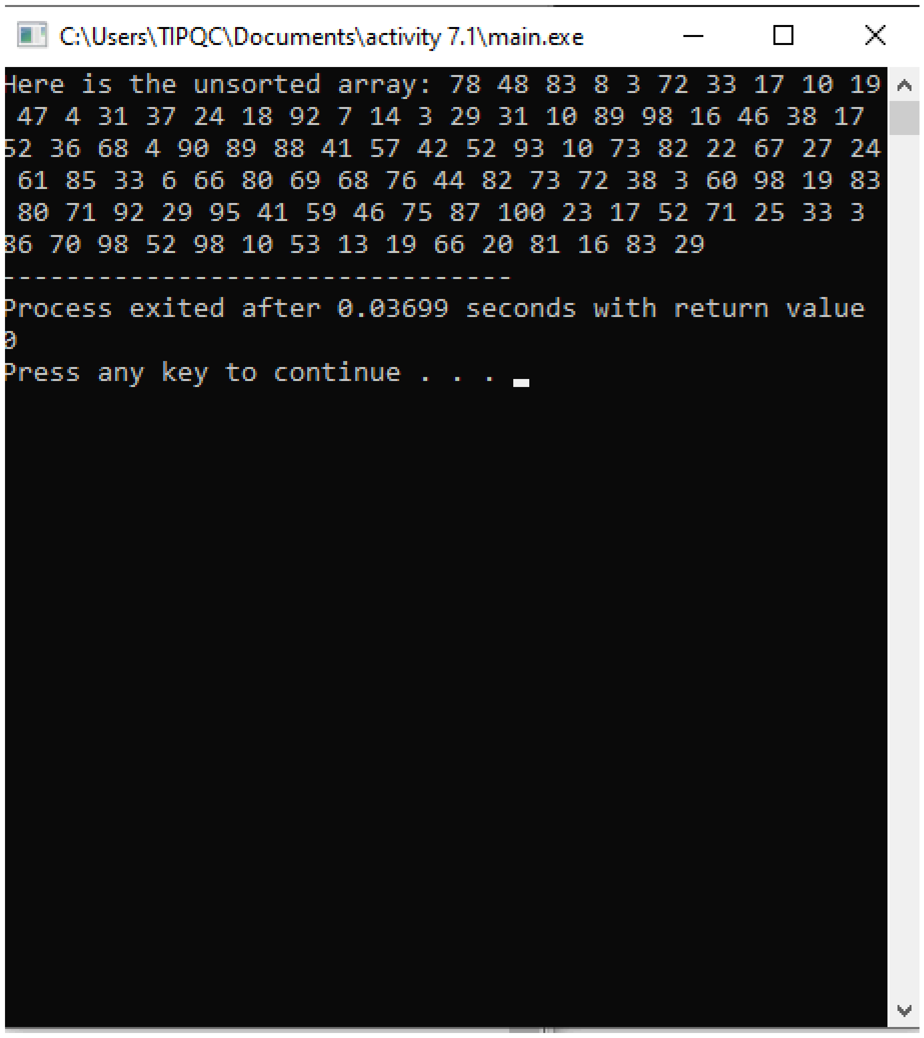
Activity No. 8	
SORTING ALGORITHM: SHELL, MERGE, AND QUICK SORT	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:10/21/2024**
Section:CPE21S4	**Date Submitted:10/23/2024**
Name(s): Santos, Ma. Kassandra Nicole	**Instructor: Ma'am Rizette Sayo**
6. Output	

Code + Console Screenshot

C/C++

```
#include <iostream>
#include <vector>
#include <ctime>
#include <cstdlib>
using namespace std;

vector<int> generateRandomArray(int size) {
    vector<int> array(size);
    srand(time(0));
    for(int i = 0; i < size; ++i) {
        array[i] = rand() % 100 + 1; // Random numbers between
1 and 100
    }
    return array;
}
    void printArray(const vector<int>& array) {
    for (size_t i = 0; i < array.size(); ++i) {
        cout << array[i] << " ";

    cout << endl;
}
}

int main() {
    vector<int> array = generateRandomArray(100);
    cout << "Here is the unsorted array: ";
    printArray(array);
    return 0;
}
```

	
Observation	After executing the code, it showed a random set of numbers.

