


Activity No. 8	
SORTING ALGORITHMS: 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): ROALLOS, Jean Gabriel Vincent G.	Instructor: Prof. Maria Rizette Sayo
6. Output	
Code + Screenshot	 <pre>main.cpp 1 #include <iostream> 2 #include <cstdlib> 3 #include <time.h> 4 using namespace std; 5 6 const int max_size = 100; 7 int dataset[max_size]; 8 9 int main() 10 { 11 srand(time(0)); 12 for (int i = 0; i < max_size; i++) 13 { 14 dataset[i] = rand() % 100; 15 } 16 17 for (int i = 0; i < max_size; i++) 18 { 19 cout << dataset[i] << " "; 20 } 21 22 return 0; 23 } 24</pre> <p>input</p> <p>49 32 10 43 66 56 13 39 1 5 97 73 34 9 11 25 63 34 9 80 97 81 36 73 5 19 5 31 3 7 52 4 39 14 99 58 23 13 97 24 18 46 49 4 8 60 81 71 47 91 51 96 72 87 70 29 58 27 12 13 34 65 18 74 31 17 32 54 30 81 78 0 28 27 5 36 40 38 59 39 29 10 35 53 49 57 34 59 84 47 72 71 64 90 45 95 8 29 2 90</p> <p>...Program finished with exit code 0 Press ENTER to exit console.</p>
Observations	Used rand() function to randomize values to assign in an array. Values are limited to random numbers under 100. 2nd for loop prints out the result of the randomized assignment of values.

Code +
Screenshot

```
main.cpp sorting-algo.h :
4  #include "sorting-algo.h"
5  using namespace std;
6
7  const int max_size = 100;
8  int dataset[max_size];
9
10 int main()
11 {
12     srand(time(0));
13     for (int i = 0; i < max_size; i++)
14     {
15         dataset[i] = rand() % 100;
16     }
17
18     for (int i = 0; i < max_size; i++)
19     {
20         cout << dataset[i] << " ";
21     }
22
23     shellSort(dataset, max_size);
24     //mergeSort(dataset, 0, max_size - 1);
25     //quickSort(dataset, 0, max_size - 1);
26
27     size_t arrSize = sizeof(dataset) / sizeof(dataset[0]);
28
29     cout << "\n\nSorted: " << endl;
30     for (size_t i = 0; i < arrSize; i++)
31     {
32         cout << dataset[i] << " ";
33     }
34
35     return 0;
36 }
37
```

input

86 55 96 11 35 10 42 34 54 63 7 46 53 45 5 12 15 40 21 38 89 75 27 19 39 45 28 15 16 92 83 2 99 3
1 14 35 93 8 21 99 71 29 45 25 26 2 37 93 43 10 31 84 37 11 3 77 8 84 44 24 76 79 27 27 10 93 62
3 1 36 2 72 17 48 49 43 2 38 36 45 1 20 30 38 83 33 15 91 69 60 68 97 39 95 25 2 88 39 5 41

Sorted:
1 1 2 2 2 2 2 3 3 5 5 7 8 8 10 10 10 11 11 12 14 15 15 15 16 17 19 20 21 21 24 25 25 26 27 27 27
28 29 30 31 31 33 34 35 35 36 36 37 37 38 38 38 39 39 39 40 41 42 43 43 44 45 45 45 45 46 48 49 5
3 54 55 60 62 63 68 69 71 72 75 76 77 79 83 83 84 84 86 88 89 91 92 93 93 93 95 96 97 99 99

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

Observations

shellSort function from the sorting-algo.h header works properly and has sorted the randomized values within the array.

