Assignment 2 INFR-2820 Report

Zachary D’Angelo - 100865977

INFR 2820U - Algorithms & Data Structures

March 24th, 2024

Introduction

The context for the sorting algorithms used in the assignment is that we have already learnt and used multiple sorting algorithms throughout the course. This assignment is now focusing on merge sorting, as well as quick sorting algorithms. The objective of this is to understand not only how to implement these algorithms in code, but also understanding how to perform these algorithms by hand. The detailed visualization will help us understand what actually occurs when using these sorting algorithms, as well as how each algorithm is executed.

Detailed Visualization

Merge Sort

The following below is the array that will be used for merge sort:

11	1	30	2	51	6	29	7	67	15	118	4	89	23

Firstly, we will count the number of elements that are in the array, and then divide by 2 to decide how we split the elements. In this case, there are 14 elements, therefore will divide 14/2, and get 7, meaning that the first split will have 7 elements each.

11	1	30	2	51	6	29

7	67	15	118	4	89	23

Now we evenly split the original array, and made two separate arrays. We will now split the array again. This time, because now the arrays are split and now have an odd number of elements in them, the first four elements will be split into a separate array from the last three elements in the array.

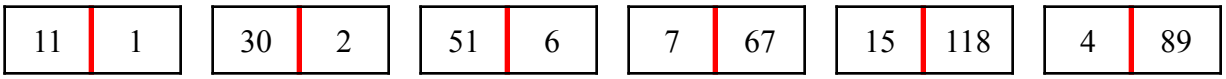
11	1	30	2

51	6	29

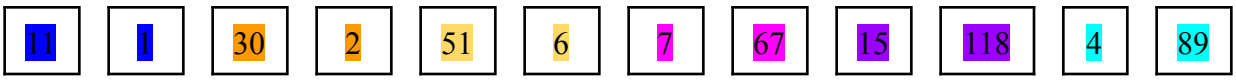
7	67	15	118

4	89	23

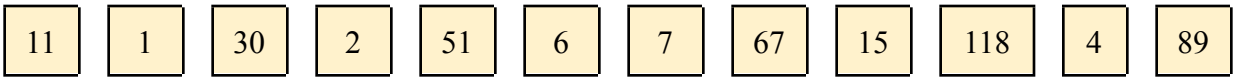
Once again, the array has been split, now leaving us with four different arrays. We will once again split the arrays. Array's 1 and 3 have an even amount of elements in them, with 4 elements. 4/2 is 2, therefore the elements will be split into arrays with 2 elements each. Array's 2 and 4 have an odd number of elements in them, with 3 elements. Therefore, I will take the first two elements in those arrays and split them with the last element, meaning that the last element of array's 2 and 4 are now done until we merge the arrays back together.

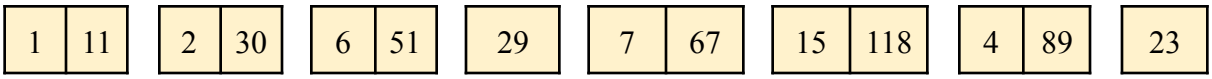


After splitting the array elements once again, we still have to split one more time before we start merging. All six arrays have an even number of elements in the array, with 2. 2/1 is 1, therefore each element is now split and now on its own.

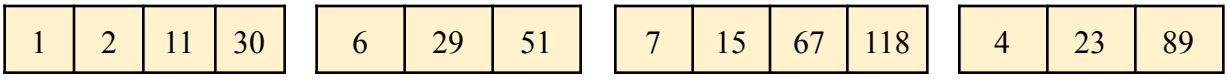


Now everything is split. We will now start merging the elements with one another. We will take the first two elements and merge them together, putting the lowest number at the front. We will do this for all the elements as well. (Note: The color in which the elements are highlighted will be merged together for this first iteration of merging)

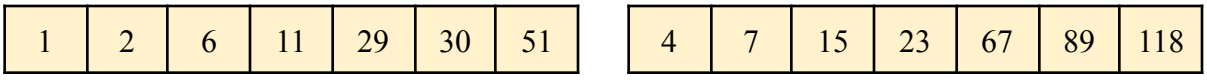




We have now completed the first iteration of merge sort. We will once again merge the first two arrays with one another, and sort them from smallest to largest number. We will do the same for all the arrays. (Note: When we split the arrays, because of an uneven number of elements, two elements became alone before the rest, therefore some arrays will only merge with arrays with one element rather than two)