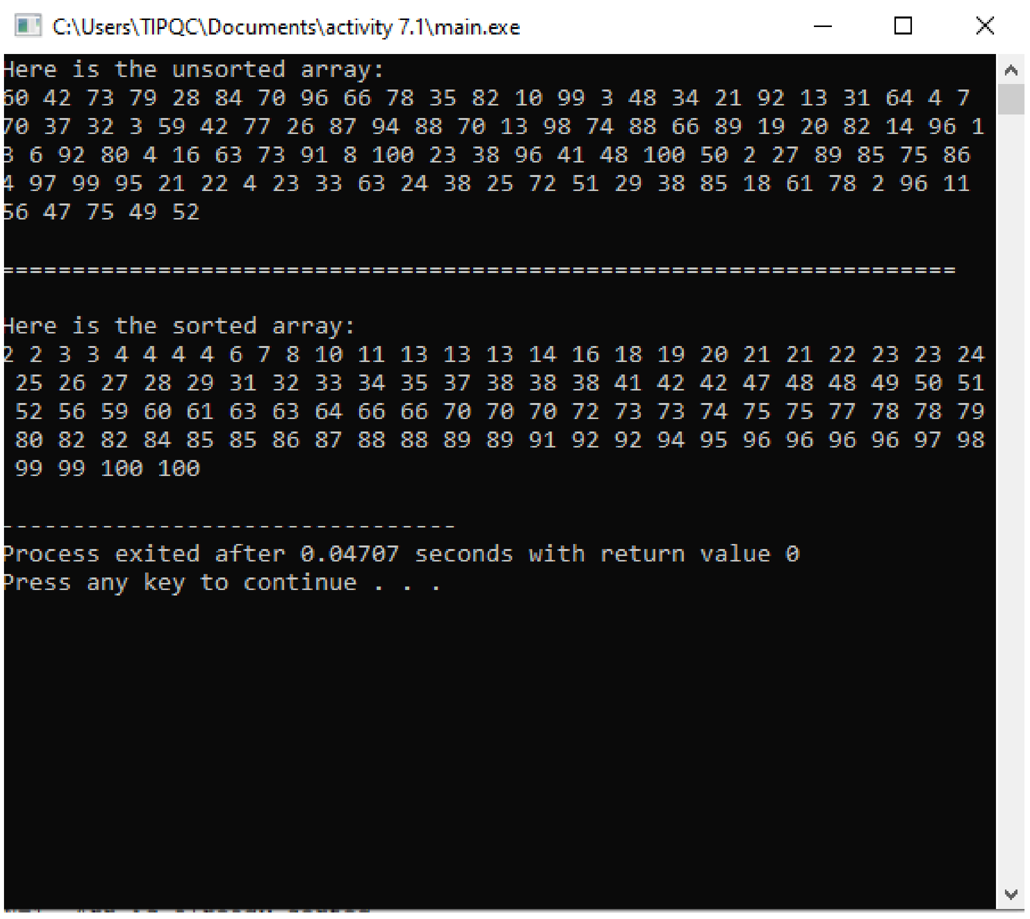
```
C:\Users\TIPQC\Documents\activity 7.1\main.exe
Here is the unsorted array: 78 48 83 8 3 72 33 17 10 19
 47 4 31 37 24 18 92 7 14 3 29 31 10 89 98 16 46 38 17
52 36 68 4 90 89 88 41 57 42 52 93 10 73 82 22 67 27 24
 61 85 33 6 66 80 69 68 76 44 82 73 72 38 3 60 98 19 83
 80 71 92 29 95 41 59 46 75 87 100 23 17 52 71 25 33 3
86 70 98 52 98 10 53 13 19 66 20 81 16 83 29
--------------------------------
Process exited after 0.03699 seconds with return value
0
Press any key to continue . . .
```

Code + Console Screenshot (Shell Sort)	

C/C++

```cpp
#include <iostream>
#include <vector>
#include <ctime>
#include <cstdlib>
using namespace std;

vector<int> generateRandomArray(int size) {
    vector<int> array(size);
    srand(time(0));
    for (int i = 0; i < size; ++i) {
        array[i] = rand() % 100 + 1; // Random numbers between 1
and 100
    }
    return array;
}
```

```cpp
void printArray(const vector<int>& array) {
    for (size_t i = 0; i < array.size(); ++i) {
        cout << array[i] << " ";
    }
    cout << endl; // New line after printing
}


void shellSort(vector<int>& array, size_t size) {
    for (size_t gap = size / 2; gap > 0; --gap) { // Start with
a large gap and then goes smaller through a decrement
        for (size_t i = gap; i < size; ++i) { \
            int temp = array[i]; // Value to place
            size_t j = i;


            while (j >= gap && array[j - gap] > temp) {
                array[j] = array[j - gap];
                j -= gap;
            }
            array[j] = temp; // Place temp in its position
        }
    }
}

int main() {
    vector<int> array = generateRandomArray(100);
    cout << "Here is the unsorted array: ";
    cout << endl;
    printArray(array);

       cout << endl;
       cout <<
"================================================================
====";
       cout <<endl;
       cout <<endl;

    shellSort(array, array.size());
    cout << "Here is the sorted array: ";
    cout << endl;
    printArray(array);

    return 0;
}
```

