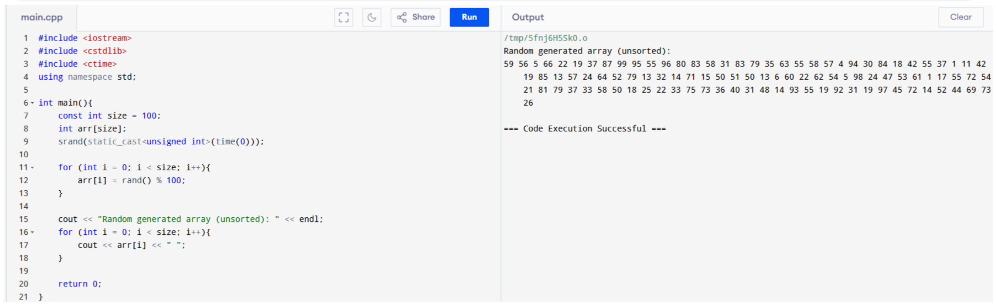
ACTIVITY NO. 8

SORTING ALGORITHMS: SHELL, MERGE, AND QUICK SORT	
Course Code: CPE021	**Program:** Computer Engineering
Course Title: Computer Architecture and Organization	**Date Performed:** 10/21/24
Section: CPE21S4	**Date Submitted:** 10/21/24
Name(s): **CALVIN EARL PLANTA**	**Instructor:** Prof. Sayo
6. Output	

Code + Console Screenshot

C/C++

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

int main(){
	const int size = 100;
	int arr[size];
	srand(static_cast<unsigned int>(time(0)));

	for (int i = 0; i < size; i++){
	arr[i] = rand() % 100;
	}

	cout << "Random generated array (unsorted): " << endl;
	for (int i = 0; i < size; i++){
	cout << arr[i] << " ";
	}

	return 0;
}
```



main.cpp

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

int main(){
    const int size = 100;
    int arr[size];
    srand(static_cast<unsigned int>(time(0)));

    for (int i = 0; i < size; i++){
        arr[i] = rand() % 100;
    }

    cout << "Random generated array (unsorted): " << endl;
    for (int i = 0; i < size; i++){
        cout << arr[i] << " ";
    }

    return 0;
}
```

Output

```
/tmp/5fnj6H5Sk0.o
Random generated array (unsorted):
59 56 5 66 22 19 37 87 99 95 55 96 80 83 58 31 83 79 35 63 55 58 57 4 94 30 84 18 42 55 37 1 11 42
    19 85 13 57 24 64 52 79 13 32 14 71 15 50 51 50 13 6 60 22 62 54 5 98 24 47 53 61 1 17 55 72 54
    21 81 79 37 33 58 50 18 25 22 33 75 73 36 40 31 48 14 93 55 19 92 31 19 97 45 72 14 52 44 69 73
    26

=== Code Execution Successful ===
```

Observations

The output of the program seemed complex yet the code was kept simple. The array was generated using rand() inside a for loop function that iterates as long as the iteration count is less than the size of the array.

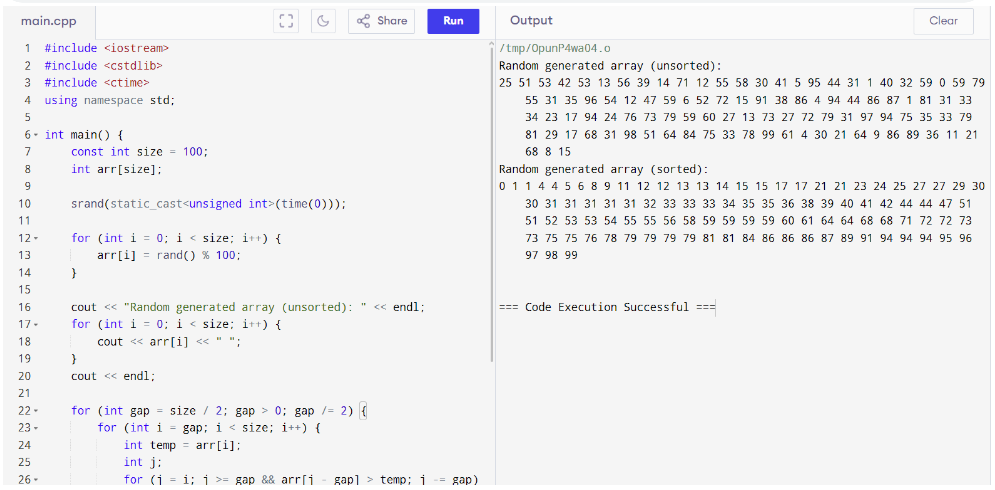
Table 8-1. Array of Values for Sort Algorithm Testing

