HACETTEPE UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING
BBM204

NAME AND SURNAME: ŞEYMA CİVAN

IDENTİTY NUMBER: 21627078

SUBJECT : Programming Assignment 1
Analysis of Sorting Algorithms

Problem Definition: In this assignment, we are expected to analyze and compare the 4 different sorting algorithms by applying them to the arrays in different sizes and scenarios we have created before.

Solution Approach: First of all, I created arrays for different sizes of best, worst and average states to analyze my algorithms. We were asked testing the algorithm with the values of N such as 5.000, 10.000, 50.000, 100.000, 250.000 and 500.000 for best, average and worst case scenarios. But, with these N values, the arrays were not sorted correctly in the bitonic sorting algorithm. In order for the bitonic sorting algorithm to work correctly, I rounded the N value to the power of the 2. So the new N values are 4096, 16384, 65536, 131072, 262144 and 524288. After determining the N values, I thought about the best, worst, and average states for general sort algorithms.Then I tested all the algorithms with different N-element arrays for the best, worst and average scenarios, and noted the execution times.

Cocktail Shaker Sort Algorithm

Cocktail Sort is similar to Bubble sort. Unlike bubble sorting, Cocktail sorting correctly sorts by comparing through a given array in both directions. Firstly through the array from left to right, adjacent items are compared and if value on the left is greater than the value on the right, then values are swapped. Then through the array from opposite direction ,adjacent items are compared and if value on the left is greater than the value on the right, then values are swapped. This loop is repeated throughout the array.

Complexity:

Best Case: Time complexity is O(n). Because there is not swapped in the array which is already ordered from small to large.

Worst – Average Case: Time complexity is O(n*n) for both the worst case and the average case. Because the elements are swapped throughout the array.

The chart below includes the execution times in terms of nanoseconds of the Cocktail Shaker sort algorithm in different N values in best, worst and average cases.

	4096	**16384**	**65536**	**131072**	**262144**	**524288**
Best Case	*1356900*	*2163900*	*7312600*	*8279600*	*11005900*	*12904600*
Worst Case	175125900	499754300	48273127600	212271270400	876009392700	3906887207600
Average Case	213025500	874961000	29388662000	239104271800	316209351700	566009937600

Pigeonhole Sort Algorithm

Pigeonhole sorting is a sorting algorithm that is suitable for sorting lists of elements where the number of elements and the number of possible key values are approximately the same.
First, we find the largest and smallest number in the N element array. Then we calculate the range value (range = max-min-1).
We create an array of the same size as the range. We visit each element of the array and then put each element in new array.
An element E is put in new array at index E – min. We place all the elements in the new array according to their key values. Finally, we copy back the new array to orijinal array.

Complexity:

Best - Worst – Average Case: Time complexity is O(n + range) for all cases.
Likewise, the space complexity of the Pigeonhole sorting algorithm is O (n+ range) because we create a temporary array.
**Pigeonhole Sort works best when range is small, and becomes rather inefficient as range gets larger.

The chart below includes the execution times in terms of nanoseconds of the pigeonhole sort algorithm in different N values in best, worst and average cases.

	4096	**16384**	**65536**	**131072**	**262144**	**524288**
Best Case	*3077000*	*5055400*	*21735300*	*29262900*	*46825100*	*50653600*
Worst Case	3104500	4595200	26792100	30527000	34788300	48295500
Average Case	2610700	4574300	18192300	24245700	29372000	44796800

Bitonic Sort Algorithm

Firstly, we split the array into two equal parts. We sort the left half in ascending order, and the right half in descending order.We compare first element of first half with first element of second half, then second element of first half with second element of second and so on. We exchange elements if an element of first half is smaller. We recursive continue until the split array size is 1. Finally, we merge all sorted subarrays back together.

Complexity:

Best - Worst – Average Case: Time complexity of Bitonic Sort is O(Logn^2) in all 3 cases (worst, average and best). Bitonic sort always divides the array into two halves and take logarithmic time to form a sorted sequence of length n from two sorted sequences of length n/2.

** Best case for bitonic sort algorithm is the array that already sorted the left half in ascending order, and the right half in descending order.

The chart below includes the execution times in terms of nanoseconds of the bitonic sort algorithm in different N values in best, worst and average cases.

	4096	**16384**	**65536**	**131072**	**262144**	**524288**
Best Case	*13322000*	*25518900*	*96295700*	*259658600*	*553843700*	*1194787500*
Worst Case	15973800	24331700	84537900	251985300	518325600	1184876400
Average Case	13390100	21093600	76402700	206100200	417622300	1145887400

Comb Sort Algorithm

Comb Sort is similar to Bubble sort. Comb Sort is the implementation Bubble Sort by using gap of size more than 1. Initially, our gap value is equal to our array size N divide by 1,3. In the next steps, we reduce our gap value to not less than 1. It means that after completing each phase the gap is divided by 1,3.

