, int r):** Calculates the median and creates a pivot and returns the pivot
* (ii) **quicksort(int arr[],int p, int r):** this function takes input as the arr input, divides into halves and recursively follows this partitioning done on the index value returned by the partition function to sort the array.
* (ix) **main()**: main driver function that helps the user make use of all the functionalities provided by the program. It takes input from the file into the array and calls quicksort function to sort the array.
* The input is checked against any exception, by using if and else to check for expected input.  
  If unexpected input is received, The program returns Invalid Input and exits.

**(iv) Median of {n/4th element, middle element, 3n/4th element} as:**

* The problem is solved by making use of arrays
* Program is written on C with these basic functions.
* (i) **partition(int arr[],int beg,int end):** takes the start and end positions and the array of numbers. It selects some element as the pivot, and places all the smaller on the lft and the larger numbers on the right, this way the pivot reaches it’s correct position and returns the index of it’s position.
* (ii) **piv\_create(int arr[], int p, int r):** Calculates the median and creates a pivot and returns the pivot
* (ii) **quicksort(int arr[],int p, int r):** this function takes input as the arr input, divides into halves and recursively follows this partitioning done on the index value returned by the partition function to sort the array.
* (ix) **main()**: main driver function that helps the user make use of all the functionalities provided by the program. It takes input from the file into the array and calls quicksort function to sort the array.
* The input is checked against any exception, by using if and else to check for expected input.  
  If unexpected input is received, The program returns Invalid Input and exits.

**Merge Sort:**

* The problem is solved by making use of arrays
* Program is written on C with these basic functions.
* (i) **merge(int a[],int p,int q,int r)):** this function is used to merge two subarrays. It takes two arrays as left and right, where it stores the entire array in two fragments. Then it sorts them and puts back the sorted order in the original array.
* (ii) **mergeSort(int a[],int p, int r):** Recursively calls itself and merges the sorted array partitions into a single sorted array.
* (ix) **main()**: main driver function that helps the user make use of all the functionalities provided by the program. It takes input from the file into the array and calls the mergesort function to sort the array.
* The input is checked against any exception, by using if and else to check for expected input.  
  If unexpected input is received, The program returns Invalid Input and exits.

**Output:**

**Quick Sort(n=1000)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Pivote\_type** | **Random** | **Shorted** | **Almost Sorted** |
| **First Element** | **0.138** | **0.125** | **0.125** |
| **Random Pivot** | **0.115** | **0.140** | **0.124** |
| **Median of first,middle and last** | **0.116** | **0.094** | **0.178** |
| **Median of n/4th,middle and3n/4th** | **0.099** | **0.17** | **0.169** |

**Quick Sort(n=10000)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Pivote\_type** | **Random** | **Shorted** | **Almost Sorted** |
| **First Element** | **1.311** | **1.404** | **1.389** |
| **Random Pivot** | **1.29** | **1.543** | **1.459** |
| **Median of first,middle and last** | **1.64** | **1.771** | **1.72** |
| **Median of n/4th,middle and3n/4th** | **1.953** | **1.965** | **2.034** |

**Quick Sort(n=100000)**

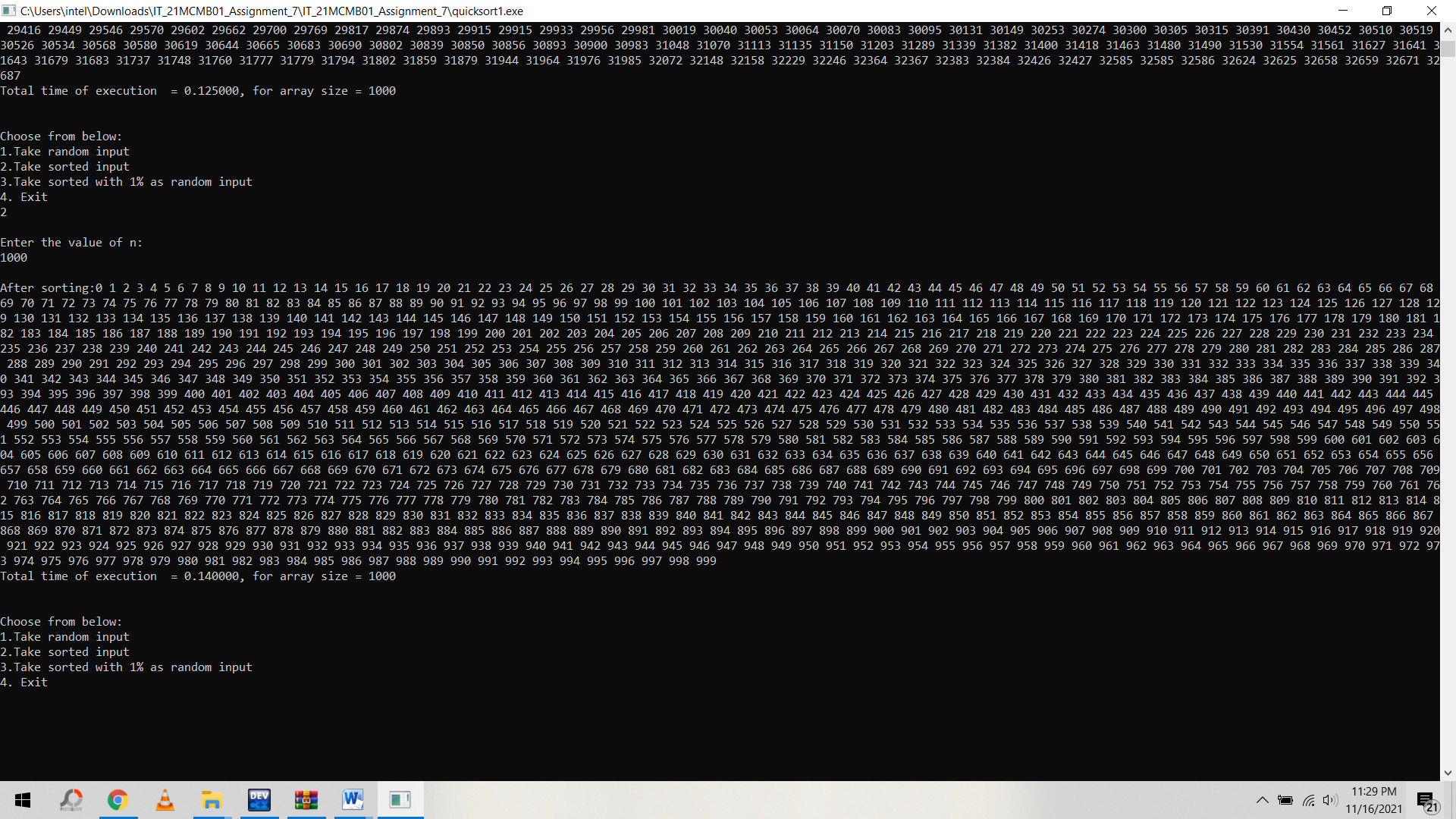
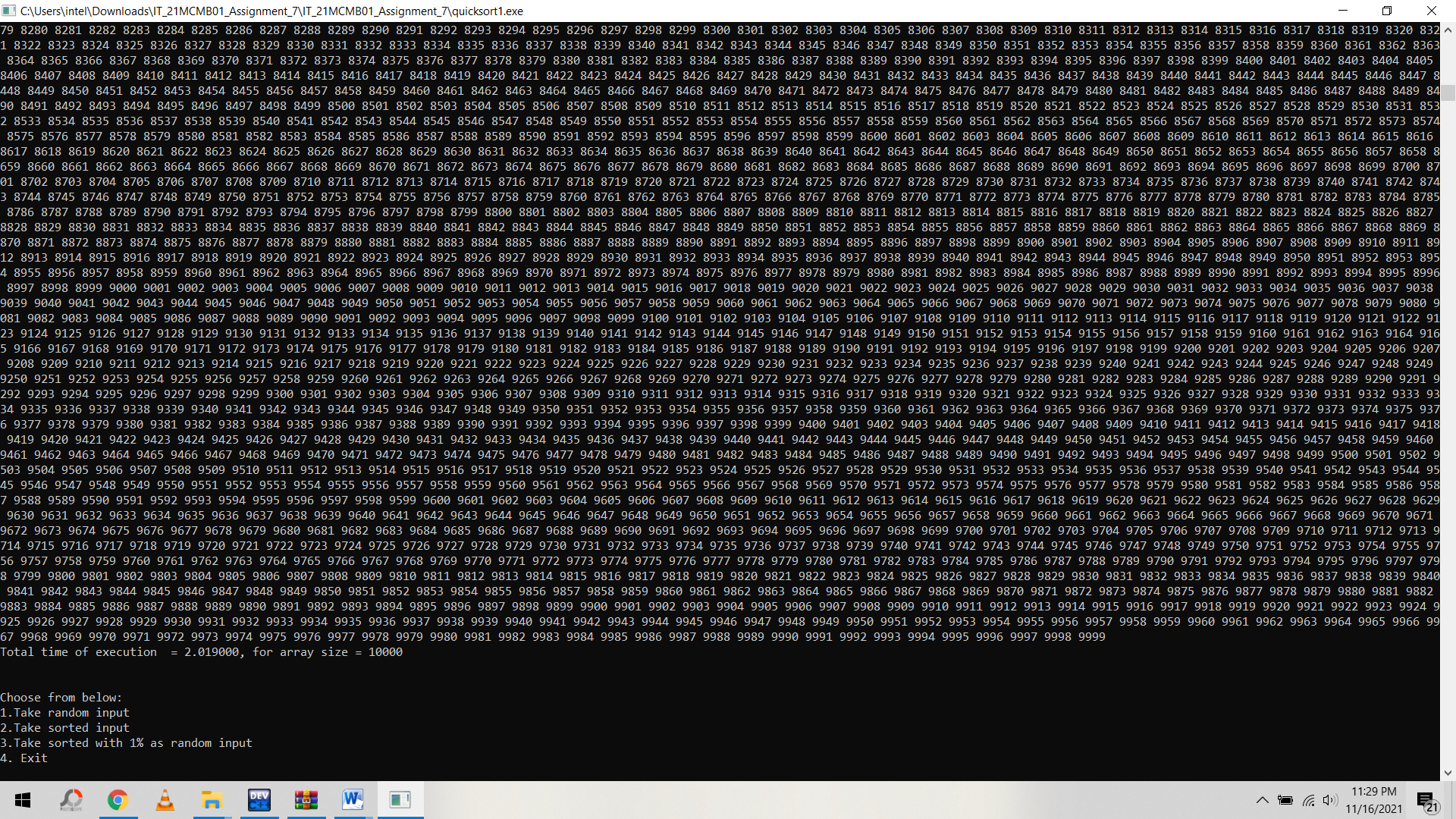
|  |  |  |  |
| --- | --- | --- | --- |
| **Pivote\_type** | **Random** | **Shorted** | **Almost Sorted** |
| **First Element** | **14.11** | **48.3** | **42.71** |
| **Random Pivot** | **13.04** | **11.995** | **10.017** |
| **Median of first,middle and last** | **16.44** | **39.21** | **35.43** |
| **Median of n/4th,middle and3n/4th** | **16.532** | **19.485** | **17.16** |