	
Observation	After executing the code, it manages to sort the random set of numbers from smallest to largest

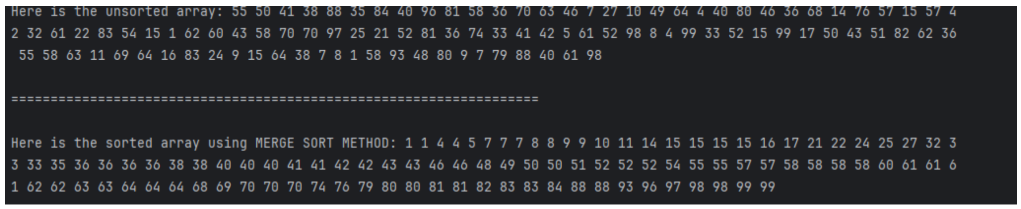
C:\Users\TIPQC\Documents\activity 7.1\main.exe

```
Here is the unsorted array:
60 42 73 79 28 84 70 96 66 78 35 82 10 99 3 48 34 21 92 13 31 64 4 7
70 37 32 3 59 42 77 26 87 94 88 70 13 98 74 88 66 89 19 20 82 14 96 1
8 6 92 80 4 16 63 73 91 8 100 23 38 96 41 48 100 50 2 27 89 85 75 86
4 97 99 95 21 22 4 23 33 63 24 38 25 72 51 29 38 85 18 61 78 2 96 11
56 47 75 49 52

===================================================================

Here is the sorted array:
2 2 3 3 4 4 4 4 6 7 8 10 11 13 13 13 14 16 18 19 20 21 21 22 23 23 24
 25 26 27 28 29 31 32 33 34 35 37 38 38 38 41 42 42 47 48 48 49 50 51
 52 56 59 60 61 63 63 64 66 66 70 70 70 72 73 73 74 75 75 77 78 78 79
 80 82 82 84 85 85 86 87 88 88 89 89 91 92 92 94 95 96 96 96 96 97 98
 99 99 100 100

--------------------------------
Process exited after 0.04707 seconds with return value 0
Press any key to continue . . .
```

Code + Console Screenshot	

C/C++

```cpp
#include <iostream>
#include <vector>
#include <ctime>
#include <cstdlib>
using namespace std;

vector<int> generateRandomArray(int size) {
    vector<int> array(size);
    srand(time(0));
    for (int i = 0; i < size; ++i) {
        array[i] = rand() % 100 + 1; // Random numbers between
1 and 100
    }
    return array;
}

void printArray(const vector<int>& array) {
    for (size_t i = 0; i < array.size(); ++i) {
        cout << array[i] << " ";
    }
    cout << endl; // New line after printing
}

void merge(vector<int>& array, int left, int center, int right)
{
    vector<int> L(array.begin() + left, array.begin() + center
+ 1);
    vector<int> R(array.begin() + center + 1, array.begin() +
right + 1);

    int i = 0, j = 0, k = left;
    while (i < L.size() && j < R.size()) {
        array[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];
    }

    while (i < L.size()) array[k++] = L[i++];
    while (j < R.size()) array[k++] = R[j++];
}


void mergeSort(vector<int>& array, int left, int right) {
    if (left < right) {
        int center = left + (right - left) / 2;

        mergeSort(array, left, center);
        mergeSort(array, center + 1, right);

        merge(array, left, center, right);
    }
}

int main() {
    vector<int> array = generateRandomArray(100);

    cout << "Here is the unsorted array: ";
    printArray(array);

    cout << endl;
    cout <<
"==============================================================
=====";
    cout <<endl;
    cout <<endl;

    mergeSort(array, 0, array.size() - 1);

    cout << "Here is the sorted array using MERGE SORT METHOD:
";
    printArray(array);

    return 0;
}
```

