FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

CME 2204

Assignment 1

Comparing Shell Sort and Dual Pivot Quick Sort

by

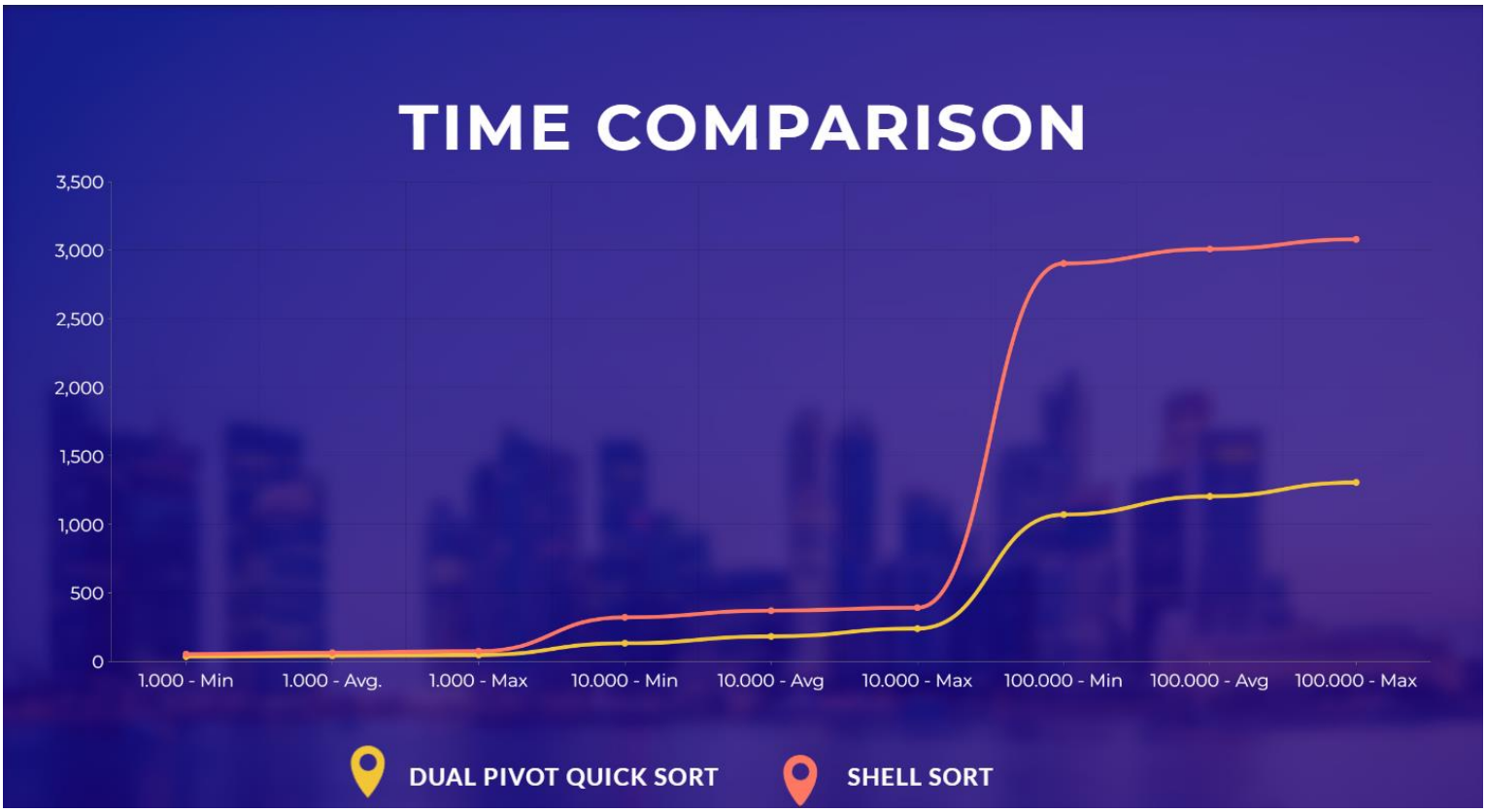
2021510010 - Çağrı AYDIN

Runtime Analysis:

	EQUAL INTEGERS			RANDOM INTEGERS			INCREASING INTEGERS			DECREASING INTEGERS		
	1.000	10.000	100.000	1.000	10.000	100.000	1.000	10.000	100.000	1.000	10.000	100.000
dualPivot QuickSor	521.400 ns	1.639.500 ns	9.977.300 ns	433.550 ns	1.838.010 ns	12.048.040 ns	3.342.400 ns	17.773.600 ns	693.821.200 ns	3.444.800 ns	16.372.600 ns	660.739.400 ns
shellSort	244.900 ns	2.375.500 ns	22.507.900 ns	655.390 ns	3.717.720 ns	27.325.710 ns	265.000 ns	1.866.200 ns	7.504.800 ns	501.100 ns	2.364.200 ns	8.539.700 ns

- In this analysis, I used AMD Ryzen 7 5800H processor and 16GB of ram. These times will be different in any other processors.
- For random integer analysis I run my code ten times for each case and used the same array in both sorting algorithms for getting better comparison results. Results of random integers on the table are the mean of runtimes.

