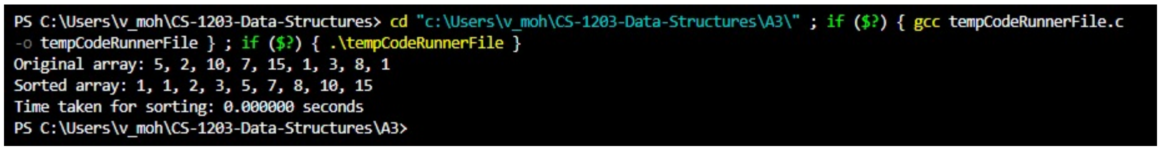
CS-1203 Data Structures

Assignment 3 **15/10/2023** **Khushi Mohta**

Question 2:

Insertion sort:



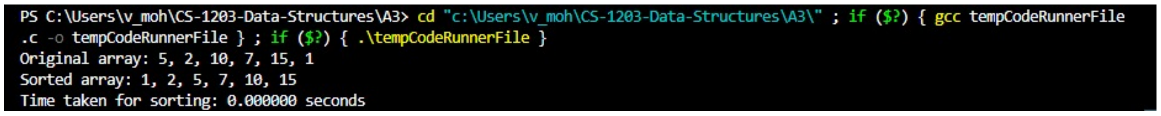
```
PS C:\Users\v_moh\CS-1203-Data-Structures> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile.c
-o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Original array: 5, 2, 10, 7, 15, 1, 3, 8, 1
Sorted array: 1, 1, 2, 3, 5, 7, 8, 10, 15
Time taken for sorting: 0.000000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3>
```

Time Complexity (Worst Case): $O(n^2)$
This is when the input array is sorted in the reverse (descending) order and the function has to make the maximum possible number of comparisons and swaps. It has to compare every element with all the previous elements and swaps them until it finds the correct position.

Time Complexity (Best Case): $O(n)$
This is when the array us already sorted correctly (ascending order) and the function has to make the minimum number of comparisons and does not have to swap elements

Space Complexity: $O(1)$
It uses constant space for temporary variables only

Bubble sort:



```
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Original array: 5, 2, 10, 7, 15, 1
Sorted array: 1, 2, 5, 7, 10, 15
Time taken for sorting: 0.000000 seconds
```

Time Complexity (Worst Case): $O(n^2)$
This is when the input array is in the reverse order and it has to make $n*(n-1)/2$ comparisons and swaps, leading to a time complexity of $O(n^2)$

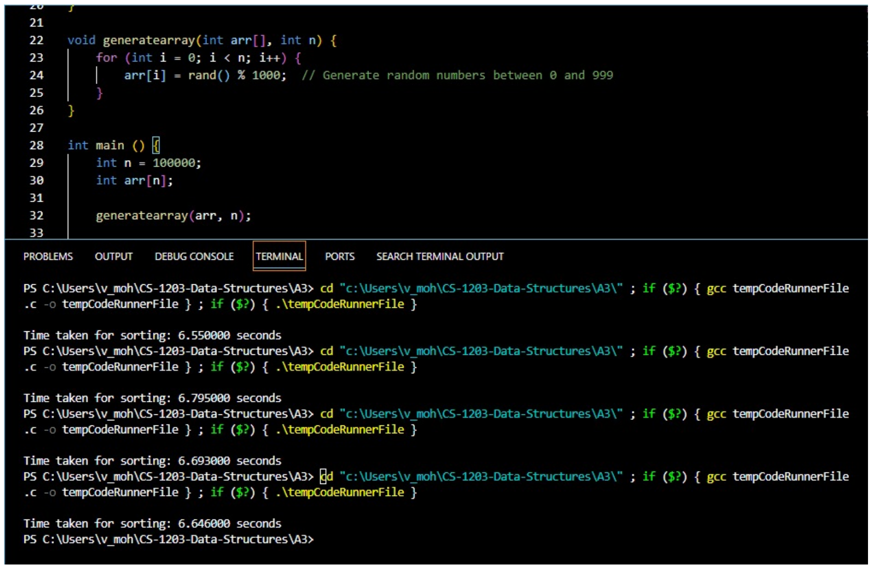
Time Complexity (Best Case): $O(n)$
This is when the array is already sorted and it has to make $n-1$ comparisons to determine that the array is sorted

Space Complexity: $O(1)$
It uses constant space for temporary variables only

Comparison:

Even though both insertion sort and bubble sort have the same worst case and best case time complexity, insertion sort performs better while dealing with large amounts of data. Insertion sort looks for the position of the nth (say) element in an array of $n-1$ sorted elements but bubble sort has to compare and swap an element with its adjacent element in each iteration