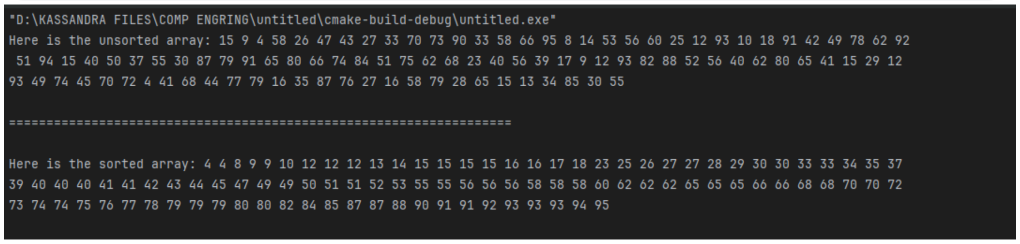


```
Here is the unsorted array: 55 50 41 38 88 35 84 40 96 81 58 36 70 63 46 7 27 10 49 64 4 40 80 46 36 68 14 76 57 15 57 4
2 32 61 22 83 54 15 1 62 60 43 58 70 70 97 25 21 52 81 36 74 33 41 42 5 61 52 98 8 4 99 33 52 15 99 17 50 43 51 82 62 36
 55 58 63 11 69 64 16 83 24 9 15 64 38 7 8 1 58 93 48 80 9 7 79 88 40 61 98

==================================================================

Here is the sorted array using MERGE SORT METHOD: 1 1 4 4 5 7 7 7 8 8 9 9 10 11 14 15 15 15 15 16 17 21 22 24 25 27 32 3
3 33 35 36 36 36 36 38 38 40 40 40 41 41 42 42 43 43 46 46 48 49 50 50 51 52 52 52 54 55 55 57 57 58 58 58 58 60 61 61 6
1 62 62 63 63 64 64 64 68 69 70 70 70 74 76 79 80 80 81 81 82 83 83 84 88 88 93 96 97 98 98 99 99
```

observation	After executing the code, it manages to sort the random set of numbers from smallest to largest

```
C/C++
#include <iostream>
#include <vector>
#include <cstdlib>
using namespace std;

vector<int> generateRandomArray(int size) {
    vector<int> array(size);
    srand(time(0));
    for (int &x : array) x = rand() % 100 + 1; // Random
numbers between 1 and 100
    return array;
}

void printArray(const vector<int>& array) {
    for (int x : array) cout << x << " ";
    cout << endl;
}

int partition(vector<int>& array, int low, int high) {
    int pivot = array[high], i = low - 1;
    for (int j = low; j < high; ++j)
        if (array[j] < pivot) swap(array[++i], array[j]);
    swap(array[i + 1], array[high]);
    return i + 1;
}

void quicksort(vector<int>& array, int low, int high) {
    if (low < high) {
        int pivot = partition(array, low, high);
        quicksort(array, low, pivot - 1);
        quicksort(array, pivot + 1, high);
    }
}

int main() {
    vector<int> array = generateRandomArray(100);
    cout << "Here is the unsorted array: "; printArray(array);

    cout << endl;
    cout <<
"================================================================
=====";
```

```cpp
        cout <<endl;
        cout <<endl;

        quicksort(array, 0, array.size() - 1);
        cout << "Here is the sorted array: "; printArray(array);
        return 0;
    }
```



```
"D:\KASSANDRA FILES\COMP ENGRING\untitled\cmake-build-debug\untitled.exe"
Here is the unsorted array: 15 9 4 58 26 47 43 27 33 70 73 90 33 58 66 95 8 14 53 56 60 25 12 93 10 18 91 42 49 78 62 92
 51 94 15 40 50 37 55 30 87 79 91 65 80 66 74 84 51 75 62 68 23 40 56 39 17 9 12 93 82 88 52 56 40 62 80 65 41 15 29 12
93 49 74 45 70 72 4 41 68 44 77 79 16 35 87 76 27 16 58 79 28 65 15 13 34 85 30 55

==================================================================

Here is the sorted array: 4 4 8 9 9 10 12 12 12 13 14 15 15 15 15 16 16 17 18 23 25 26 27 27 28 29 30 30 33 33 34 35 37
39 40 40 40 41 41 42 43 44 45 47 49 49 50 51 51 52 53 55 55 56 56 56 58 58 58 60 62 62 62 65 65 65 66 66 68 68 70 70 72
73 74 74 75 76 77 78 79 79 79 80 80 82 84 85 87 87 88 90 91 91 92 93 93 93 94 95
```

