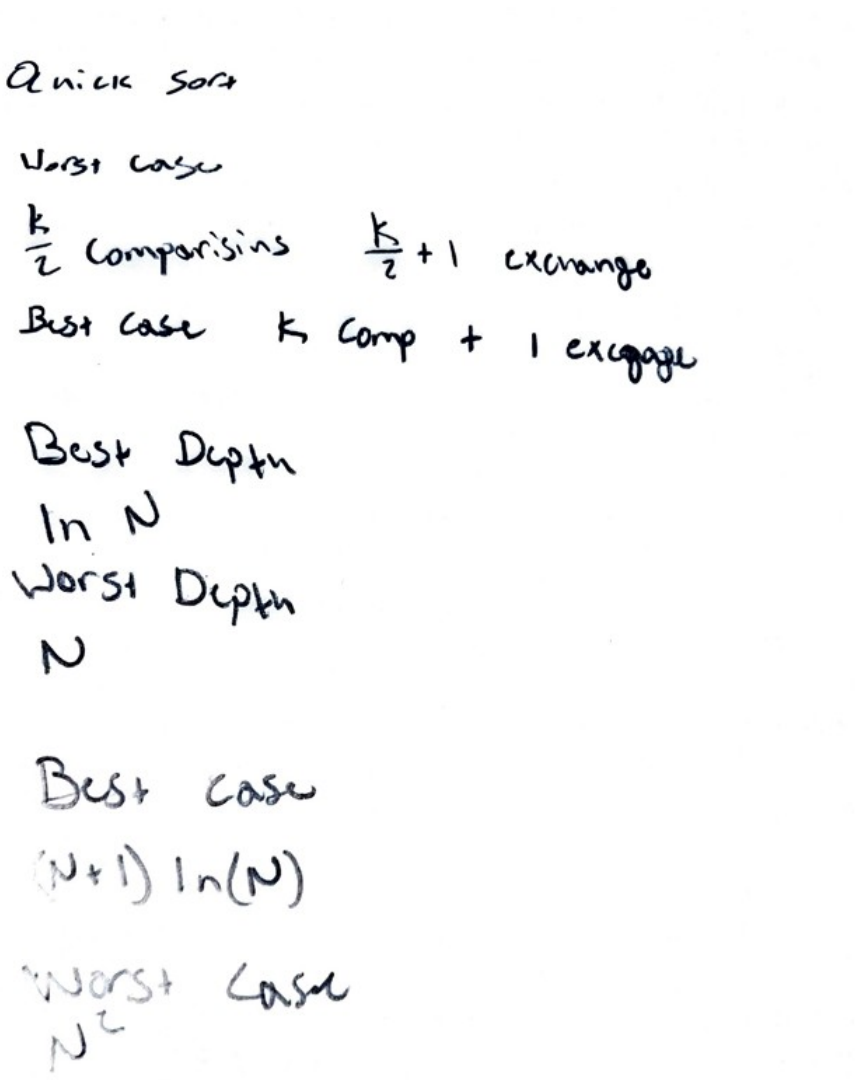
PA 3

David Greni and Haskell Cappers

Introduction

The two algorithms that we will analyze in this report are bubble sort and quick sort. David implemented quick sort and Haskell implemented bubble sort. Our hypothesis was that quick sort would be faster for both randomized and reverse order arrays, but slower for sorted arrays.

Complexity Analysis



Experimental Plan

We decided that our experiment would be to test both algorithms in three different ways. First being that we would test the algorithms with random data, second with already sorted data, and third with reverse sorted data. These tests were done three times with each algorithm. One with the array size being 2000, second with 4000, and third with 8000. We implemented these test by creating a test client that had methods to test the algorithms. First it would test the algorithms with known arrays to ensure the sorting algorithms were implemented correctly. Then they would call the experiment method that would send the large arrays through the algorithms.