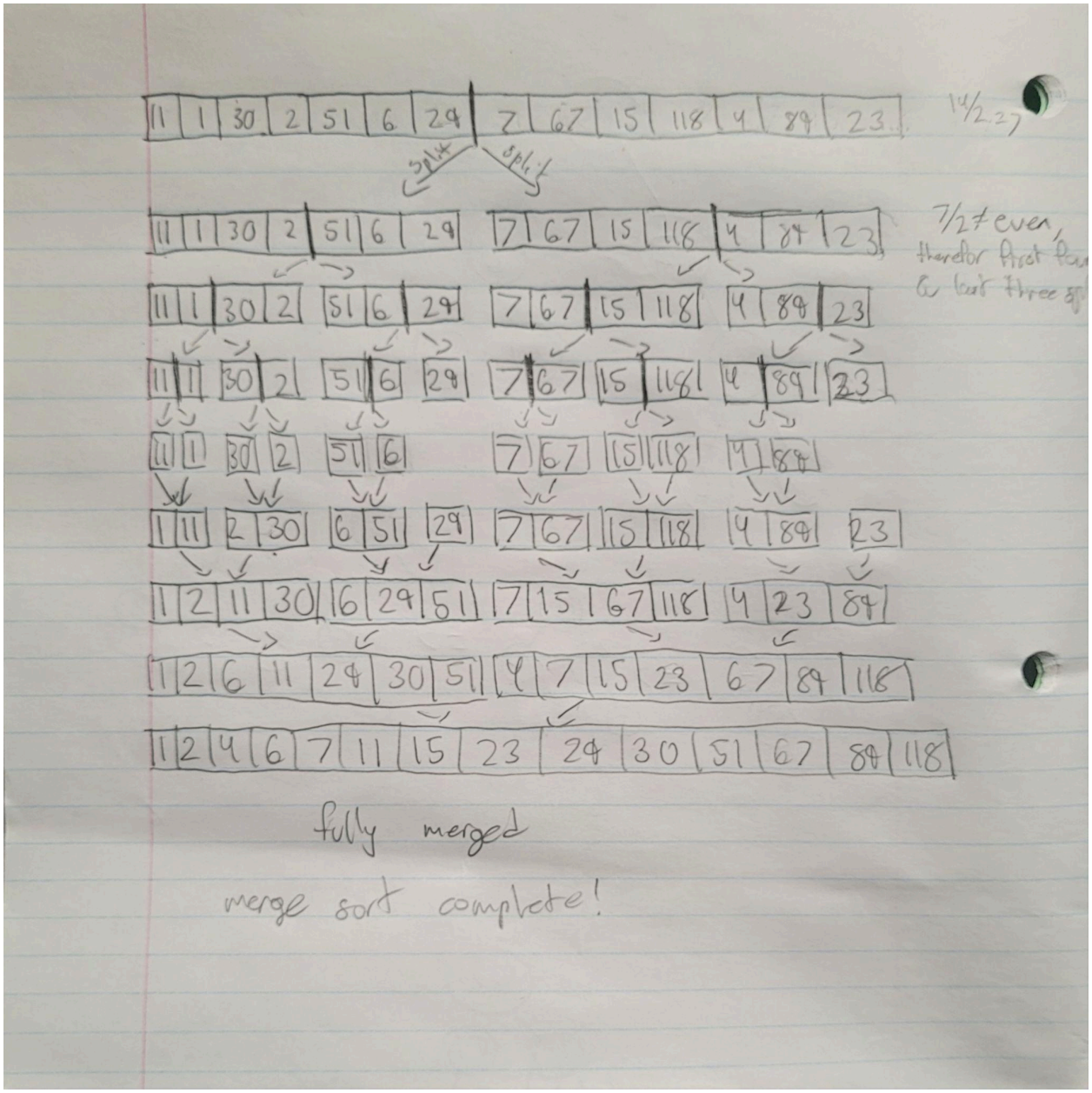
The second iteration of merging the arrays are now complete. Once again, we take the first two arrays and merge them with one another, and sort them just like the previous iterations.