Code +
Screenshot

```
main.cpp  sorting-algo.h
51
52
53 while (j < Half2) {
54     arr[k] = rightArr[j];
55     j++;
56     k++;
57 }
58 return;
59 }
60
61 void mergeSort(int Arr[], int left, int right) {
62     if (left < right) {
63         int middle = left + (right - left) / 2;
64
65         mergeSort(Arr, left, middle);
66         mergeSort(Arr, middle + 1, right);
67
68         merge(Arr, left, middle, right);
69     }
70     return;
71 }
72
```

input

56 20 21 15 20 61 94 36 14 60 58 34 44 30 82 79 14 27 86 39 49 64 21 65 2 60 90 20 12 90 67 68 11 8
8 36 83 50 30 72 16 91 30 51 87 60 85 66 26 12 5 66 14 21 87 79 24 48 21 96 60 12 15 80 75 4 16 10
6 99 82 22 90 13 25 29 25 62 48 52 75 53 18 41 74 57 72 50 5 94 46 17 6 62 50 81 18 66 43 76 17

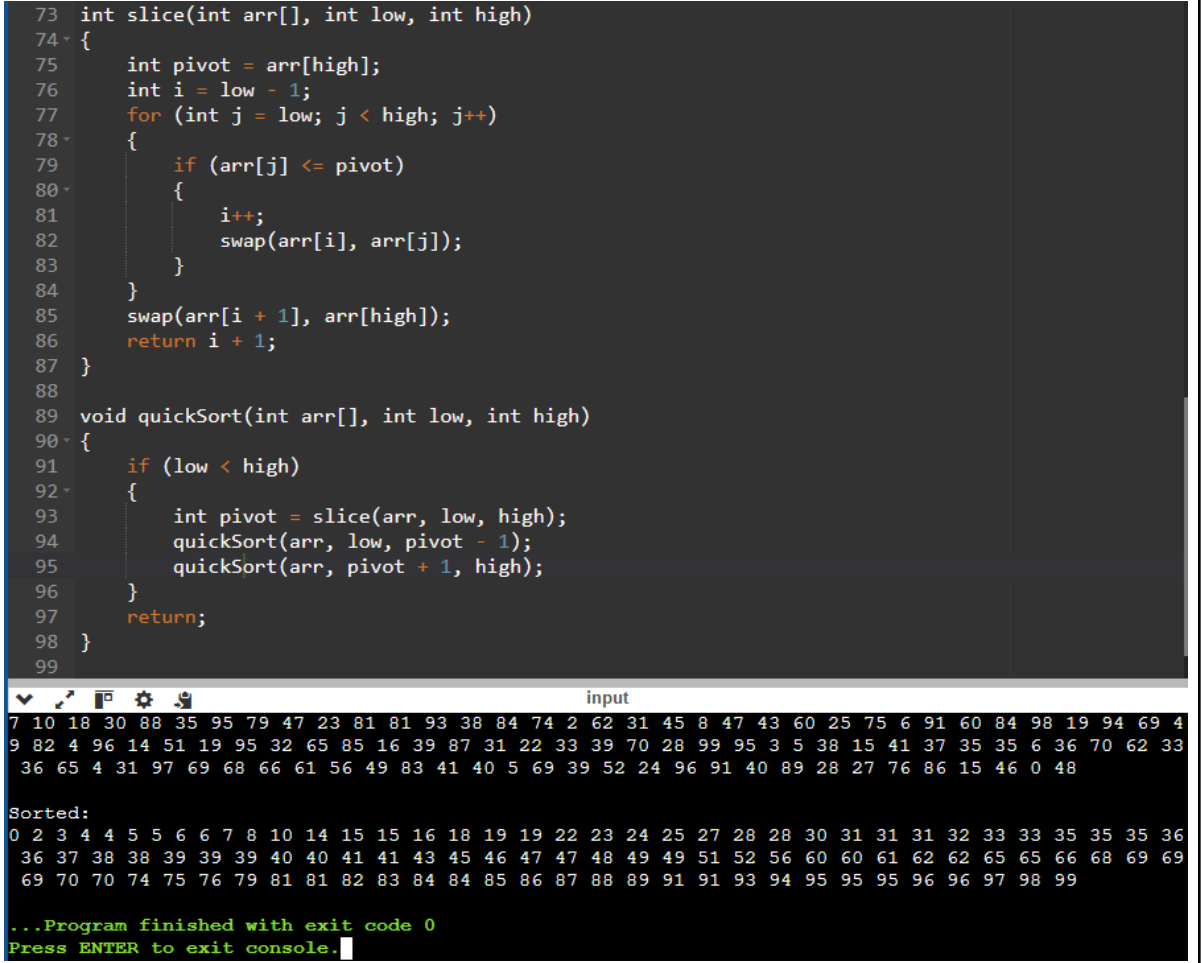
Sorted:

2 4 5 5 6 6 10 11 12 12 12 13 14 14 14 15 15 16 16 17 17 18 18 20 20 20 21 21 21 21 22 24 25 25 26
27 29 30 30 30 34 36 36 39 41 43 44 46 48 48 49 50 50 50 51 52 53 56 57 58 60 60 60 60 61 62 62 64
65 66 66 66 67 68 72 72 74 75 75 76 79 79 80 81 82 82 83 85 86 87 87 88 90 90 90 91 94 94 96 99

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

Observations

The sorting method of the merge sort takes a longer logic, I am yet to know if it is faster than the other two processes.

<p>Code + Screenshot</p>	 <pre> 73 int slice(int arr[], int low, int high) 74 { 75 int pivot = arr[high]; 76 int i = low - 1; 77 for (int j = low; j < high; j++) 78 { 79 if (arr[j] <= pivot) 80 { 81 i++; 82 swap(arr[i], arr[j]); 83 } 84 } 85 swap(arr[i + 1], arr[high]); 86 return i + 1; 87 } 88 89 void quickSort(int arr[], int low, int high) 90 { 91 if (low < high) 92 { 93 int pivot = slice(arr, low, high); 94 quickSort(arr, low, pivot - 1); 95 quickSort(arr, pivot + 1, high); 96 } 97 return; 98 } 99 input 7 10 18 30 88 35 95 79 47 23 81 81 93 38 84 74 2 62 31 45 8 47 43 60 25 75 6 91 60 84 98 19 94 69 4 9 82 4 96 14 51 19 95 32 65 85 16 39 87 31 22 33 39 70 28 99 95 3 5 38 15 41 37 35 35 6 36 70 62 33 36 65 4 31 97 69 68 66 61 56 49 83 41 40 5 69 39 52 24 96 91 40 89 28 27 76 86 15 46 0 48 Sorted: 0 2 3 4 4 5 5 6 6 7 8 10 14 15 15 16 18 19 19 22 23 24 25 27 28 28 30 31 31 31 32 33 33 35 35 35 36 36 37 38 38 39 39 39 40 40 41 41 43 45 46 47 47 48 49 49 51 52 56 60 60 61 62 62 65 65 66 68 69 69 69 70 70 74 75 76 79 81 81 82 83 84 84 85 86 87 88 89 91 91 93 94 95 95 95 96 96 97 98 99 ...Program finished with exit code 0 Press ENTER to exit console. </pre>
<p>Observations</p>	<p>Quick sort method utilizes an element as a pivot point of per partition</p>

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, we can sort both sublists made from the partition method using other sorting algorithms. The quick sort algorithm would identify a pivot. From that, we can sort the left and right sublists using bubble sort. After doing these and successfully sorting the elements in the sublists, these sorted sublists would then be joined again to the assigned pivot.

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

In my opinion, both are effective sorting methods for the given array, but I think the quicksort algorithm is marginally better than merge sort, despite having the same time-complexity level.

8. Conclusion

This and the previous lab activities have helped me understand the concepts and process of sorting algorithms. I was also enlightened about the various levels of efficiency in code processing. The common goal of creating code processes

is to accomplish the task fast while consuming less resources. Optimizing code processes would definitely help in a much larger setting, such as in a full-blown app or program.

9. Assessment Rubric