

Activity No. 7	
Hands-on Activity 7.2 Sorting Algorithms	
Course Code: CPE010	Program: Computer Engineering
Course Title: Data Structures and Algorithms	Date Performed: October 21, 2024
Section: CPE21S4	Date Submitted: October 23, 2024
Name(s): Don Eleazar T. Fernandez	Instructor: Maria Rizette Sayo

6. Output

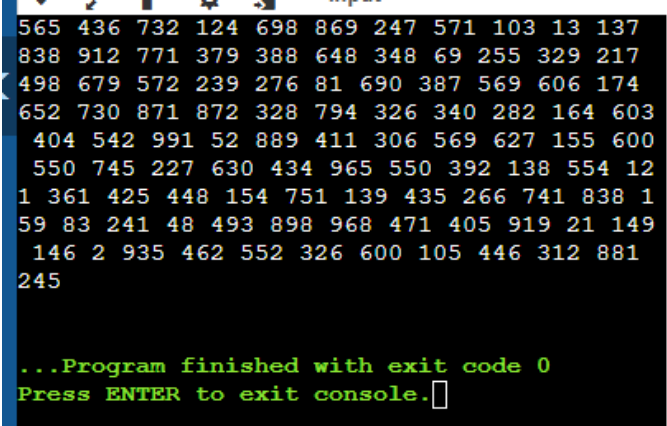
Code + Console Screenshot	<pre>#include <iostream> #include <ctime> using namespace std; int main() { const int size = 100; int array[size]; srand(static_cast<unsigned int>(time(0))); for (int i = 0; i < size; ++i) { array[i] = rand() % 1000 + 1; } for (int i = 0; i < size; ++i) { cout << array[i] << " "; } cout << endl; return 0; }</pre> 
Observations	The program has performed as intended, which is to create a one hundred random elements of an array.

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

Code + Console Screenshot	#include <iostream>
---------------------------	---------------------

```

#include <ctime>
using namespace std;

void shellSort(int array[], int size) {
    for (int interval = size / 2; interval > 0; interval /= 2) {
        for (int i = interval; i < size; i++) {
            int temp = array[i];
            int j;
            for (j = i; j >= interval && array[j - interval] > temp; j
            -= interval) {
                array[j] = array[j - interval];
            }
            array[j] = temp;
        }
    }
}

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

    for (int i = 0; i < size; ++i) {
        array[i] = rand() % 1000 + 1;
    }
    cout << "Unsorted Array: ";
    for (int i = 0; i < size; ++i) {
        cout << array[i] << " ";
    }
    cout << "\n\n";

    shellSort(array, size);
    cout << "Sorted Array: ";
    for (int i = 0; i < size; ++i) {
        cout << array[i] << " ";
    }
    cout << endl;

    return 0;
}

```

```

input
Unsorted Array: 27 133 880 868 743 515 501 548 890
147 543 232 429 925 781 550 968 291 640 447 533 559
517 284 880 700 762 165 732 613 852 758 98 732 625
192 598 478 740 839 976 282 422 756 559 202 305 87
8 845 297 325 377 855 193 660 734 245 773 898 976 7
38 101 733 835 832 709 378 782 538 469 620 865 103
42 621 661 595 277 538 791 573 214 520 779 407 531
512 651 304 761 626 41 862 710 227 45 770 956 178 3
08

Sorted Array: 27 41 42 45 98 101 103 133 147 165 17
8 192 193 202 214 227 232 245 277 282 284 291 297 3
04 305 308 325 377 378 407 422 429 447 469 478 501
512 515 517 520 531 533 538 538 543 548 550 559 559
573 595 598 613 620 621 625 626 640 651 660 661 70
0 709 710 732 732 733 734 738 740 743 756 758 761 7
62 770 773 779 781 782 791 832 835 839 845 852 855
862 865 868 878 880 880 890 898 925 956 968 976 976

...Program finished with exit code 0
Press ENTER to exit console.

```

Observations

The program sorts the declared array with the Shell Sort method. It first displays the unsorted elements and then displays them sorted.