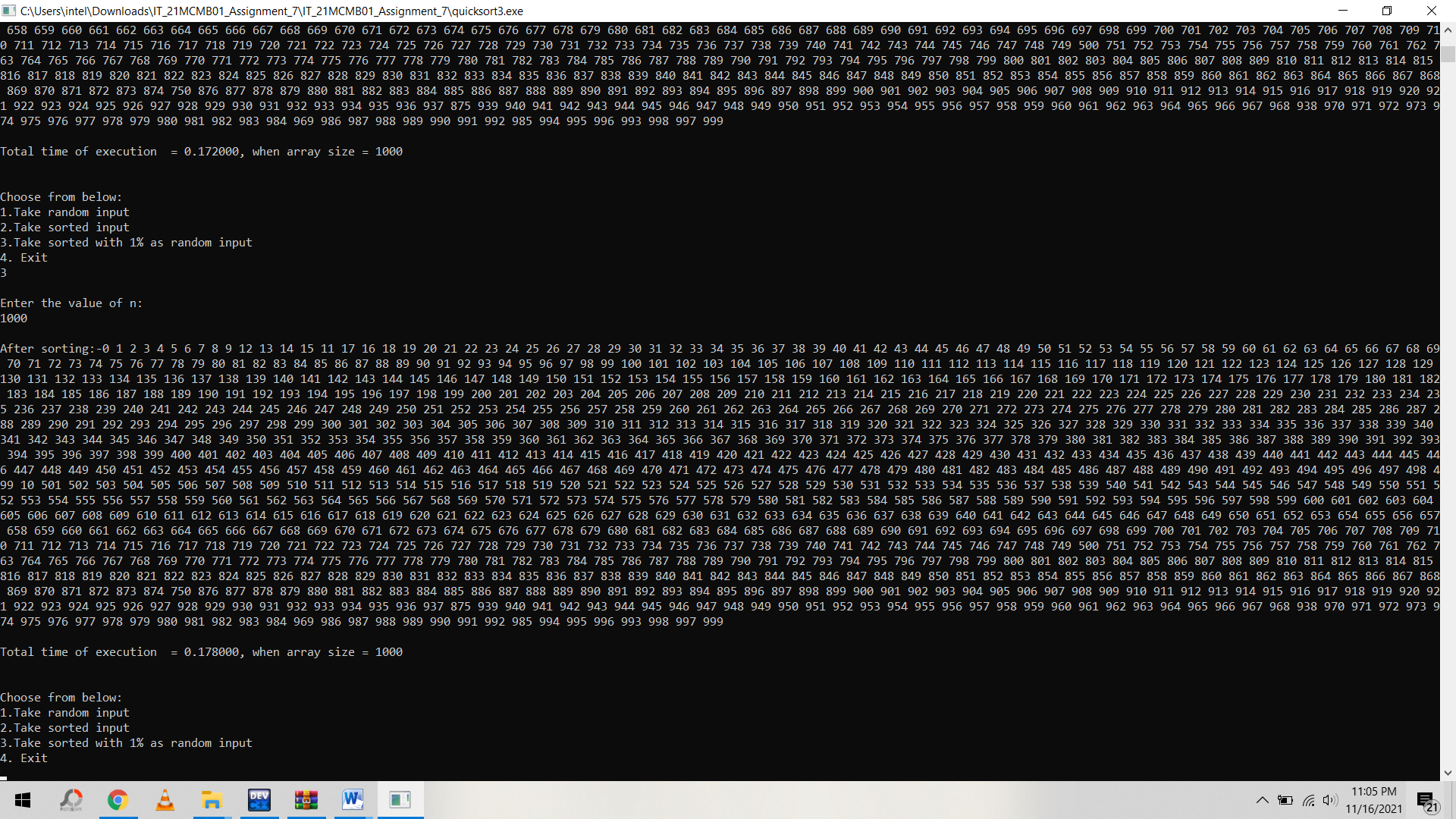
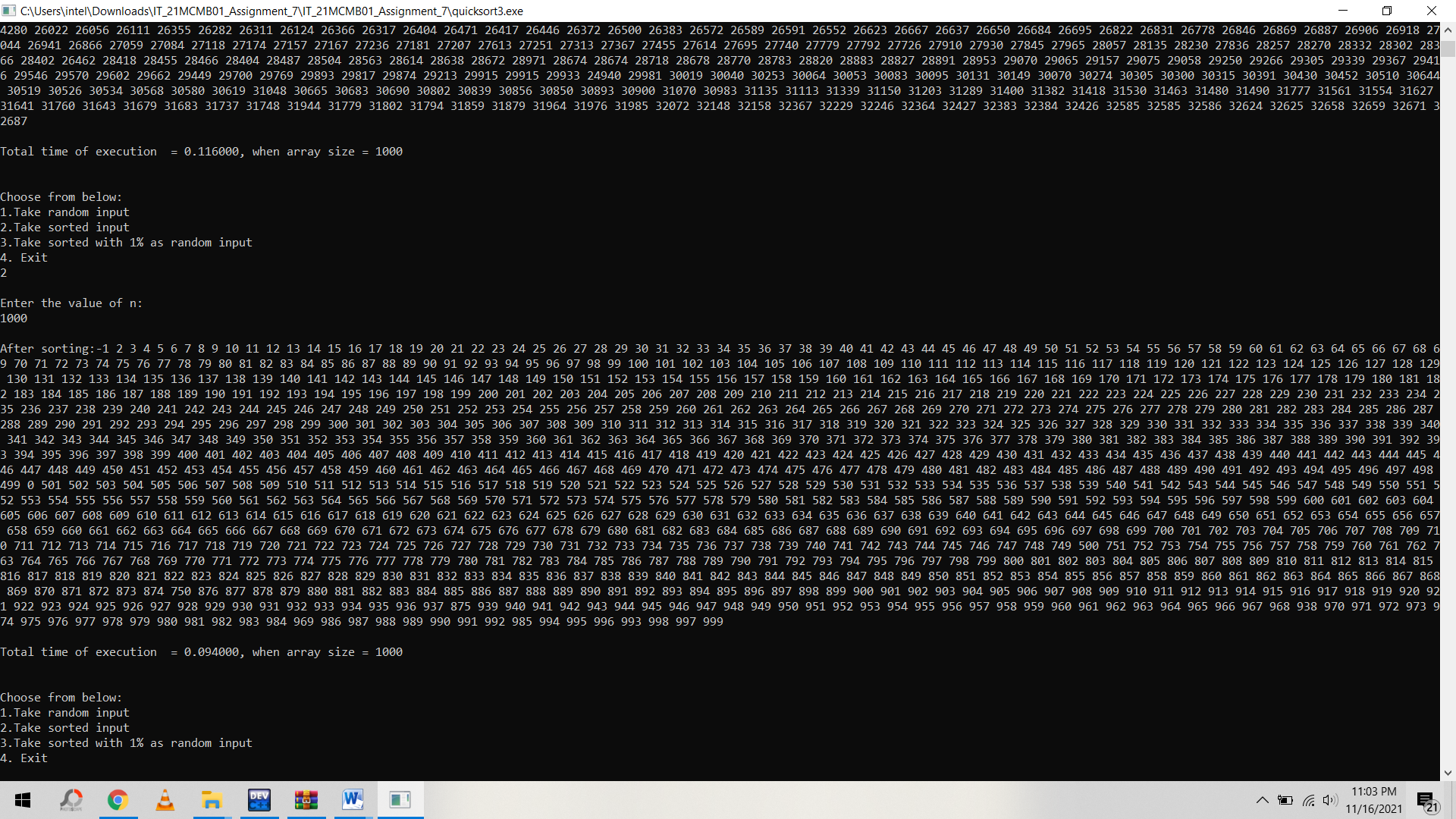