Complexity:

Best Case: Time complexity is O(n) when the array is already sorted, in which case swapped is false.

Average – Worst Case: Time Complexity is O(n^2), in this case the swapped is true, and the loop continues as long as the gap value is greater than 1.

** Space complexity of the comb sort algorithm is O(1).

The chart below includes the execution times in terms of nanoseconds of the comb sort algorithm in different N values in best, worst and average cases.

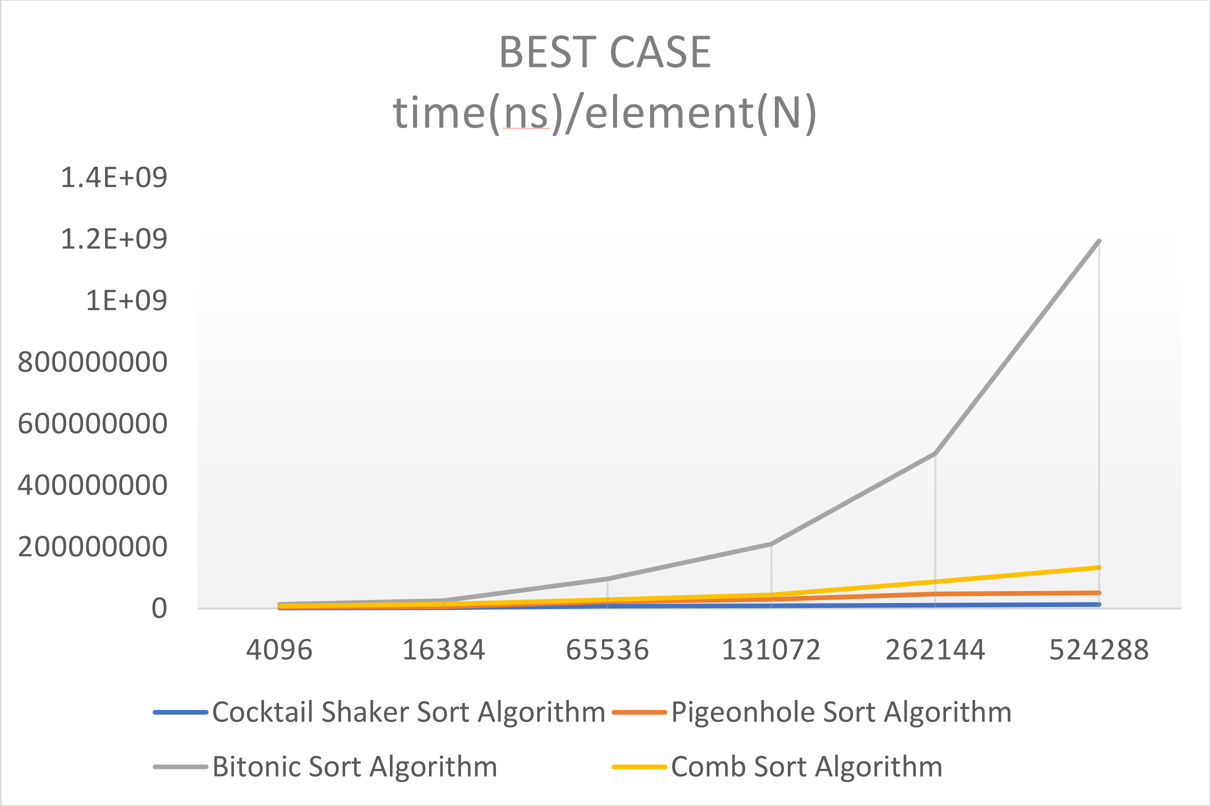
	4096	**16384**	**65536**	**131072**	**262144**	**524288**
Best Case	*7916800*	*13631900*	*27910700*	*44123300*	*86088200*	*132800300*
Worst Case	9827700	18510700	40547200	80509200	141514100	261361100
Average Case	8890500	15250600	32246700	71314700	95313100	158911900

******** Since I could not find the common best, average and worst situation for the 4 different algorithms given, I wrote below the best, average and worst case scenarios considering the general sorting algorithms.

Best Case Scenario

The best situation for an N-element array that we want to rank from small to large; that the array is already sorted. So the best case arrays contain consecutive numbers from 0 to N.