Code + Console Screenshot

C/C++

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

```
int main() {
    const int size = 100;
    int arr[size];

    srand(static_cast<unsigned int>(time(0)));

    for (int i = 0; i < size; i++) {
        arr[i] = rand() % 100;
    }

    cout << "Random generated array (unsorted): " << endl;
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    for (int gap = size / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < size; i++) {
            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -=
gap) {
                arr[j] = arr[j - gap];
            }
            arr[j] = temp;
        }
    }

    cout << "Random generated array (sorted): " << endl;
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```



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

int main() {
    const int size = 100;
    int arr[size];

    srand(static_cast<unsigned int>(time(0)));

    for (int i = 0; i < size; i++) {
        arr[i] = rand() % 100;
    }

    cout << "Random generated array (unsorted): " << endl;
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    for (int gap = size / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < size; i++) {
            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
```

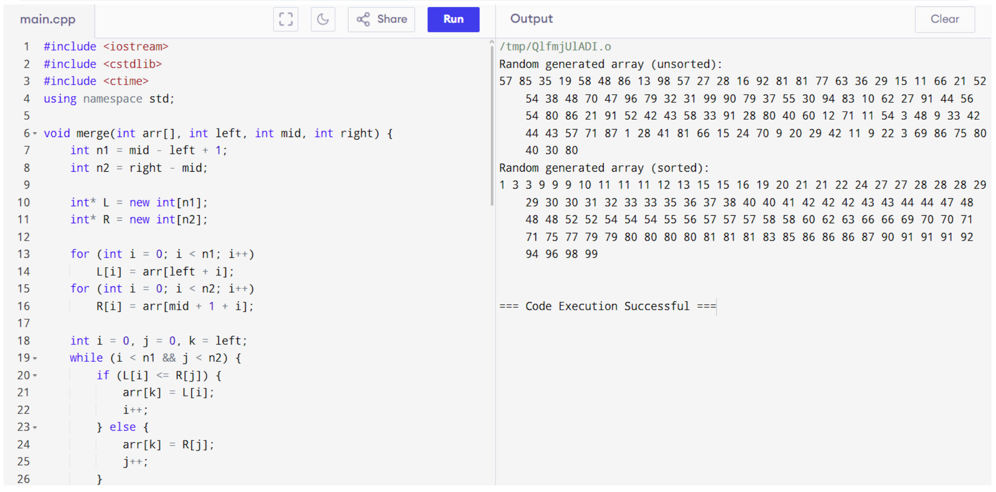
```
/tmp/OpunP4wa04.o
Random generated array (unsorted):
25 51 53 42 53 13 56 39 14 71 12 55 58 30 41 5 95 44 31 1 40 32 59 0 59 79
    55 31 35 96 54 12 47 59 6 52 72 15 91 38 86 4 94 44 86 87 1 81 31 33
    34 23 17 94 24 76 73 79 59 60 27 13 73 27 72 79 31 97 94 75 35 33 79
    81 29 17 68 31 98 51 64 84 75 33 78 99 61 4 30 21 64 9 86 89 36 11 21
    68 8 15
Random generated array (sorted):
0 1 1 4 4 5 6 8 9 11 12 12 13 13 14 15 15 17 17 21 21 23 24 25 27 27 29 30
    30 31 31 31 31 31 32 33 33 33 34 35 35 36 38 39 40 41 42 44 44 47 51
    51 52 53 53 54 55 55 56 58 59 59 59 59 60 61 64 64 68 68 71 72 72 73
    73 75 75 76 78 79 79 79 79 81 81 84 86 86 86 87 89 91 94 94 94 95 96
    97 98 99


=== Code Execution Successful ===
```