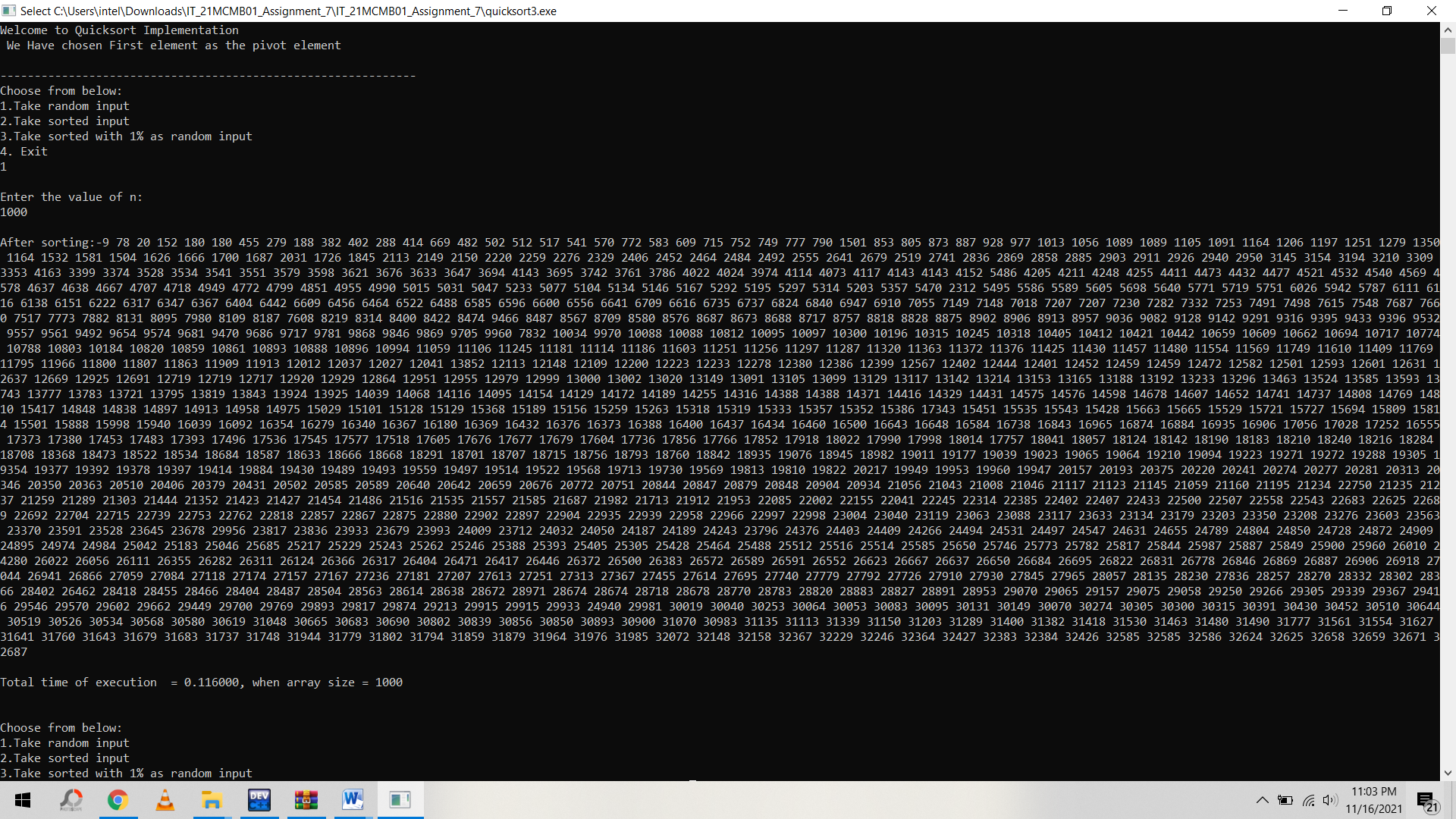
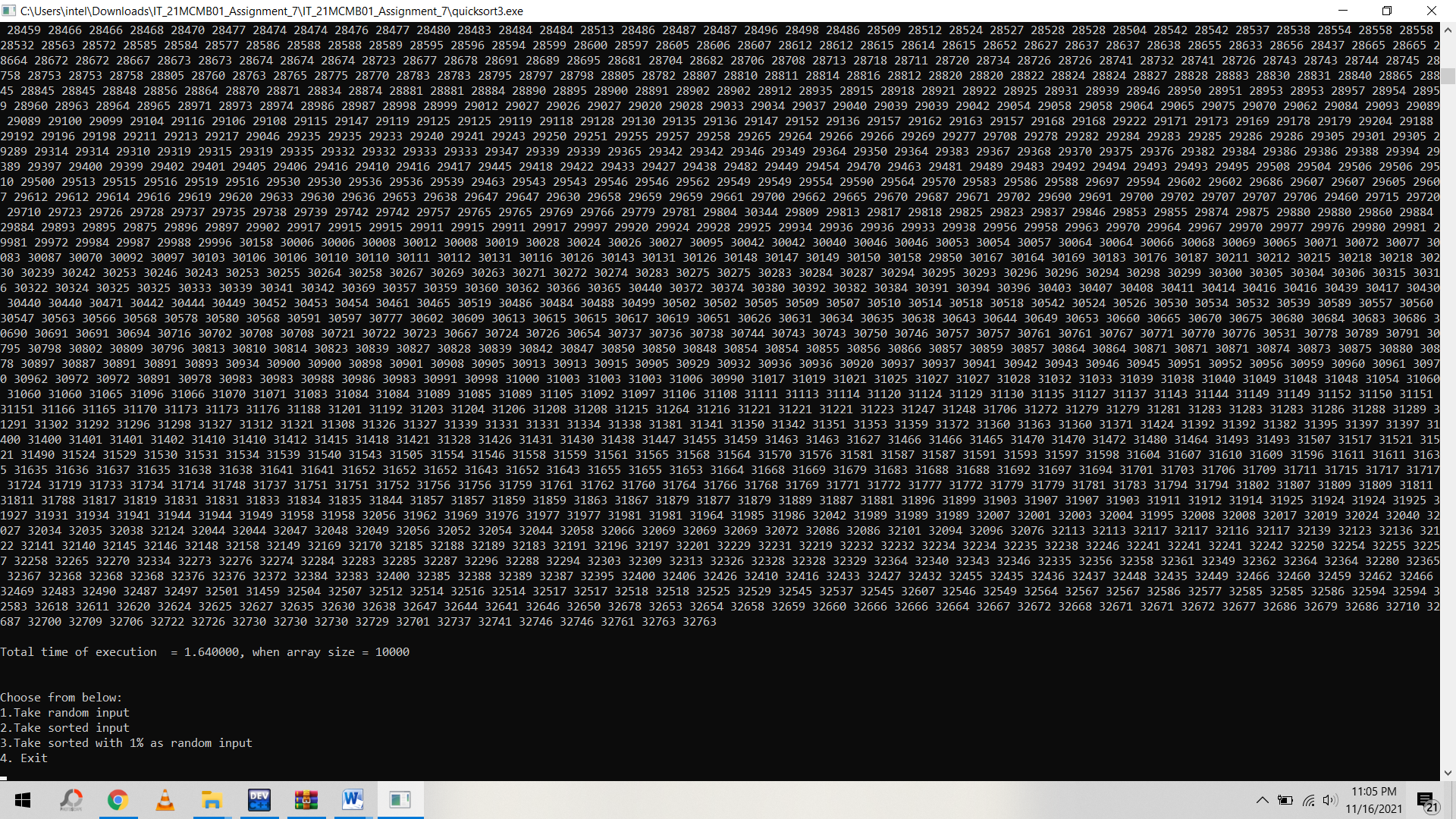