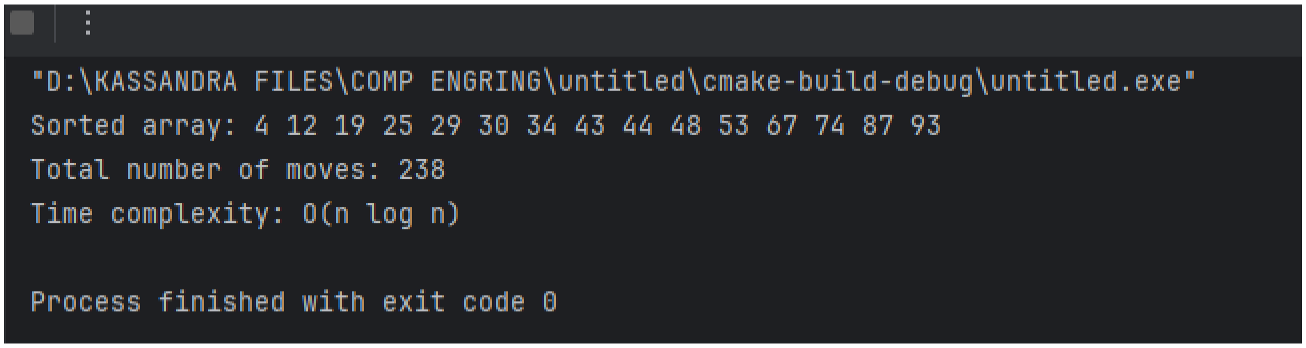
After executing the code, it manages to sort the random set of numbers from smallest to largest

7. Supplementary Activity

2. For me to answer this question I added a program where it counts the amount of moves the program makes in order for the array to be sorted, here are the codes. The first is the quicksort followed by the merge:

```cpp
C/C++
#include <iostream>
#include <vector>

using namespace std;

void quicksort(vector<int>& arr, int left, int right, int& moveCount);
void quicksort(vector<int>& arr, int left, int right, int& moveCount) {
    if (left < right) {
        int pivot = arr[right];
        int i = left - 1;

        for (int j = left; j < right; ++j) {
            if (arr[j] < pivot) {
                i++;
                swap(arr[i], arr[j]);
                moveCount += 2;
            }
        }
        swap(arr[i + 1], arr[right]);
        moveCount += 2;
        int partitionIndex = i + 1;

        quicksort(arr, left, partitionIndex - 1, moveCount);
        quicksort(arr, partitionIndex + 1, right, moveCount);
        }
    }

    int main() {
        vector<int> my_array = {4, 12, 19, 25, 29, 30, 34, 43, 44, 48, 53, 67, 74, 87, 93};
        int moveCount = 0;

        quicksort(my_array, 0, my_array.size() - 1, moveCount);


        cout << "Sorted array: ";
        for (int x : my_array) {
            cout << x << " ";
        }
        cout << endl;

        cout << "Total number of moves: " << moveCount << endl;
        cout << "Time complexity: O(n log n)" << endl;

        return 0;
    }
```

OUTPUT



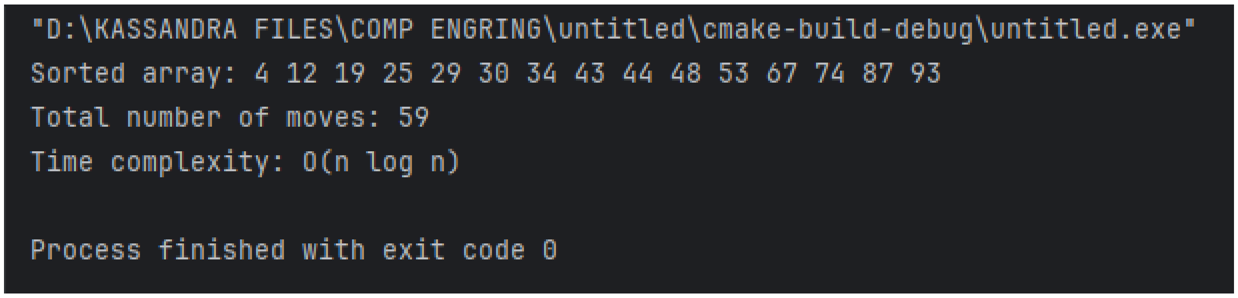
```
"D:\KASSANDRA FILES\COMP ENGRING\untitled\cmake-build-debug\untitled.exe"
Sorted array: 4 12 19 25 29 30 34 43 44 48 53 67 74 87 93
Total number of moves: 238
Time complexity: O(n log n)

Process finished with exit code 0
```