Observations	Shell Sort is more efficient than Bubble Sort for large arrays because it reduces the number of comparisons needed by first sorting elements that are far apart and progressively reducing the gap between elements.

Table 8-2. Shell Sort Technique

Code + Console Screenshot	C/C++
	 ```#include <iostream>```
	 ```#include <cstdlib>```
	 ```#include <ctime>```
	 ```using namespace std;```
	
	 ```void merge(int arr[], int left, int mid, int right) {```
	 ``` int n1 = mid - left + 1;```
	 ``` int n2 = right - mid;```
	
	 ``` int* L = new int[n1];```
	 ``` int* R = new int[n2];```
	
	 ``` for (int i = 0; i < n1; i++)```
	 ``` L[i] = arr[left + i];```
	 ``` for (int i = 0; i < n2; i++)```
	 ``` R[i] = arr[mid + 1 + i];```
	
	 ``` int i = 0, j = 0, k = left;```
	 ``` while (i < n1 && j < n2) {```
	 ``` if (L[i] <= R[j]) {```
	 ``` arr[k] = L[i];```
	 ``` i++;```
	 ``` } else {```
	 ``` arr[k] = R[j];```
	 ``` j++;```
	 ``` }```
	 ``` k++;```
	 ``` }```
	
	 ``` while (i < n1) {```
	 ``` arr[k] = L[i];```
	 ``` i++;```
	 ``` k++;```
	 ``` }```
	
	 ``` while (j < n2) {```
	 ``` arr[k] = R[j];```
	 ``` j++;```
	 ``` k++;```
	 ``` }```
	
	 ``` delete[] L;```
	 ``` delete[] R;```
	 ```}```
	
	 ```void Sort(int arr[], int left, int right) {```

```cpp
    if (left < right) {
        int mid = left + (right - left) / 2;

        Sort(arr, left, mid);
        Sort(arr, mid + 1, right);

        merge(arr, left, mid, right);
    }
}

int main() {
    const int size = 100;
    int arr[size];

    srand(static_cast<unsigned int>(time(0)));

    for (int i = 0; i < size; i++) {
        arr[i] = rand() % 100;
    }

    cout << "Random generated array (unsorted): " << endl;
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    Sort(arr, 0, size - 1);

    cout << "Random generated array (sorted): " << endl;
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```



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

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int* L = new int[n1];
    int* R = new int[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int i = 0; i < n2; i++)
        R[i] = arr[mid + 1 + i];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
```

```
/tmp/QlfmjUlADI.o
Random generated array (unsorted):
57 85 35 19 58 48 86 13 98 57 27 28 16 92 81 81 77 63 36 29 15 11 66 21 52
    54 38 48 70 47 96 79 32 31 99 90 79 37 55 30 94 83 10 62 27 91 44 56
    54 80 86 21 91 52 42 43 58 33 91 28 80 40 60 12 71 11 54 3 48 9 33 42
    44 43 57 71 87 1 28 41 81 66 15 24 70 9 20 29 42 11 9 22 3 69 86 75 80
    40 30 80
Random generated array (sorted):
1 3 3 9 9 9 10 11 11 11 12 13 15 15 16 19 20 21 21 22 24 27 27 28 28 28 29
    29 30 30 31 32 33 33 35 36 37 38 40 40 41 42 42 42 43 43 44 44 47 48
    48 48 52 52 54 54 54 55 56 57 57 57 58 58 60 62 63 66 66 69 70 70 71
    71 75 77 79 79 80 80 80 80 81 81 81 83 85 86 86 86 87 90 91 91 91 92
    94 96 98 99


=== Code Execution Successful ===
```