Experimental Results

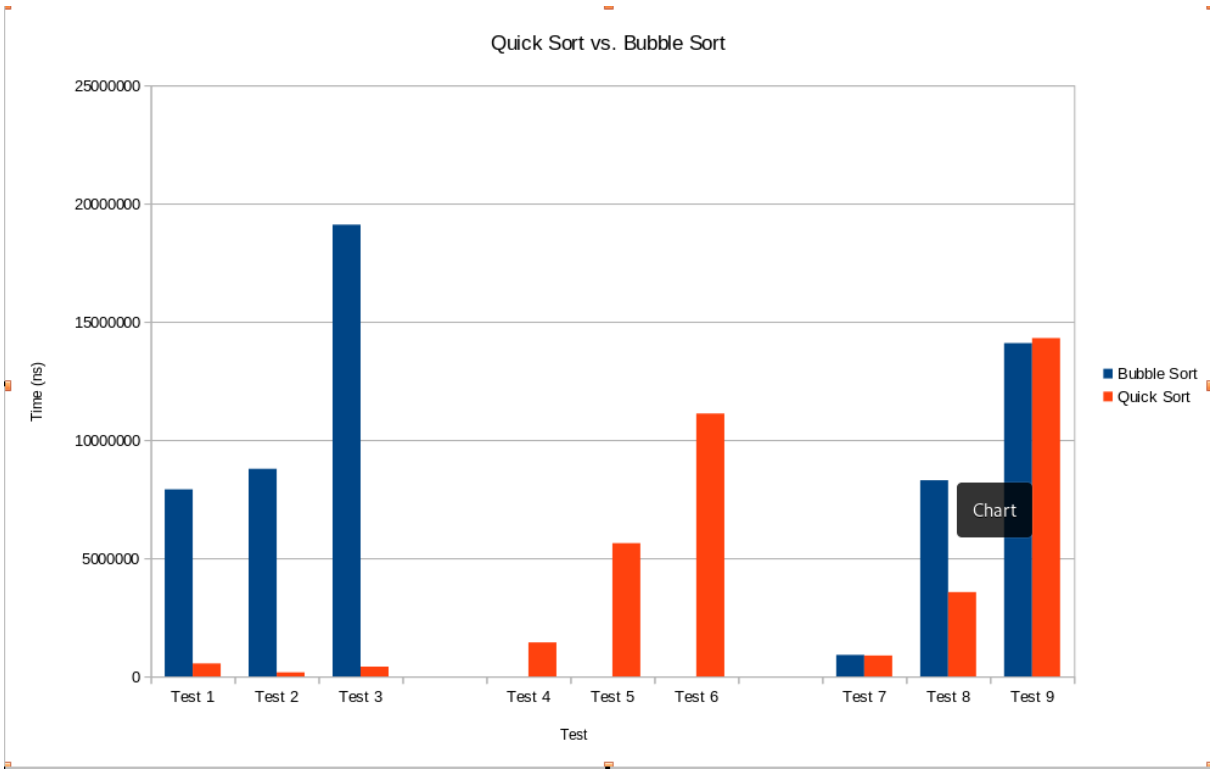


Figure 1: This graph depicts the time difference between quick sort and bubble sort.

	Bubble Sort	Quick Sort
Test 1	7908712	550120
Test 2	8781722	172790
Test 3	19106025	412120
Test 4	2020	1438251
Test 5	2620	5636182
Test 6	2450	11120403
Test 7	911181	882800
Test 8	8296932	3567901
Test 9	14098006	14309604

Table 1: This table represents the data that was collected from the program when running the data through the algorithms.