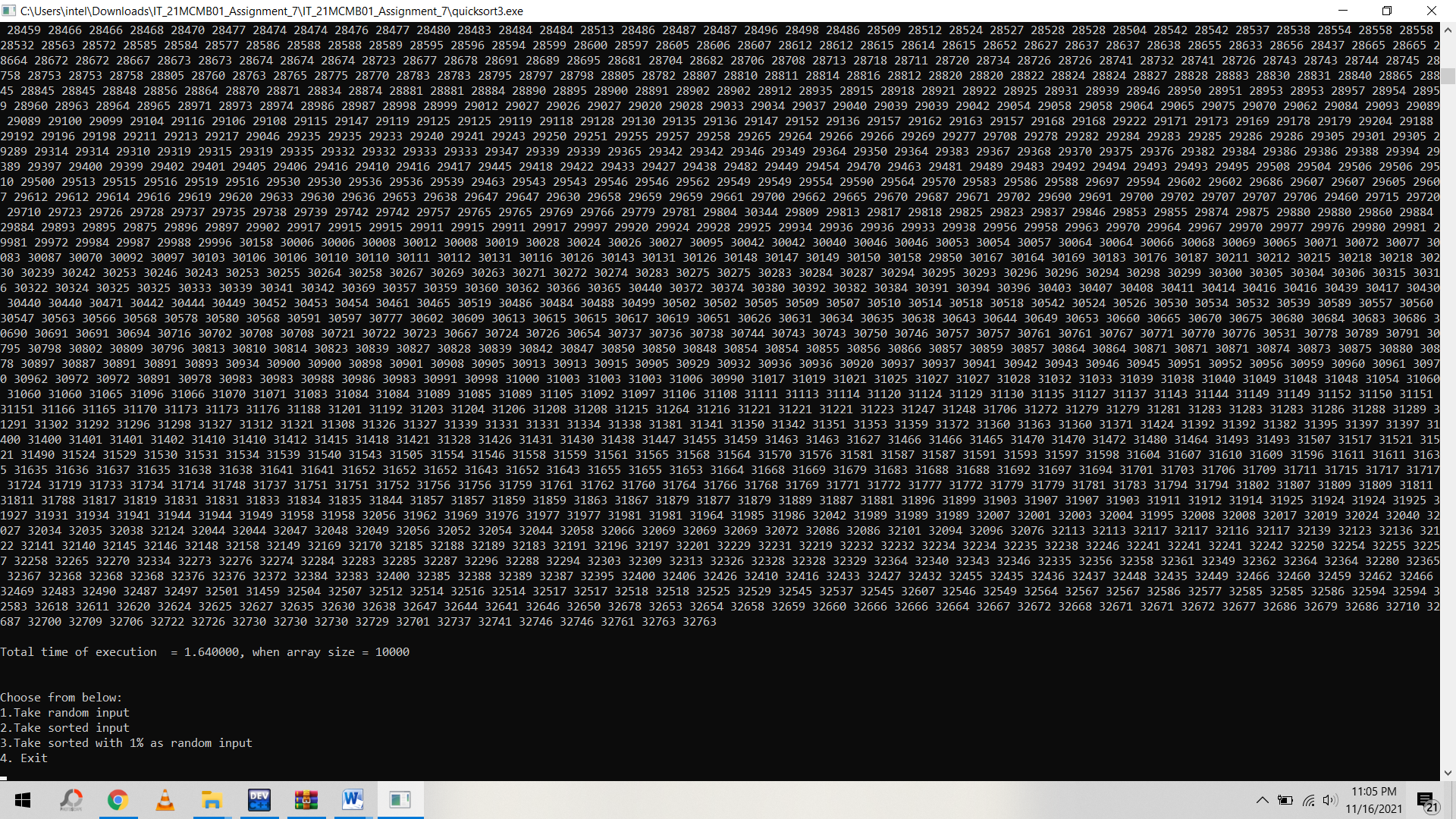
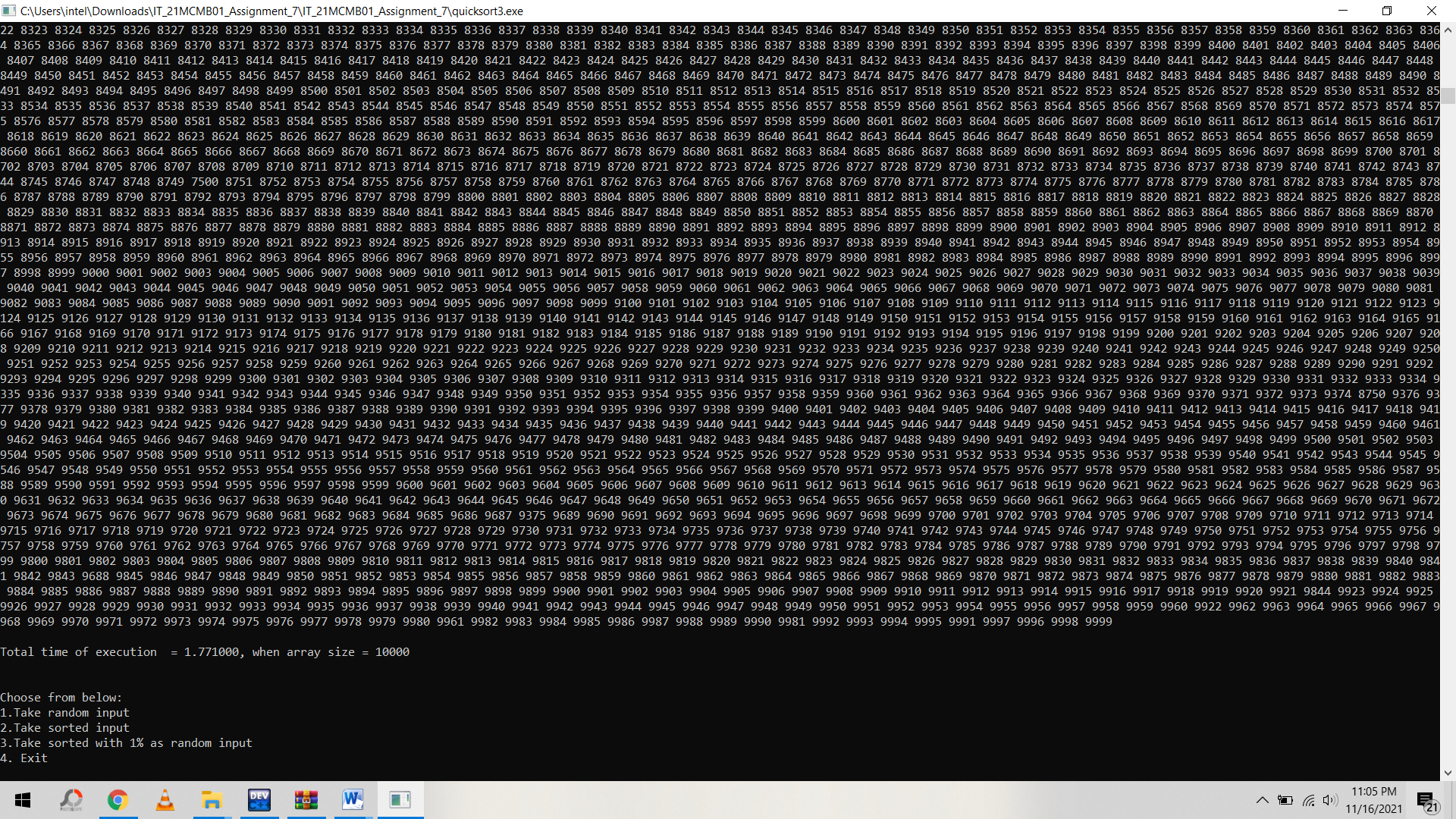