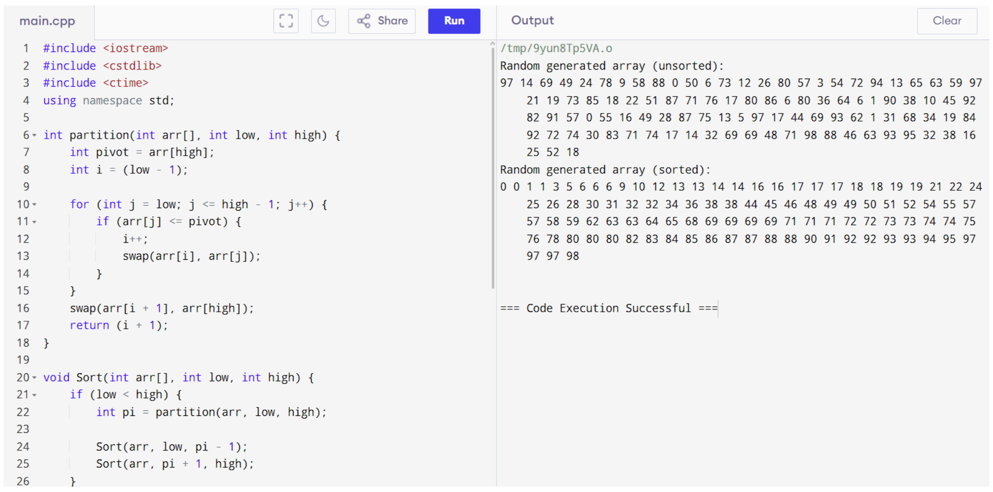
Observations	Merge Sort maintains the relative order of equal elements, and is much faster than

	algorithms like Bubble Sort, especially for large arrays, because of its logarithmic nature.

Table 8-3. Merge Sort Algorithm

Code + Console Screenshot	C/C++

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

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return (i + 1);
}

void Sort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);

        Sort(arr, low, pi - 1);
        Sort(arr, pi + 1, high);
    }
}

int main() {
    const int size = 100;
    int arr[size];

    srand(static_cast<unsigned int>(time(0)));

    for (int i = 0; i < size; i++) {
        arr[i] = rand() % 100;
    }

    cout << "Random generated array (unsorted): " << endl;
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    Sort(arr, 0, size - 1);

    cout << "Random generated array (sorted): " << endl;
```

	```
	    for (int i = 0; i < size; i++) {
	        cout << arr[i] << " ";
	    }
	    cout << endl;

	    return 0;
	}
	```
	
	main.cpp
	```
	#include <iostream>
	#include <cstdlib>
	#include <ctime>
	using namespace std;

	int partition(int arr[], int low, int high) {
	    int pivot = arr[high];
	    int i = (low - 1);

	    for (int j = low; j <= high - 1; j++) {
	        if (arr[j] <= pivot) {
	            i++;
	            swap(arr[i], arr[j]);
	        }
	    }
	    swap(arr[i + 1], arr[high]);
	    return (i + 1);
	}

	void Sort(int arr[], int low, int high) {
	    if (low < high) {
	        int pi = partition(arr, low, high);