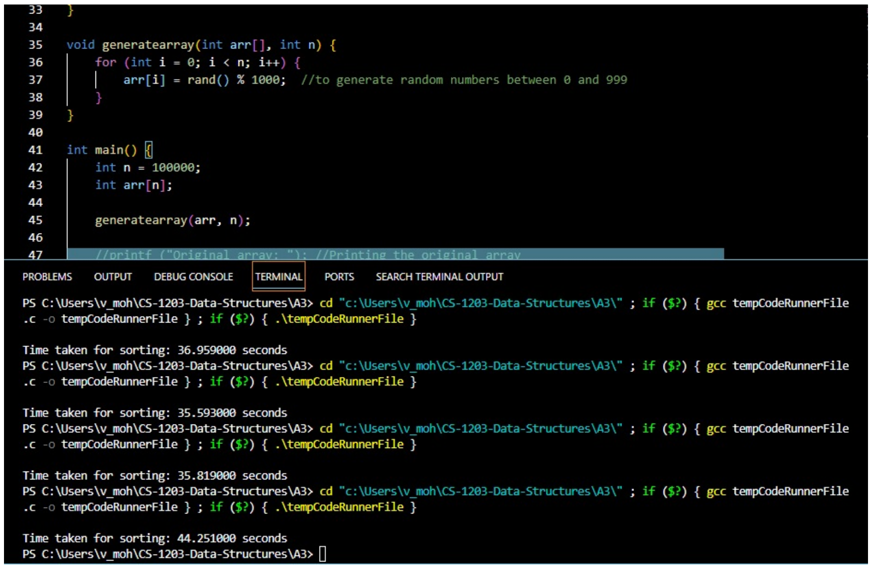
Experimental data for insertion sort with an array of 1,00,000 elements whose values ranged from 0 to 999:



```c
}

void generatearray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1000;  // Generate random numbers between 0 and 999
    }
}

int main () {
    int n = 100000;
    int arr[n];

    generatearray(arr, n);

```

```
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Time taken for sorting: 6.550000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Time taken for sorting: 6.795000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Time taken for sorting: 6.693000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Time taken for sorting: 6.646000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3>
```

Experimental data for insertion sort with an array of 1,00,000 elements whose values ranged from 0 to 999:

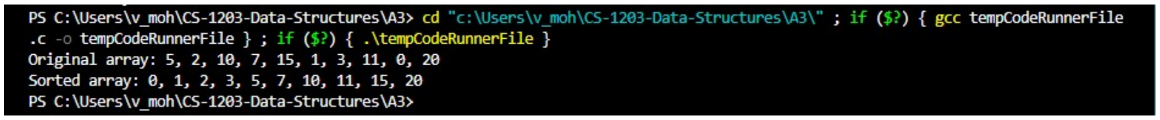


```c
}

void generatearray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1000;  //to generate random numbers between 0 and 999
    }
}

int main() {
    int n = 100000;
    int arr[n];

    generatearray(arr, n);

    //printf ("Original array: "); //Printing the original array
```

```
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Time taken for sorting: 36.959000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Time taken for sorting: 35.593000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Time taken for sorting: 35.819000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Time taken for sorting: 44.251000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3>
```

Question 3:

Merge sort:



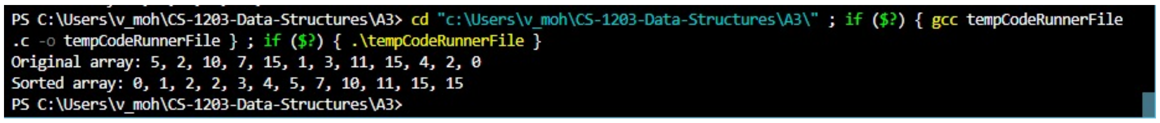
```
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Original array: 5, 2, 10, 7, 15, 1, 3, 11, 0, 20
Sorted array: 0, 1, 2, 3, 5, 7, 10, 11, 15, 20
PS C:\Users\v_moh\CS-1203-Data-Structures\A3>
```

Time Complexity (Worst Case): $O(nlogn)$
This algorithm divides an array into two halves and sorts each half recursively. The merge step to combine both halves takes $O(n)$ time in the worst case. The dividing step happens recursively in a balanced way, and the worst case time complexity is $O(nlogn)$

Time Complexity (Best Case): $O(nlogn)$
This is because merge sort follows the same divide-and-conquer approach, regardless of the input order

Space Complexity: $O(n)$
Requires additional space to store the temporary arrays, and that is directly proportional to the size of the input array (n)

Quick sort:



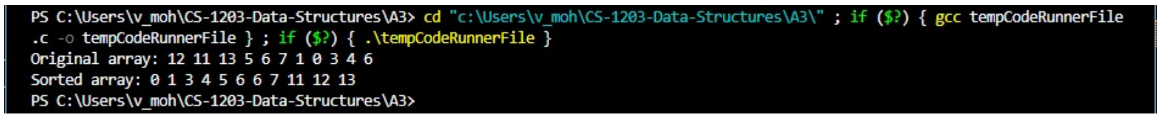
```
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Original array: 5, 2, 10, 7, 15, 1, 3, 11, 15, 4, 2, 0
Sorted array: 0, 1, 2, 2, 3, 4, 5, 7, 10, 11, 15, 15
PS C:\Users\v_moh\CS-1203-Data-Structures\A3>
```

Time Complexity (Worst Case): $O(n^2)$
This happens when the pivot element is selected as the largest or the smallest element. The partitioning does not evenly divide the array

Time Complexity (Best Case): $O(nlogn)$
This happens when the array is perfectly divided into two equal sub-arrays

Space Complexity: $O(logn)//$ This is determined by how deep the recursion stack needs to go. In the best-case scenario, it doesn't need many trays ($O(logn)$, which is very efficient)

Heap Sort:



```
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Original array: 12 11 13 5 6 7 1 0 3 4 6
Sorted array: 0 1 3 4 5 6 6 7 11 12 13
PS C:\Users\v_moh\CS-1203-Data-Structures\A3>
```

Time Complexity (Worst Case): $O(nlogn)$
This is because it builds a max heap and keeps extracting the maximum element to construct the sorted array. The heapify operation ensures that the largest element is at the root of the heap, making each extraction operation take $O(logn)$ time

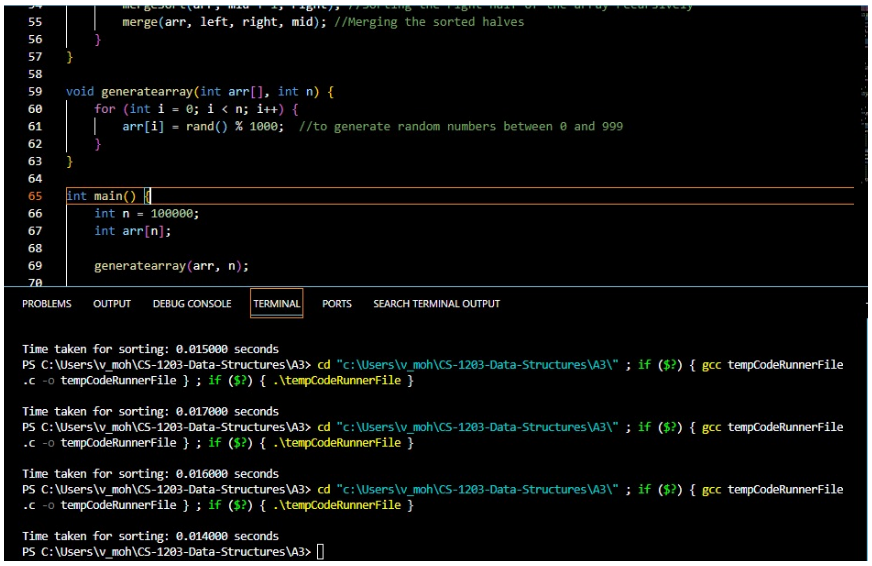
Time Complexity (Best Case): $(O(nlogn)$
Heap Sort's behavior is consistent and not dependent on the order of elements in the input array

Space Complexity: $O(1)$
It uses a constant amount of extra memory for swapping elements and maintaining the heap structure

Comparison:

Merge sort is always efficient with a time complexity of $O(nlogn)$ but may require more memory due to its use of temporary arrays. Quick sort is efficient with a best-case time complexity of $O(nlogn)$, making it a choice for practical sorting tasks. Its worst case performance can be improved with better pivot selection strategies. Heap sort is also consistently efficient with a time complexity of $O(nlogn)$ and its minimal addi-tional memory usage