The third iteration of merging the arrays is finished. The final iteration will merge the remaining two arrays together, and once again, sorts from smallest to greatest number.

1	2	4	6	7	11	15	23	29	30	51	67	89	118

With that, the elements are sorted and merge sort is complete. The following below shows all the iterations in one go to understand where everything split and merged.



Quick Sort

The following below is the array that we will be using for quick sort:

11	1	30	2	51	6	29	7	67	15	118	4	89	23

First, we pick the pivot. In this case, the pivot is already chosen in the assignment requirements, therefore we will use 11 as our pivot variable. Because our pivot is the very first element, we will compare it with the elements starting from the right side of the array. We will swap the pivot with the first element on the right side that is smaller than the pivot.

4	1	30	2	51	6	29	7	67	15	118	11	89	23

We compared the pivot, 11, with the very last element, 23. Because 23 is greater than 11, we look at the next element before 23. In this case, it is 89. Once again, the pivot is smaller than 89, so we check the next element. The next element is 4, which is smaller than our pivot, therefore, we swap the positions of our pivot, and 4. We will now compare the pivot with the numbers on the left side of the array. We will swap the pivot with a number closest on the left side and an element that is greater than the pivot.

4	1	11	2	51	6	29	7	67	15	118	30	89	23

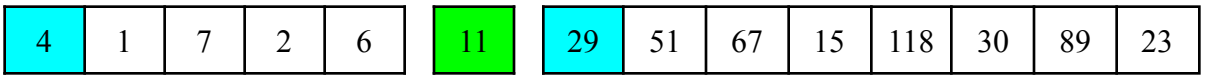
Since we swapped our pivot with 4, we do not need to compare these again, so we start our comparison with 1. We see that 1 is less than 30, so we leave that element alone. We then compare the pivot with 30. Because 30 is greater than 11, we swap both elements. We will repeat the same process until our pivot is in its correct position.

4	1	7	2	51	6	29	11	67	15	118	30	89	23

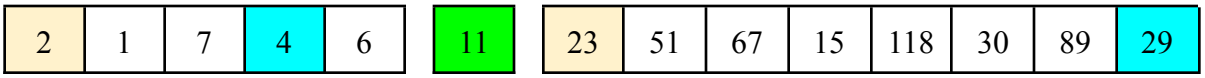
4	1	7	2	11	6	29	51	67	15	118	30	89	23

4	1	7	2	6	11	29	51	67	15	118	30	89	23

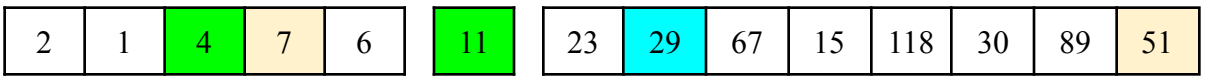
Our initial pivot is now in its correct position. We now have two sublists. We will make the first element of the two sublists our pivot, and perform the exact same operations as before.



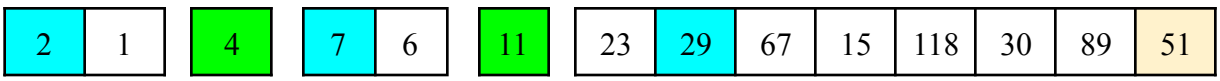
Once again, because the pivot is the first element of the sublists, we will compare them to the elements on the right side first, and will swap with the first element that is smaller than the pivot.

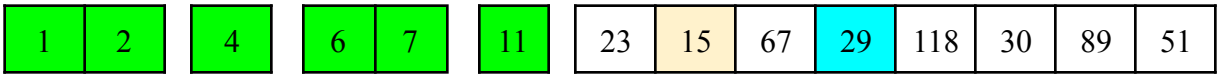


Now that the first iteration for these sublists are finished, we will now compare the pivot with elements on the left side of the array. Again, we will compare and swap with the first element nearest to the left that is a bigger element than the pivot.

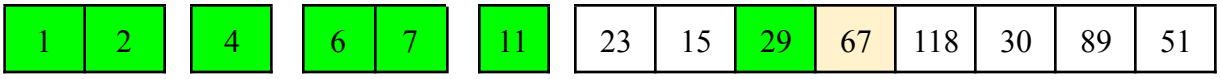


As we can see, the pivot in the first sublist is now in place, but the pivot in the second sublist is not. With the pivot of the first sublist in place, we now have an additional two sublists. We will make the first element of the additional sublists a pivot, and execute the same as we did previously. As for the original second sublist, we will proceed as before.