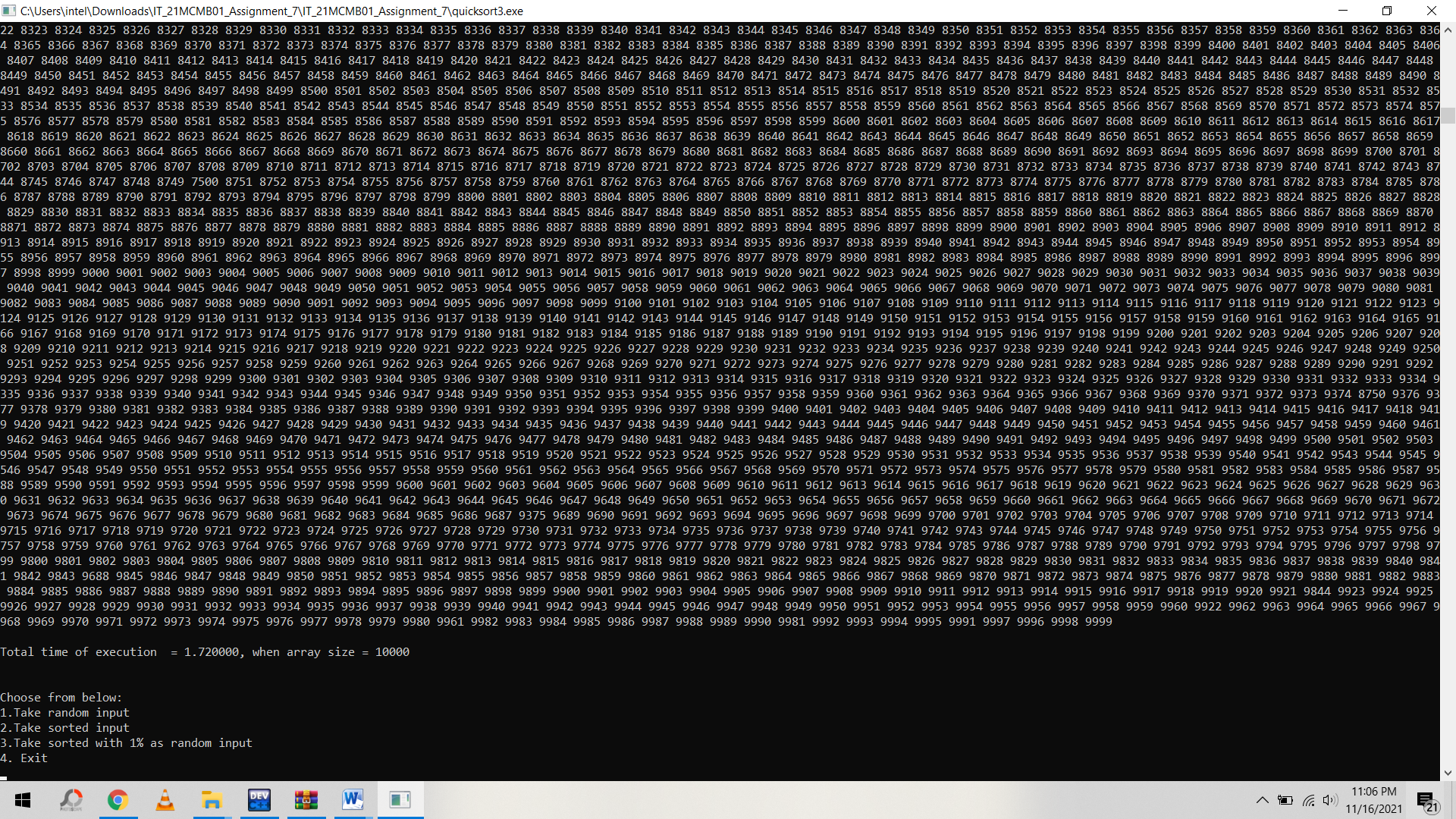
**Merge Sort**