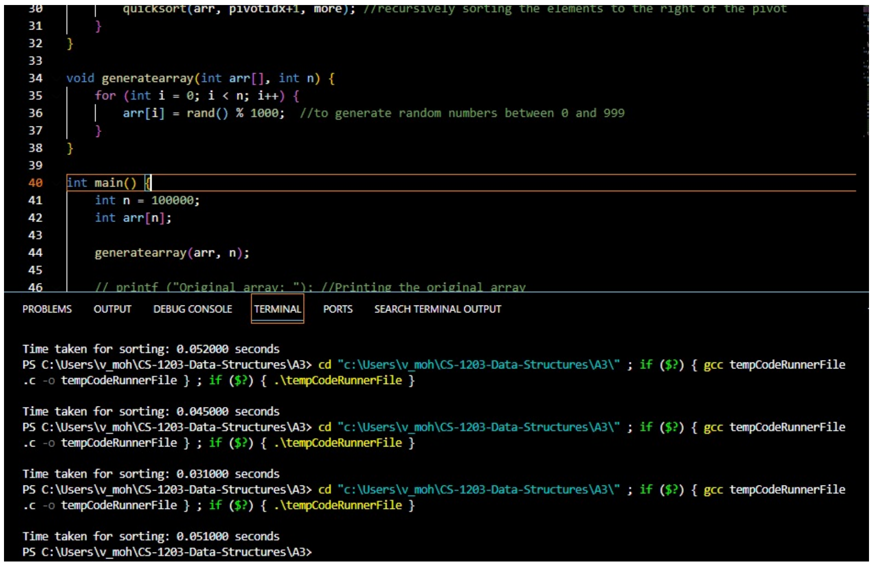
Experimental data for merge sort with an array of 1,00,000 elements whose values ranged from 0 to 999:



```
        mergeSort(arr, mid + 1, right); //Sorting the right half of the array recursively
        merge(arr, left, right, mid); //Merging the sorted halves
    }
}

void generatearray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1000;  //to generate random numbers between 0 and 999
    }
}

int main() {
    int n = 100000;
    int arr[n];

    generatearray(arr, n);
```

```
Time taken for sorting: 0.015000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Time taken for sorting: 0.017000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Time taken for sorting: 0.016000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Time taken for sorting: 0.014000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3>
```

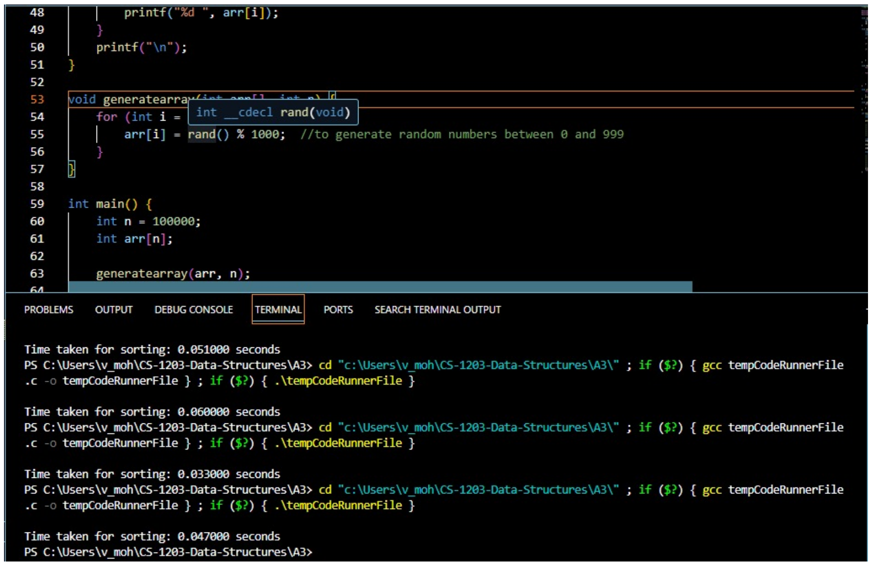
Experimental data for quick sort with an array of 1,00,000 elements whose values ranged from 0 to 999:



```
        quicksort(arr, pivotidx+1, more); //recursively sorting the elements to the right of the pivot
    }
}

void generatearray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1000;  //to generate random numbers between 0 and 999
    }
}

int main() {
    int n = 100000;
    int arr[n];

    generatearray(arr, n);

    // printf ("Original array: "); //Printing the original array
```

```
Time taken for sorting: 0.052000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Time taken for sorting: 0.045000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Time taken for sorting: 0.031000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Time taken for sorting: 0.051000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3>
```

Experimental data for heap sort with an array of 1,00,000 elements whose values ranged from 0 to 999:



```c
        printf("%d ", arr[i]);
    }
    printf("\n");
}

void generatearray(int arr[], int n) {
    for (int i = 
        arr[i] = rand() % 1000;  //to generate random numbers between 0 and 999
    }
}

int main() {
    int n = 100000;
    int arr[n];

    generatearray(arr, n);
```

```
Time taken for sorting: 0.051000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Time taken for sorting: 0.060000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Time taken for sorting: 0.033000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3> cd "c:\Users\v_moh\CS-1203-Data-Structures\A3\" ; if ($?) { gcc tempCodeRunnerFile
.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Time taken for sorting: 0.047000 seconds
PS C:\Users\v_moh\CS-1203-Data-Structures\A3>
```