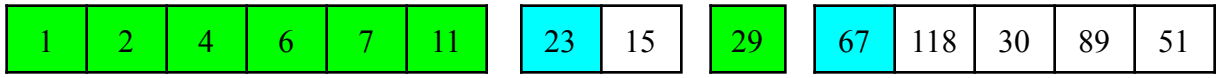


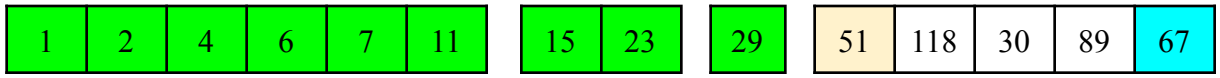
Now the original first sublist is now completely in order. The focus is now only on the second sublist. The pivot for the second sublist has also been placed in order, making a new additional two sublists for the second sublist. The pivots used will be once again the first element in the sublists, and we will use the same procedure as before.



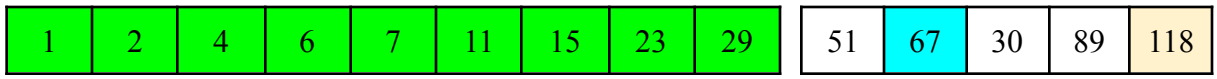
The whole left side of the original pivot is now sorted, and the pivot from our second sublist is also in order. We now have gotten an additional two sublists. We will once again use the first

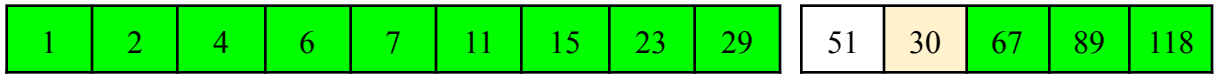
element in the two additional sublists as the pivot, and execute the algorithm the same way as before.



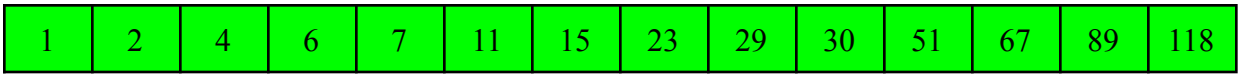


The first additional sublist that was created after element 29 when the pivot became sorted is now complete. We will continue to execute the other additional sublist using quick sort.