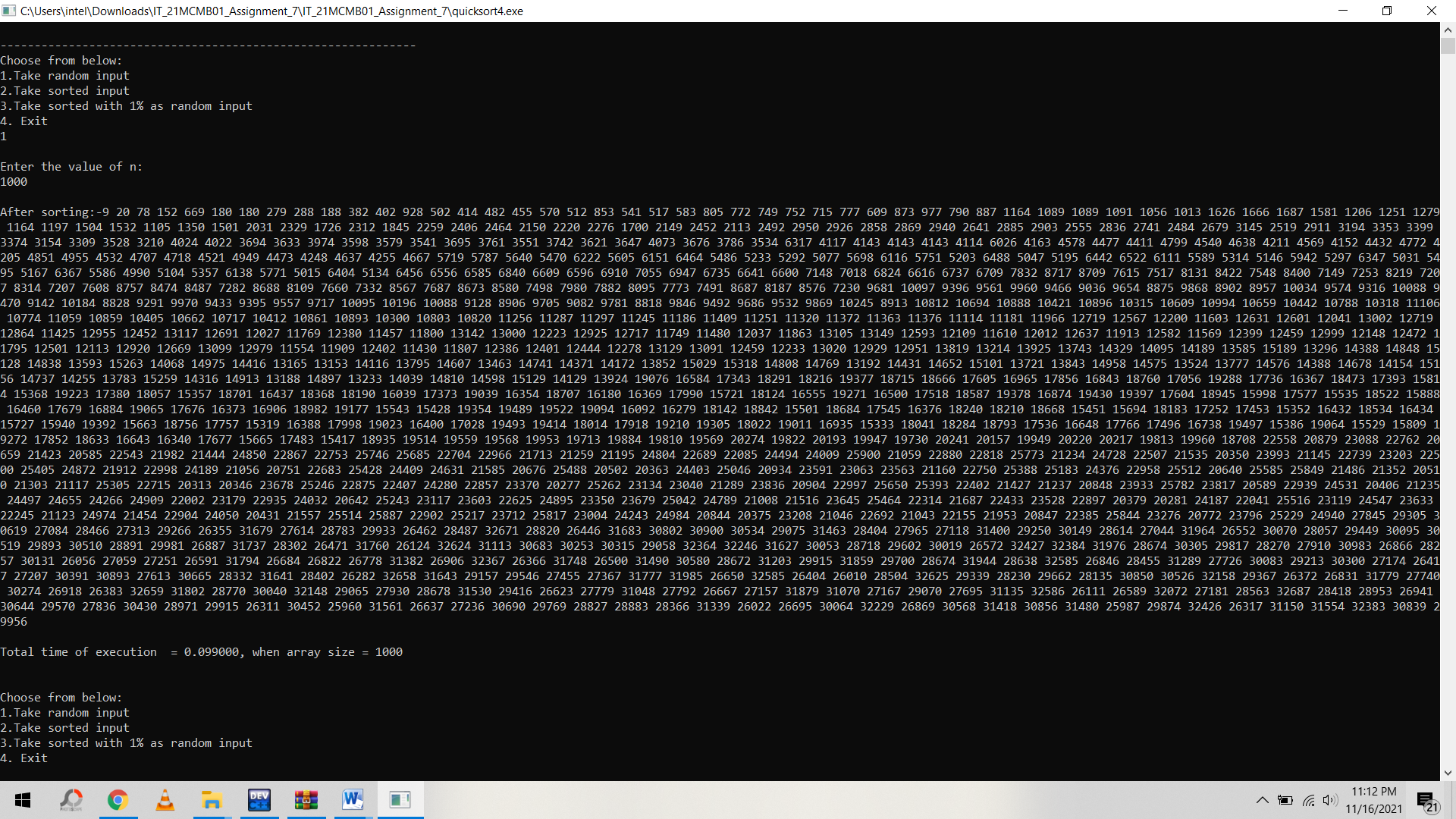
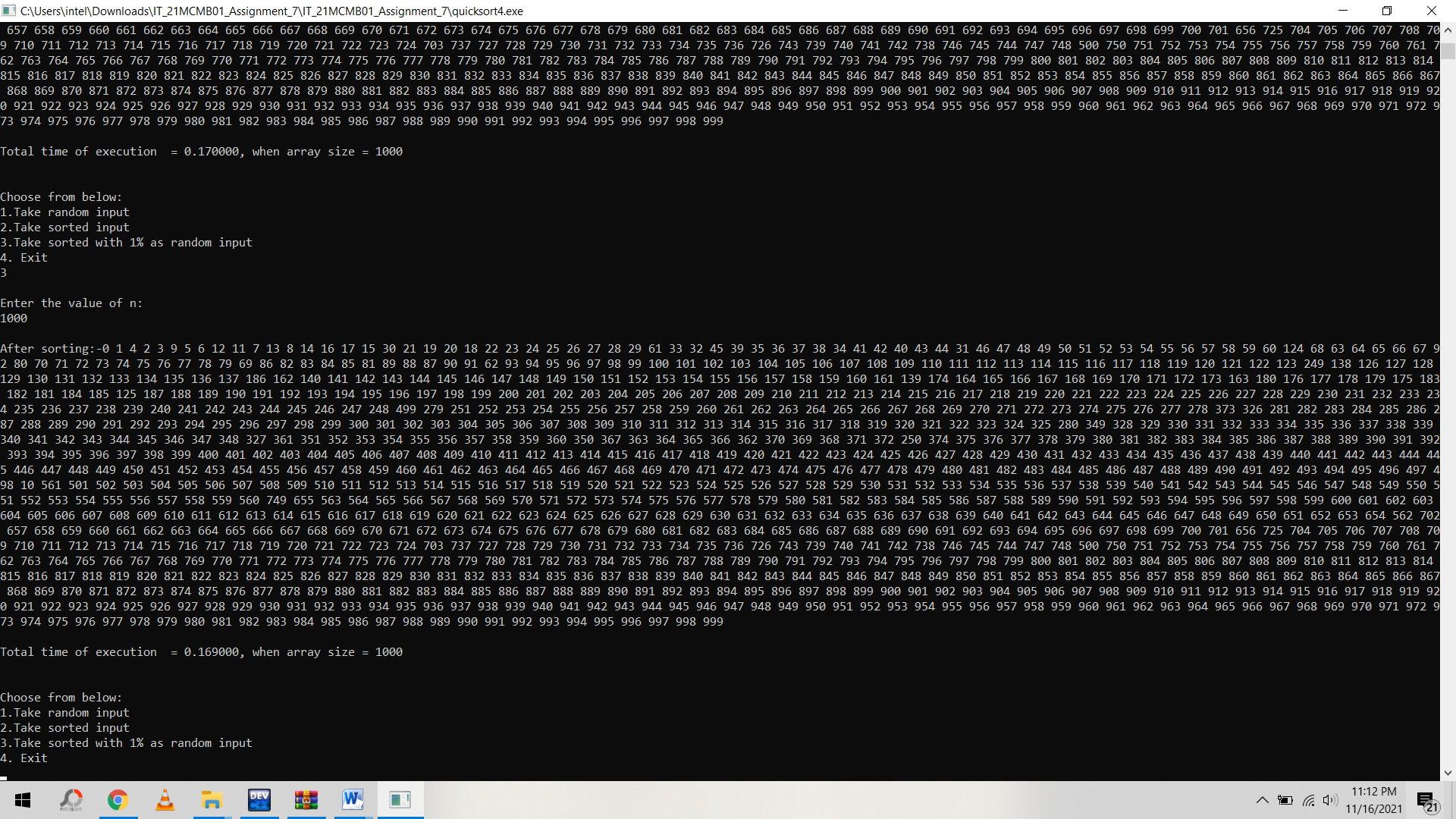
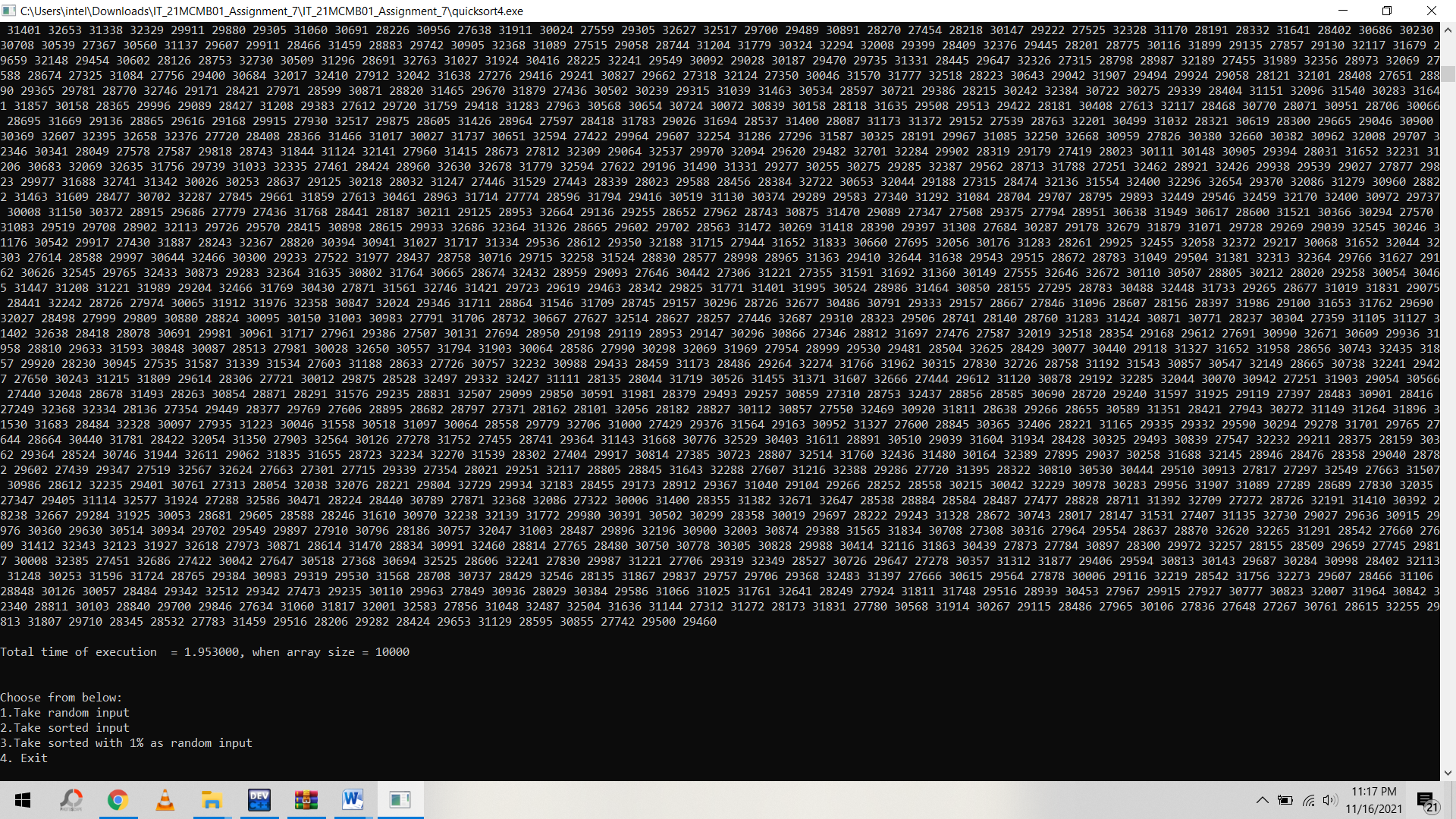
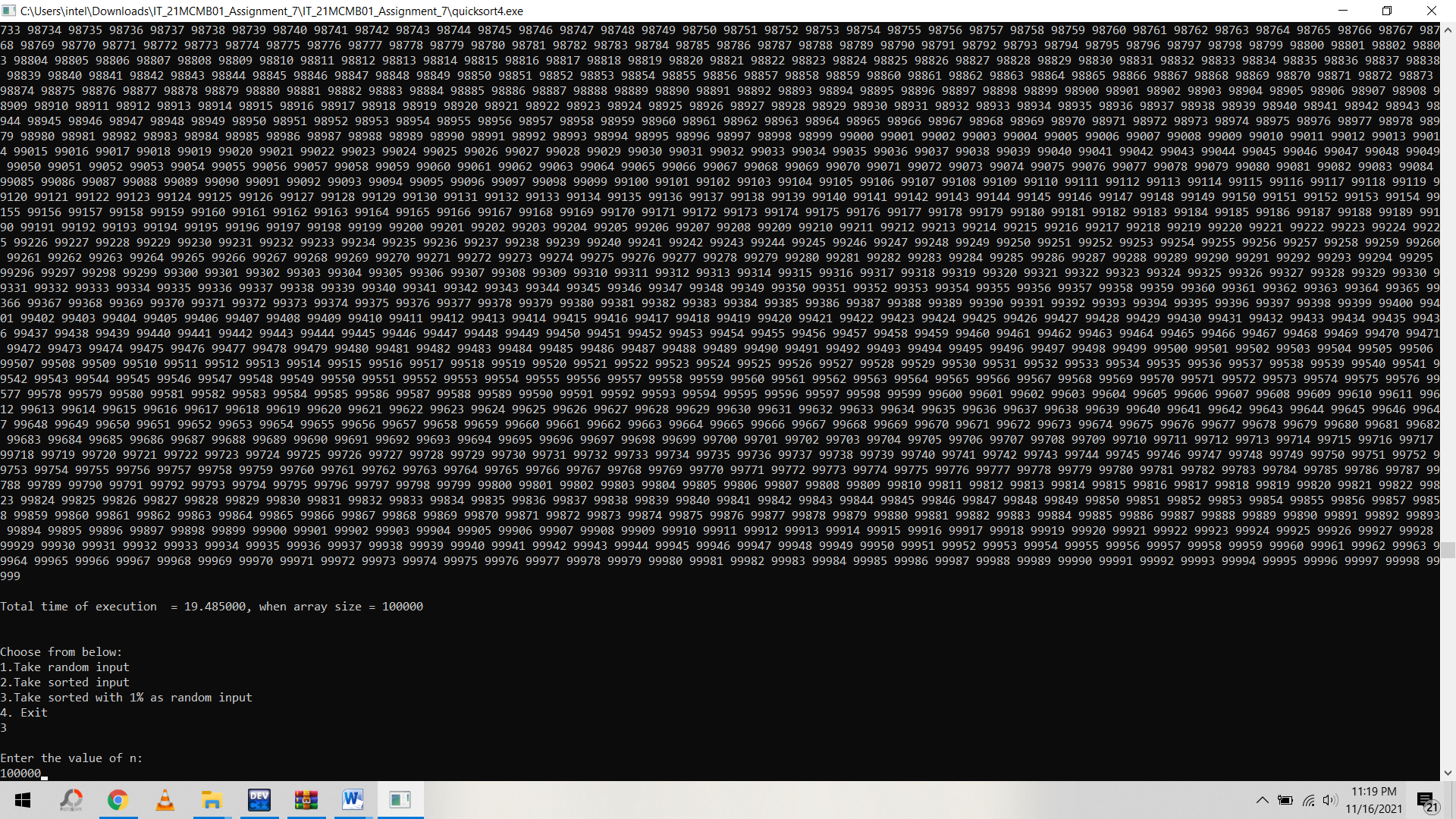
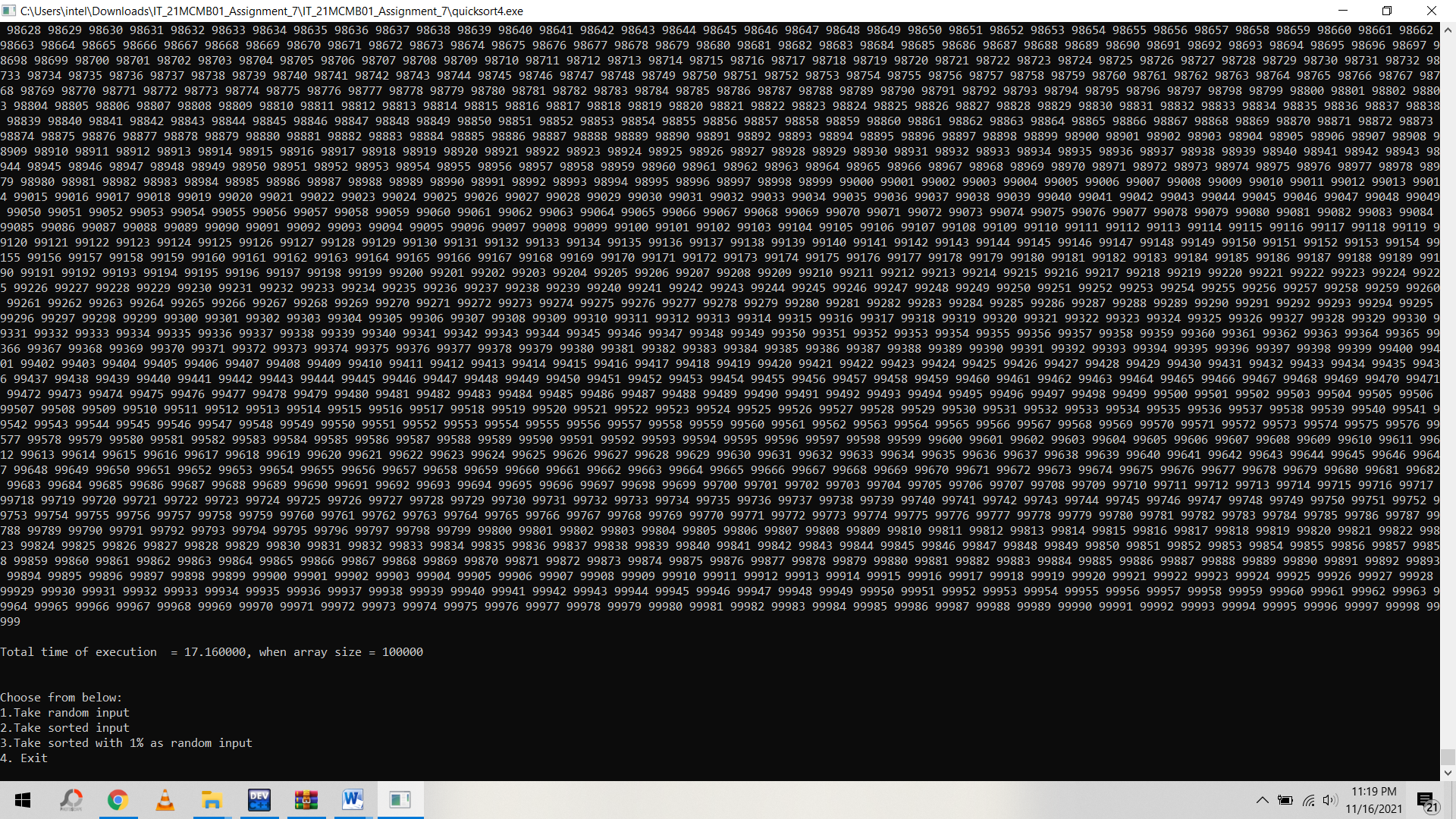
|  |  |  |  |
| --- | --- | --- | --- |
| **Pivote\_type** | **Random** | **Shorted** | **Almost Sorted** |
| **n = 1000** | **0.131** | **0.093** | **0.155** |
| **n = 10000** | **2.046** | **1.704** | **1.887** |
| **n = 100000** | **16.864** | **17.085** | **15.137** |

**What is your observation? Which short is faster and why?**

Mergesort does not use pivot and implements divide and conquer, thereby is faster than Quicksort.

**Execution Screenshots:** 

Merge Sort