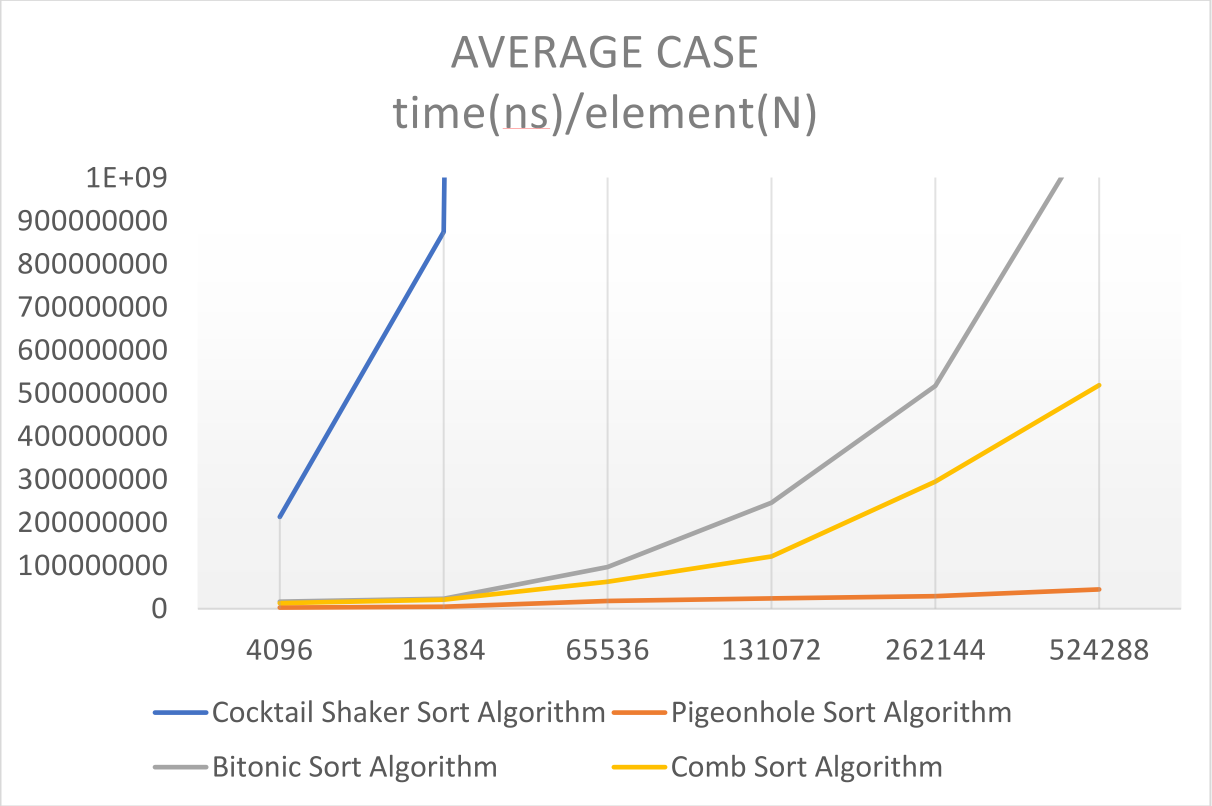
As seen in the graph below, the fastest running algorithm in the best case arrays is the Cocktail Shaker Sort. The reason for this is that there is no swapped throughout the array because the array is already in order. Best case time complexity of Cocktail Shaker Sort is O(n), the algorithm speed will increase as the number of N decreases. The slowest algorithm in the best case arrays is the Bitonic Sort. For the bitonic sort to run fastest, the left half of the array must be in ascending order and the right half in descending order.



Average Case Scenario

The average situation of a arrays of N-elements that we want to rank from small to large; array is made up of random numbers. Average case arrays consist of random numbers between 0 and N. To calculate the average, I created 3 different sequences for each N number. To get more accurate results in the average scenario, I tested all algorithms with 3 different N-element arrays in each situation. I calculated the average of 3 different numbers for each case. I noted the average value obtained as the execution time.

As seen in the graph below, the slowest running algorithm in the average case arrays is the Cocktail Shaker Sort. The elements are swapped throughout the array. In this case, the time complexity of the Cocktail Shaker Sort is O (n^2). Time complexity of the pigeonhole sort is O(n+range) in average case. Since the range of the average case arrays is N, the fastest running algorithm is the pigeonhole sort algorithm.