Table 8 - 2. Shell Sort Technique

Code + Console Screenshot

```

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

void merge(int arr[], int left, int middle, int right) {
    int n1 = middle - left + 1;
    int n2 = right - middle;
    vector<int> L(n1), R(n2);
    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[middle + 1 + j];

    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++;
    }
}

void merge_sort(int arr[], int left, int right) {
    if (left < right) {
        int middle = left + (right - left) / 2;
        merge_sort(arr, left, middle);
        merge_sort(arr, middle + 1, right);
        merge(arr, left, middle, right);
    }
}

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

    for (int i = 0; i < size; ++i) {
        array[i] = rand() % 1000 + 1;
    }

    cout << "Original array: ";
    for (int i = 0; i < size; ++i) {
        cout << array[i] << " ";
    }
    cout << "\n\n";

    merge_sort(array, 0, size - 1);

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

    return 0;
}

```

```

Original array: 229 737 692 598 130 718 842 491 799
353 821 411 775 228 130 909 498 457 707 821 366 36
7 892 239 79 504 273 316 728 308 259 956 45 303 905
174 20 746 664 818 99 484 581 225 63 62 133 913 51
8 191 85 236 909 328 474 987 183 98 654 910 406 913
866 802 215 122 975 586 868 990 404 318 825 336 54
2 240 397 26 152 266 216 236 501 476 564 326 463 74
6 776 468 656 181 380 873 334 946 346 308 532 213

Sorted array: 20 26 45 62 63 79 85 98 99 122 130 13
0 133 152 174 181 183 191 213 215 216 225 228 229 2
36 236 239 240 259 266 273 303 308 308 316 318 326
328 334 336 346 353 366 367 380 397 404 406 411 457
463 468 474 476 484 491 498 501 504 518 532 542 56
4 581 586 598 654 656 664 692 707 718 728 737 746 7
46 775 776 799 802 818 821 821 825 842 866 868 873
892 905 909 909 910 913 913 946 956 975 987 990

...Program finished with exit code 0
Press ENTER to exit console.

```

Observations

The program sorts the declared array with the Merge Sort method. It first displays the unsorted elements and then displays them sorted.

Table 8 - 3. Merge Sort Algorithm

Code + Console Screenshot

```

#include <iostream>
#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; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

void quicksort(int A[], int low, int high) {
    if (low < high) {
        int pivot = partition(A, low, high);
        quicksort(A, low, pivot - 1);
        quicksort(A, pivot + 1, high);
    }
}

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

```

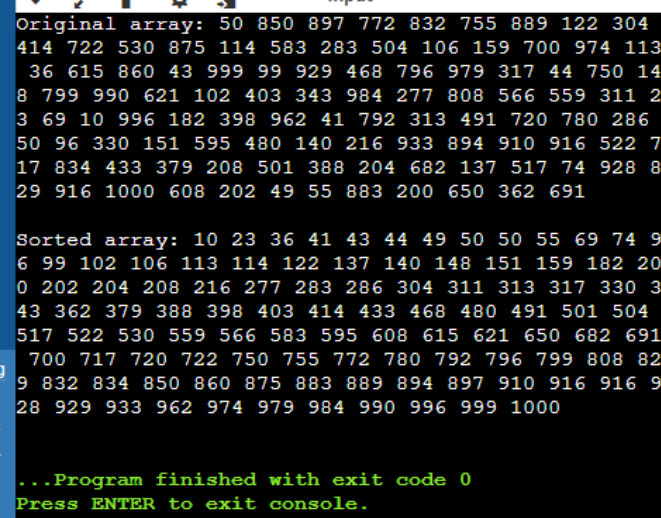
```
srand(static_cast<unsigned int>(time(0)));

for (int i = 0; i < size; ++i) {
    array[i] = rand() % 1000 + 1;
}
cout << "Original array: ";
for (int i = 0; i < size; ++i) {
    cout << array[i] << " ";
}
cout << "\n\n";

quicksort(array, 0, size - 1);

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

return 0;
}
```



The screenshot shows a terminal window with a black background and white text. It displays the output of the Quick Sort program. The first part shows the 'Original array' with 50 numbers. The second part shows the 'Sorted array' with the same 50 numbers in ascending order. At the bottom, it says '...Program finished with exit code 0' and 'Press ENTER to exit console.'