MERGE

C/C++

```
#include <iostream>
#include <vector>

using namespace std;


void merge(vector<int>& arr, int left, int mid, int right, int& moveCount) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    vector<int> leftArr(arr.begin() + left, arr.begin() + mid + 1);
    vector<int> rightArr(arr.begin() + mid + 1, arr.begin() + right + 1);

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        arr[k++] = (leftArr[i] <= rightArr[j]) ? leftArr[i++] : rightArr[j++];
        moveCount++;
    }
    while (i < n1) {
        arr[k++] = leftArr[i++];
        moveCount++;
    }
    while (j < n2) {
        arr[k++] = rightArr[j++];
        moveCount++;
    }
}


void mergeSort(vector<int>& arr, int left, int right, int& moveCount) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid, moveCount);
        mergeSort(arr, mid + 1, right, moveCount);
        merge(arr, left, mid, right, moveCount);
    }
}

int main() {
    vector<int> my_array = {4, 12, 19, 25, 29, 30, 34, 43, 44, 48, 53, 67, 74, 87, 93};
    int moveCount = 0;

    mergeSort(my_array, 0, my_array.size() - 1, moveCount);

    cout << "Sorted array: ";
    for (int x : my_array) {
        cout << x << " ";
    }
    cout << endl;

    cout << "Total number of moves: " << moveCount << endl;
    cout << "Time complexity: O(n log n)" << endl;

    return 0;
}
```

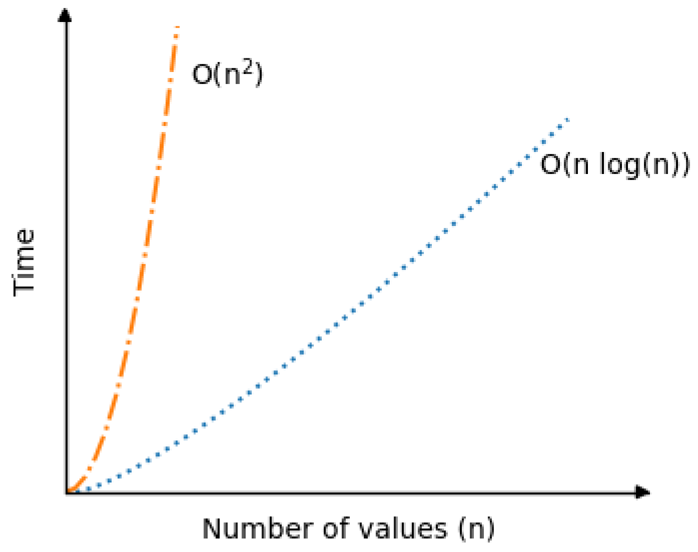
OUTPUT



```
"D:\KASSANDRA FILES\COMP ENGRING\untitled\cmake-build-debug\untitled.exe"
Sorted array: 4 12 19 25 29 30 34 43 44 48 53 67 74 87 93
Total number of moves: 59
Time complexity: O(n log n)

Process finished with exit code 0
```

Based on the output, it is clear that the merge method is faster in sorting than quick sort, it is pretty common sense compared to quick sort wherein the program would have to pick out each element one by one; swapping and repositioning them around the pivot which requires multiple operations of moves for each component; compared to merge sort since it copies elements from a temporary array (since it separates them first), reposition it and then the return is the original set which results to minimal effort.

I would like to base my answer on this diagram on why merge sort and quick sort have O(N • log N) for their time complexity.



As you can see, the average time complexity of O(nlog n) is efficient for handling large datasets. However, as the number of values increases, the longer the system would have to sort the elements but it is still manageable. In contrast, the worst-case complexity (O(n^2)) can lead to longer sorting time even for smaller datasets. Given in this scenario the worst-case complexity is particularly inefficient and can be time-consuming especially as the dataset size increases

8. Conclusion

In conclusion, we learned various methods on how to sort an unorganized data set no matter how big is it, not only that we were able to learn as to why average time complexity is most commonly used in sorting methods. The procedure is simple as we needed to show the codes of three different sorting techniques such as the shell, merge, and quick sort and our objective was to observe how each methods sort a random number that has a data size of 100. In the supplementary activity, I learned which sorting methods are faster and from the supplementary, I learned that merge can be faster and quick sort, however, I do believe that it is subjective and the efficiency of the sorting methods can depend on the scenario, I also learned why average time complexity is more efficient in sorting out large datasets compres to worst-case complexity. I think that in this activity, I did good since I got to understand how each sorting methods works.

9. Assessment Rubric