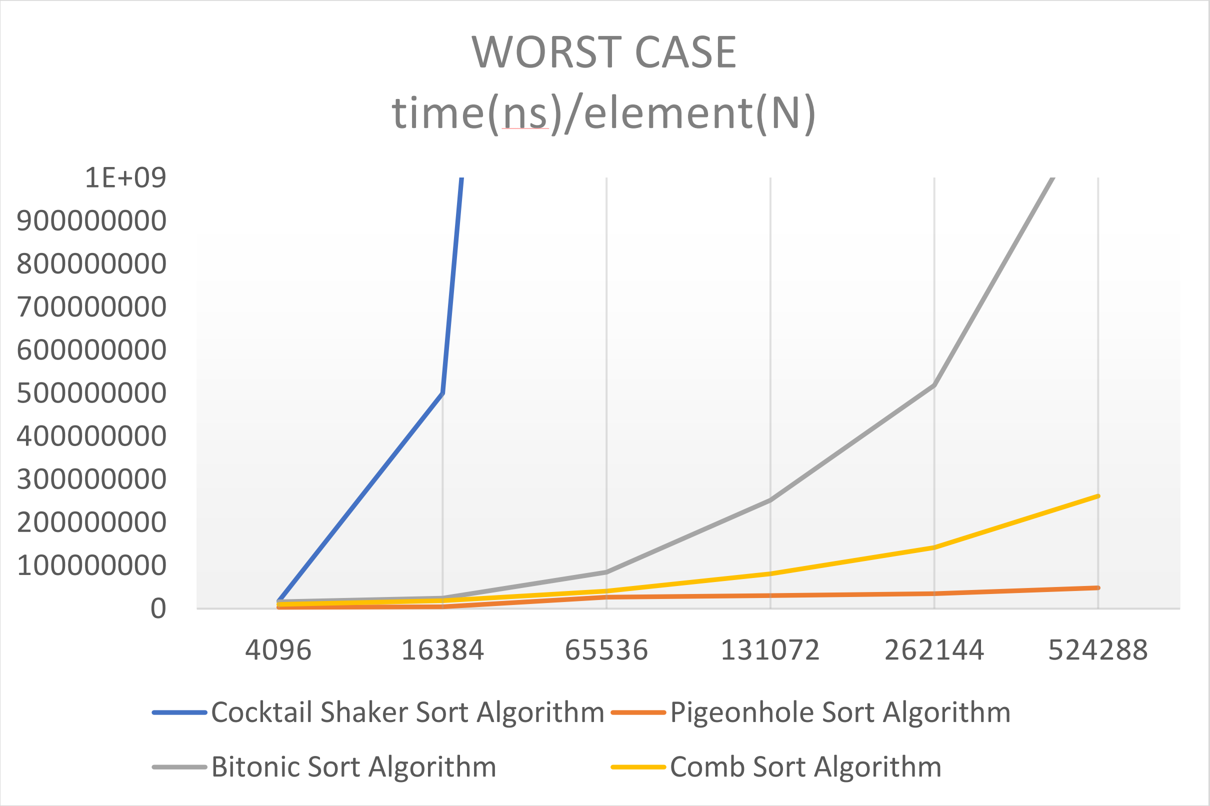
Worst Case Scenario

The worst situation for an N-element array that we want to rank from small to large; on the contrary, the array is ordered from large to small because I couldn't find the worst common situation for the 4 algorithms given to us. So the worst case arrays contain consecutive numbers from N to 0.

As seen in the graph below, the slowest running algorithm in the worst case is the Cocktail Shaker Sort, as in the average case scenario. The elements are swapped throughout the array. In this case, the time complexity of the Cocktail Shaker Sort is O (n^2).
We said that, in the worst case, the time complexity of the comb Sort is O(n^2). But the graph shows that it runs faster than O (n^2). This is because the comb sort algorithm compares the elements to a certain "gap" ratio.
In the worst case, the time complexity of the Bitonic Sort is O(logn^2).

Which of the algorithm(s) you deal with behave(s) fine even on worst case scenario?

As can be seen from the graphic, the fastest algorithm is pigeonhole sort. The time complexity of the pigeonhole sort algorithm in this case is O(n + range).Pigeonhole Sort works best when range is small, and becomes rather inefficient as range gets larger.



Which of the algorithm(s) you deal with behave(s) badly on average and worst case scenarios?

In the worst and average case, the algorithm that behave badly is the Cocktail Shaker Sort.The elements are swapped throughout the array. In this case, the time complexity of the Cocktail Shaker Sort is O (n^2).

References:

https://en.wikipedia.org/wiki/Cocktail_shaker_sort
https://en.wikipedia.org/wiki/Pigeonhole_sort
https://en.wikipedia.org/wiki/Bitonic_sorter
https://en.wikipedia.org/wiki/Comb_sort