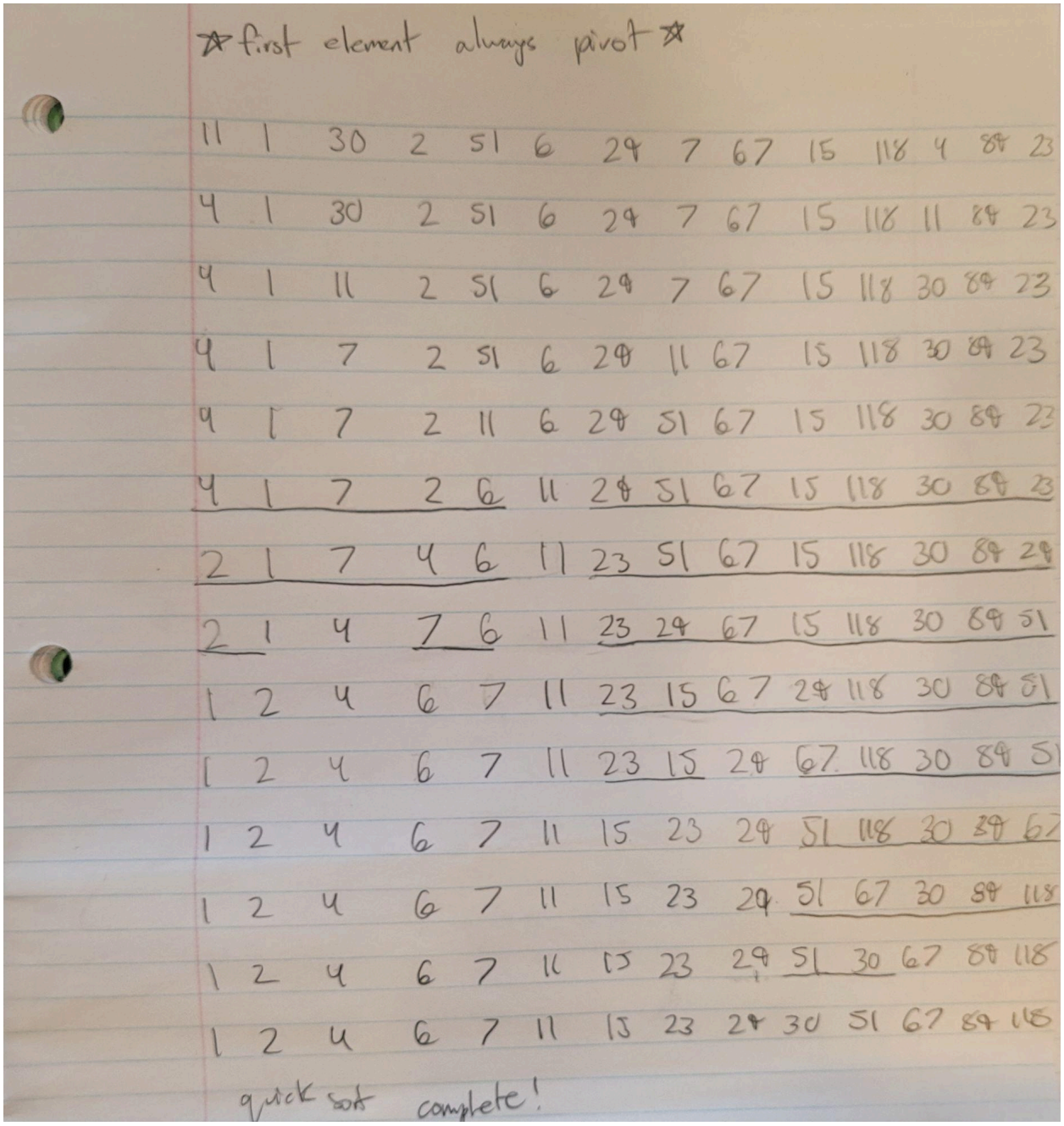




The pivot of the additional second sublist is now in order. As we can see, the last two elements after 67 are also in order, meaning that the only sublist left to sort are elements 51 and 30. We will use element 51 as our pivot and swap the elements to put them in order.



Everything is now sorted and the quick sort algorithm is now complete. The following shows all the iterations on a sheet of paper to make it quicker and easier to understand all the iterations and what has occurred throughout the process of this sort.



Comparison

	Best-Case Time Complexity	Worst-Case Time Complexity
Merge Sort	$\Omega(n \log(n))$	$O(n \log(n))$
Quick Sort	$\Omega(n \log(n))$	$O(n^2)$

Code Implementation

https://github.com/Zach21591/DSA---Assignment-2

```
Please enter the length of the array: 8
Please enter all the elements that will be in the oringal array (input a number, then press enter, input another number, enter, repeat):
9
1
8
2
7
3
6
4

The Original Array:
9, 1, 8, 2, 7, 3, 6, 4

Implementing the Merge Sort Algorithm:
Split: [9, 1, 8, 2, 7, 3, 6, 4]
Split: [7, 3, 6, 4]
Split: [6, 4]
4
Merge: [4, 6]
Split: [7, 3]
3
Merge: [3, 7]
3
3, 4
3, 4, 6
Merge: [3, 4, 6, 7]
Split: [9, 1, 8, 2]
Split: [8, 2]
2
Merge: [2, 8]
Split: [9, 1]
1
Merge: [1, 9]
1
1, 2
1, 2, 8
Merge: [1, 2, 8, 9]
1
1, 2
1, 2, 3
1, 2, 3, 4
1, 2, 3, 4, 6
1, 2, 3, 4, 6, 7
Merge: [1, 2, 3, 4, 6, 7, 8, 9]

The Sorted Array:
1, 2, 3, 4, 6, 7, 8, 9

Sorting Complete!
```

Conclusion

To summarize my findings and learning outcomes, I learnt to understand more sorting algorithms that can help us sort elements within an array or list. While doing the detailed visualization, I realized that visualizing merge sorting is much easier than visualizing quick sort. For me, I had an easier time doing merge sort over the quick sort algorithm.