	        Sort(arr, low, pi - 1);
	        Sort(arr, pi + 1, high);
	    }
	```
	Output
	```
	/tmp/9yun8Tp5VA.o
	Random generated array (unsorted):
	97 14 69 49 24 78 9 58 88 0 50 6 73 12 26 80 57 3 54 72 94 13 65 63 59 97
	    21 19 73 85 18 22 51 87 71 76 17 80 86 6 80 36 64 6 1 90 38 10 45 92
	    82 91 57 0 55 16 49 28 87 75 13 5 97 17 44 69 93 62 1 31 68 34 19 84
	    92 72 74 30 83 71 74 17 14 32 69 69 48 71 98 88 46 63 93 95 32 38 16
	    25 52 18
	Random generated array (sorted):
	0 0 1 1 3 5 6 6 6 9 10 12 13 13 14 14 16 16 17 17 17 18 18 19 19 21 22 24
	    25 26 28 30 31 32 32 34 36 38 38 44 45 46 48 49 49 50 51 52 54 55 57
	    57 58 59 62 63 63 64 65 68 69 69 69 69 71 71 71 72 72 73 73 74 74 75
	    76 78 80 80 80 82 83 84 85 86 87 87 88 88 90 91 92 92 93 93 94 95 97
	    97 97 98


	=== Code Execution Successful ===
	```

Observations	The pivot selection has a significant impact on Quick Sort's efficiency. While bad pivot decisions might result in unbalanced partitions and decreased performance, good decisions (such as selecting a median) produce balanced partitions.

Table 8-4. Quick Sort Algorithm

7. Supplementary Activity

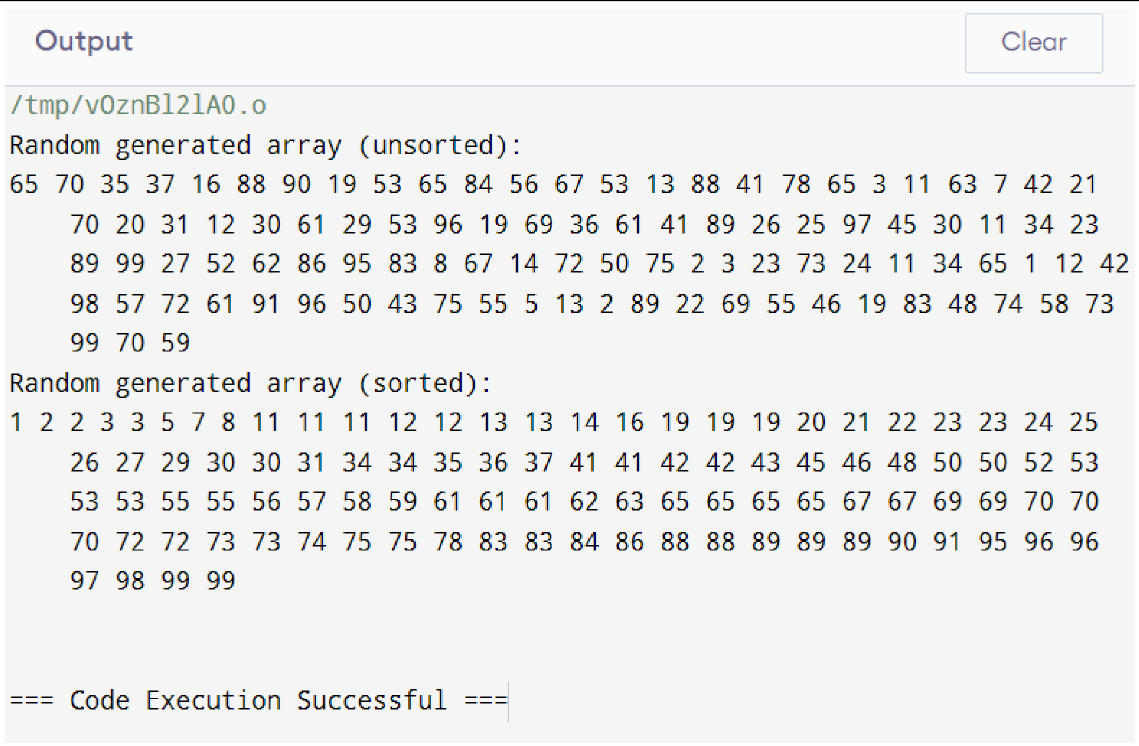
ILO B: Solve given data sorting problems using appropriate basic sorting algorithms

Problem 1: Can we sort the left sub list and right sub list from the partition method in quick sort using other sorting algorithms? Demonstrate an example.

Yes, in C++ you can sort the left and right sublists from the partition method in Quick Sort using other sorting algorithms, such as Merge Sort, Insertion Sort, or Bubble Sort. This technique is sometimes referred to as hybrid sorting.

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

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);
```

```cpp
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return (i + 1);
}

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int* L = new int[n1];
    int* R = new int[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int i = 0; i < n2; i++)
        R[i] = arr[mid + 1 + i];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }

    delete[] L;
    delete[] R;
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

void insertionSort(int arr[], int low, int high) {
    for (int i = low + 1; i <= high; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= low && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

void Sort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);

        mergeSort(arr, low, pi - 1);

        insertionSort(arr, pi + 1, high);
    }
}

int main() {
    const int size = 100;
    int arr[size];

    srand(static_cast<unsigned int>(time(0)));

    for (int i = 0; i < size; i++) {
        arr[i] = rand() % 100;
    }


    cout << "Random generated array (unsorted): " << endl;
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    Sort(arr, 0, size - 1);

    cout << "Random generated array (sorted): " << endl;
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```