Observations	The program sorts the declared array with the Quick Sort method. It first displays the unsorted elements and then displays them sorted.
--------------	---

Table 8 - 4. Quick Sort Algorithm

7. Supplementary Activity

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, the left and right sub list from the partition method in quick sort can be sorted with other sort method, such as the shell method. An example of that is the image below, where it is applied in the shell sort.

```
1 #include <iostream>
2 using namespace std;
3
4 void shellSort(int arr[], int size) {
5     for (int gap = size / 2; gap > 0; gap /= 2) {
6         for (int i = gap; i < size; i++) {
7             int temp = arr[i];
8             int j;
9             for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {
10                 arr[j] = arr[j - gap];
11             }
12             arr[j] = temp;
13         }
14     }
15 }
16
17 int partition(int arr[], int low, int high) {
18     int pivot = arr[high];
19     int i = low - 1;
20
21     for (int j = low; j < high; j++) {
22         if (arr[j] < pivot) {
23             i++;
24             swap(arr[i], arr[j]);
25         }
26     }
27     swap(arr[i + 1], arr[high]);
28     return i + 1;
29 }
30
31 void quickSort(int arr[], int low, int high) {
32     if (low < high) {
33         int pivotIndex = partition(arr, low, high);
34         int leftSize = pivotIndex - low;
35         int rightSize = high - pivotIndex;
36         int left[leftSize];
37         int right[rightSize];
38
39         for (int i = 0; i < leftSize; i++) {
40             left[i] = arr[low + i];
41         }
42         for (int i = 0; i < rightSize; i++) {
43             right[i] = arr[pivotIndex + 1 + i];
44         }
45         shellSort(left, leftSize);
46         shellSort(right, rightSize);
47         for (int i = 0; i < leftSize; i++) {
48             arr[low + i] = left[i];
49         }
50         arr[pivotIndex] = arr[pivotIndex];
51         for (int i = 0; i < rightSize; i++) {
52             arr[pivotIndex + 1 + i] = right[i];
53         }
54     }
55 }
56
57 int main() {
58     int array[] = {5, 4, 3, 2, 1, 6, 7, 8, 9, 10};
59     int size = sizeof(array) / sizeof(array[0]);
60
61     cout << "Original Array: ";
62     for (int num : array) {
63         cout << num << " ";
64     }
65     cout << endl;
66
67     quickSort(array, 0, size - 1);
68
69     cout << "Sorted Array: ";
70     for (int num : array) {
71         cout << num << " ";
72     }
73     cout << endl;
74     return 0;
75 }
```

```
Original Array: 5 4 3 2 1 6 7 8 9 10
Sorted Array: 1 2 3 4 5 6 7 8 9 10
...Program finished with exit code 0
Press ENTER to exit console.
```

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 $O(N \cdot \log N)$ for their time complexity?

- For the array, Quick Sort will give the fastest time performance due to its average case efficiency performance. The merge sort and quick sort have $O(N \cdot \log N)$ for their time complexity, because merge sort divides the array into halves, sort it, and then merge it back, while quick sort partitions the array around a pivot and sort the array.

8. Conclusion

To conclude, through the laboratory activity that I have done the shell, merge, and quick sort method was demonstrated that it can be applied to an array. The merge sort splits the list into halves, sorts them, and merges them back together. And the shell sort is that sorts elements that are not in order and is sorted later on. Then the quick sort picks a partition to

divide the list and sorts them. In the supplementary activity, I have experimented if the partition part of the quick sort can be utilized to other sort method, and it could be done.

9. Assessment Rubric

I affirm that I will not give or receive any unauthorized help on this activity/exam and that all work will be my own.