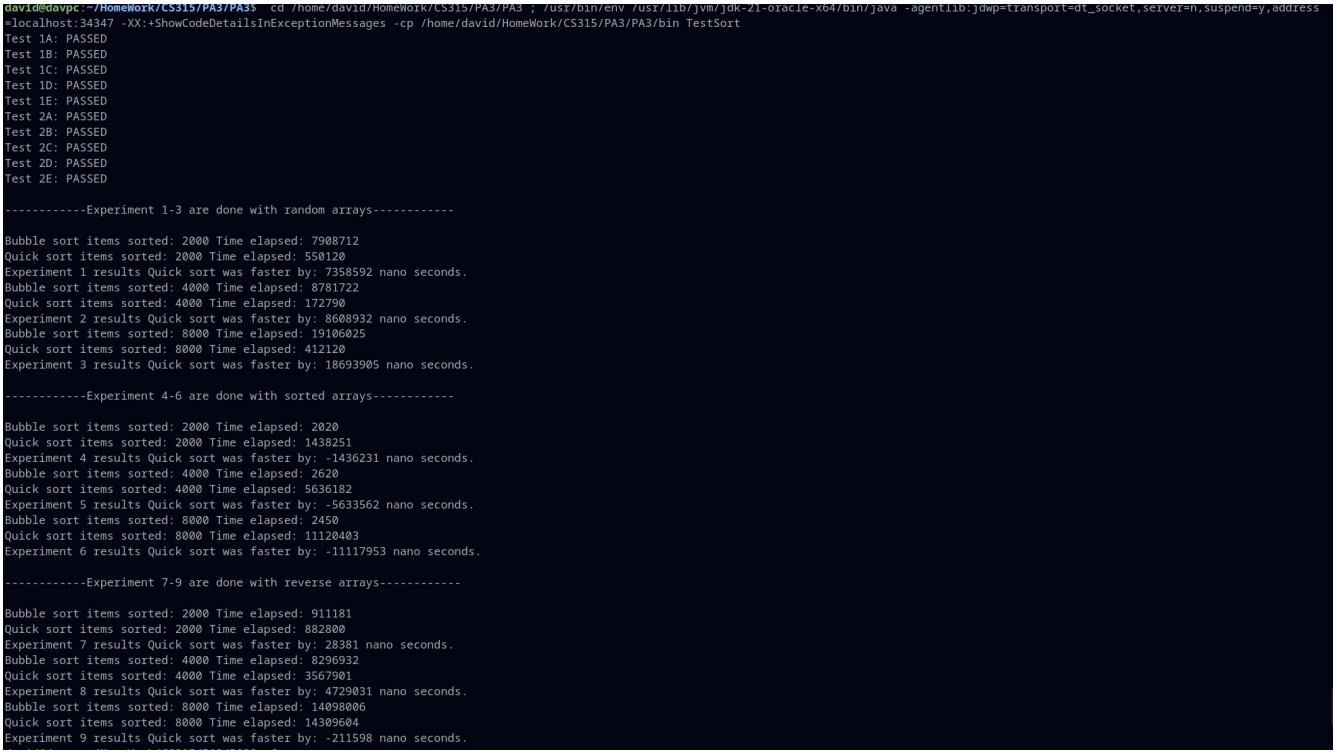


Image 1: This image is the raw output from the program.

Results Analysis

The results followed the hypothesis closely. In experiments 1-3 quick sort was much faster than bubble sort. This was because the data was randomized and there quick sort is much better at handling that data. Experiments 4-6 bubble sort was incredibly fast. This was because the data was already sorted and there was no need for the bubble sort to make any changes.

Experiments 7-9 the algorithms were close to taking the same amount of time. There were a few unexpected deviations from the hypothesis. First quick sort would unexpectedly get faster when it was handling an array of size 4000 over size 2000. This was unexpected because the array is double the size of the first one. This can be seen in experiments 2 and 8. Second the initial hypothesis was that quick sort would be much faster than bubble sort on reverse order arrays. This was proven false by the data.

Lessons Learned

Implementing the algorithms was very easy but ensuring that they were correct and implementing the test clients was difficult. Initially there were a few issues with the test client but that was quickly handled. Overall it was very interesting to learn how these algorithms handle these data sets and how efficient quick sort was at handling the large randomized data. It was also very interesting to see how quick sort worked when the worst case scenario was forced into quick sort with tests 4-6.

Overview of Source Code

```
public class QuickSort {

    //Swap helper function takes in an array and some indexes and swaps the values in that array
    public void swap(int[] nums, int i, int j){
        int temp = 0;
        temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }
```

This is the beginning of the quick sort algorithm. I implemented a helper function to assist the algorithm. This helper function takes in an array of integers, and two indexes. Then the function swaps the two values given in those indexes.

```java
//Partition the array
public int partition(int nums[], int low, int high){
    //Chose the pivot
    int pivot = nums[low];

    //Assign i
    int i = low;

    for(int j = low + 1; j <= high; j++){
        //Check if the current j is greater than the pivot
        if(nums[j] < pivot){
            //Swap the numbers
            i++;
            swap(nums, i, j);
        }

    }
    //Swap the pivot into the correct place
    swap(nums, low, i);
    return (i);
}
```

This is the partitioning function. It takes in a array of ints a high and a low. The pivot is decided based on the first value in that sub array. Then the I and J values are created to evaluate all values within that sub array and swap them when necessary. It then swaps the pivot value into the correct position after sorting the sub array.

```java
public void quickSort(int[] nums, int low, int high){
    //If the low and high are not the same
    if(low < high){

        //Make the new partition
        int part = partition(nums, low, high);
        //Recurse on the left and the right of the part
        quickSort(nums, low, part-1);
        quickSort(nums, part + 1, high);
    }
}
```

This is the quick sort algorithm. It takes in an array of ints a high and a low. It first partitions the array into two sub arrays then recursively calls on itself with the right and left sub array.

```java
//Bubble sort with random
public static long experiment1A(int[] array, BubbleSort b){
    //
    long start1 = System.nanoTime();

    int numOfItems = array.length;
    b.bSort(array, numOfItems);
    long end1 = System.nanoTime();
    long totalTime = end1 - start1;
    System.out.println("Bubble sort items sorted: " + numOfItems + " Time elapsed: "+ totalTime);
    return totalTime;
}

public static long experiment1B(int[] array, QuickSort q){
    //
    long start1 = System.nanoTime();

    int numOfItems = array.length;
    q.quickSort(array, low:0, numOfItems - 1);
    long end1 = System.nanoTime();
    long totalTime = end1 - start1;
    System.out.println("Quick sort items sorted: " + numOfItems + " Time elapsed: "+ totalTime);
    return totalTime;
}
```