```
/tmp/vOznBl2lA0.o
Random generated array (unsorted):
65 70 35 37 16 88 90 19 53 65 84 56 67 53 13 88 41 78 65 3 11 63 7 42 21
    70 20 31 12 30 61 29 53 96 19 69 36 61 41 89 26 25 97 45 30 11 34 23
    89 99 27 52 62 86 95 83 8 67 14 72 50 75 2 3 23 73 24 11 34 65 1 12 42
    98 57 72 61 91 96 50 43 75 55 5 13 2 89 22 69 55 46 19 83 48 74 58 73
    99 70 59
Random generated array (sorted):
1 2 2 3 3 5 7 8 11 11 11 12 12 13 13 14 16 19 19 19 20 21 22 23 23 24 25
    26 27 29 30 30 31 34 34 35 36 37 41 41 42 42 43 45 46 48 50 50 52 53
    53 53 55 55 56 57 58 59 61 61 61 62 63 65 65 65 65 67 67 69 69 70 70
    70 72 72 73 73 74 75 75 78 83 83 84 86 88 88 89 89 89 90 91 95 96 96
    97 98 99 99


=== Code Execution Successful ===
```

Problem 2: Suppose we have an array which consists of {4, 34, 29, 48, 53, 87, 12, 30, 44, 25, 93, 67, 43, 19, 74}. What sorting algorithm will give you the fastest time performance? Why can merge sort and quick sort have O(N • log N) for their time complexity?

For the array {4, 34, 29, 48, 53, 87, 12, 30, 44, 25, 93, 67, 43, 19, 74}, Quick Sort will likely give the best performance due to its average-case time complexity and practical in-place sorting advantage. Both Merge Sort and Quick Sort share the same O(NlogN) time complexity, but their behavior differs based on the input and memory overhead considerations.

8. Conclusion

In this experiment, we were further introduced to more sorting techniques in C++, namely the shell sort, the merge sort and the quick sort techniques. Shell Sort is a comparison-based sorting algorithm that generalizes Insertion Sort to allow the exchange of elements that are far apart. The merge sort algorithm utilizes the "divide and conquer" tactic, which breaks the problem down into smaller issues.and deals with those subproblems separately. Quicksort algorithm chooses a certain element known as the "pivot" and divides the list or array to be divided into two halves according to this pivot s0, where the components below the pivot are to elements larger than the pivot are at the right of the list, while the elements at the left are at the left. Every algorithm has trade-offs between complexity, memory utilization, and efficiency, which allows them to be used with various kinds of data and situations.