These are the experiment functions. They are responsible for running the tests. They first start by getting the time at the beginning of the function. Then they call the sort algorithm. After the sort algorithm has finished they record the time. Then they output a println statement that tells

the user how many items were sorted and how long it took for each. It then returns the time for later use.

```java
public static void main(String[] args) {

    Random r = new Random();

    //Make array for tests
    int array1A[] = new int[2000];
    int array1B[] = new int[2000];
    int array2A[] = new int[4000];
    int array2B[] = new int[4000];
    int array3A[] = new int[8000];
    int array3B[] = new int[8000];

    int array4A[] = new int[2000];
    int array4B[] = new int[2000];
    int array5A[] = new int[4000];
    int array5B[] = new int[4000];
    int array6A[] = new int[8000];
    int array6B[] = new int[8000];

    int array7A[] = new int[2000];
    int array7B[] = new int[2000];
    int array8A[] = new int[4000];
    int array8B[] = new int[4000];
    int array9A[] = new int[8000];
    int array9B[] = new int[8000];


    //Fill array with random items
    for(int i = 0; i < array1A.length; i++){
        array1A[i] = array1B[i] = r.nextInt(bound:5000);
    }
    for(int i = 0; i < array2A.length; i++){
        array2A[i] = array2B[i] = r.nextInt(bound:5000);
    }
    for(int i = 0; i < array3A.length; i++){
        array3A[i] = array3B[i] = r.nextInt(bound:5000);
    }
    //Already sorted
    for(int i = 0; i < array4A.length; i++){
        array4A[i] = array4B[i] = i;
    }
    for(int i = 0; i < array5A.length; i++){
        array5A[i] = array5B[i] = i;
    }
    for(int i = 0; i < array6A.length; i++){
        array6A[i] = array6B[i] = i;
    }
    //Reverse order
```

This is the code that is responsible for running the tests. It first initializes all of the arrays to the correct length. Then it fills them with data through the for loops. The first three for loops fills

them with randomized data. The next three fill them with sorted data. The last three fill them with reverse sorted data.

```
System.out.println();
System.out.println(x:"------------Experiment 1-3 are done with random arrays------------");
System.out.println();
long time1 = experiment1A(array1A, b);
long time2 = experiment1B(array1B, q);
System.out.println("Experiment 1 results Quick sort was faster by: " + (time1 - time2) + " nano seconds.");
long time3 = experiment1A(array2A, b);
long time4 = experiment1B(array2B, q);
System.out.println("Experiment 2 results Quick sort was faster by: " + (time3 - time4) + " nano seconds.");
long time5 = experiment1A(array3A, b);
long time6 = experiment1B(array3B, q);
System.out.println("Experiment 3 results Quick sort was faster by: " + (time5 - time6) + " nano seconds.");

System.out.println();
System.out.println(x:"------------Experiment 4-6 are done with sorted arrays------------");
System.out.println();

long time7 = experiment1A(array4A, b);
long time8 = experiment1B(array4B, q);
System.out.println("Experiment 4 results Quick sort was faster by: " + (time7 - time8) + " nano seconds.");
long time9 = experiment1A(array5A, b);
long time10 = experiment1B(array5B, q);
System.out.println("Experiment 5 results Quick sort was faster by: " + (time9 - time10) + " nano seconds.");
long time11 = experiment1A(array6A, b);
long time12 = experiment1B(array6B, q);
System.out.println("Experiment 6 results Quick sort was faster by: " + (time11 - time12) + " nano seconds.");

System.out.println();
System.out.println(x:"------------Experiment 7-9 are done with reverse arrays------------");
System.out.println();

long time13 = experiment1A(array7A, b);
long time14 = experiment1B(array7B, q);
System.out.println("Experiment 7 results Quick sort was faster by: " + (time13 - time14) + " nano seconds.");
long time15 = experiment1A(array8A, b);
long time16 = experiment1B(array8B, q);
System.out.println("Experiment 8 results Quick sort was faster by: " + (time15 - time16) + " nano seconds.");
long time17 = experiment1A(array9A, b);
long time18 = experiment1B(array9B, q);
System.out.println("Experiment 9 results Quick sort was faster by: " + (time17 - time18) + " nano seconds.");
```

This is the running of the experiments. First there are a few println statements for readability then the experiments are ran and the time is saved for later use. Then the time difference is outputted to the user.

Conclusion

In conclusion this was a fun assignment and brought a lot of understanding to how these algorithms work. This also made me understand how to implement these algorithms. The hypothesis was mostly correct and I now understand why the hypothesis was incorrect.

References

[1] “Quicksort - data structure and algorithm tutorials,” GeeksforGeeks, https://www.geeksforgeeks.org/quick-sort/ (accessed Oct. 30, 2023).

[2] Tono NamTono Nam, “Order of growth of specific recursive function,” Stack Overflow, https://stackoverflow.com/questions/9509038/order-of-growth-of-specific-recursive-function (accessed Oct. 30, 2023).