- Here is a graph comparison of my finding for random integer values:



Overall Assessment of Two Sorting Algorithms

As we can see on the run time analysis Dual Pivot Quick Sort outperforms Shell Sort in randomly selected values and equal values when our data sets size getting larger. Whenever we are dealing with totally random integers which is more likely to happen in real time data sets, we should consider using Dual Pivot Quick Sort over Shell Sort for better sorting time. For other cases if we know our data is in both increasing or decreasing order, we should use Shell's Sorting algorithm for better performance and run time.

Time Complexities

Dual Pivot Quick Sort

```
4 usages
public void dualPivotQuickSort(int[] arrayToSort, int low, int high) {
    if (low < high) {
        int[] piv = partition(arrayToSort, low, high);                    //c1
        dualPivotQuickSort(arrayToSort, low, high: piv[0] - 1);           //c2
        dualPivotQuickSort(arrayToSort, low: piv[0] + 1, high: piv[1] - 1); //c3
        dualPivotQuickSort(arrayToSort, low: piv[1] + 1, high);           //c4
    }
}
```

Partition Function:

```
while (k <= g) {                                //k1
    if (array[k] < p) {
        exchange(array, k, j);
        j++;
    } else if (array[k] >= q) {
        while (array[g] > q && k < g) {         //k2
            g--;
        }
        exchange(array, k, g);
        g--;

        if (array[k] < p) {
            exchange(array, k, j);
            j++;
        }
    }
    k++;
}
```

c_1's complexity is $k_1.n + \sum_{i=0}^{n} n.k_2$ which is $O(n^2)$

$T(n) = c_1.O(n^2) + c_2.T(n/3) + c_3.T(n/3) + c_4.T(n/3)$

$T(n) = 3T(n/3) + O(n^2)$

According to Master Theorem:

a = 3 and b = 3 so;

$log_a b = log_3 3 = 1$ → $n^1 log_3 3 + \varepsilon = 2$
$f(n)=n^2$

Master Theorem

$3f(n/3) < c.f(n)$ → $3((n^2)/9) = n^2/3$ → $n^2/3 < c.n^2$ (Case 3 in Master Theorem because $n < n^2$)

So $T(n) => \Theta(f(n)) -> \Theta(n^2)$

$T(n) = T(f(n)) = T(O(n^2)) = \Theta(n^2)$

Shell Sort

```
1 usage
public void shellSort (int[] arrayToSort) {
    for (int gap = arrayToSort.length/2;gap>0;gap/=2){                  //c1
        for (int i = gap;i<arrayToSort.length;i++){                     //c2
            int j =i;                                                   //c3
            while (j>=gap && arrayToSort[j]<(arrayToSort[j-gap])){      //c4
                int temp = arrayToSort[j];                              //c5
                arrayToSort[j] = arrayToSort[j-gap];                    //c6
                arrayToSort[j-gap] = temp;                              //c7
                j-=gap;                                                 //c8
            }
        }
    }
}
```

$T(n) = c_1.(log_2 n + 1) + c_2.\sum_{i=\frac{n}{2}}^{1}(n+1) + c_3.\sum_{i=\frac{n}{2}}^{1} n + c_4.\sum_{i=\frac{n}{2}}^{1}\sum_{i=0}^{n}(log_2 n + 1) + (c_5 + c_6 + c_7 + c_8).\sum_{i=\frac{n}{2}}^{1}\sum_{i=0}^{n}(log_2 n)$

When we simplify our function,

$T(n) = a.(log_2 n) + b.(n.log_2 n) + c.(n.(log_2 n)^2)$

Worst Case Scenario, Big O Notation (O)

$O(n.(log_2 n)^2)$

Average Case Scenario, Big Teta Notation(Θ)

$\Theta(n.(log_2 n)^2)$

Best Case Scenario, Big Omega Notation (Ω) -When we don't need to get in the c_4 loop-

$\Omega(n.(log_2 n))$

Scientific Report

Scenario A:

According to the numerical data posted in 2023 YKS exam [ref.1] most of the students have a Gaussian distribution in fields like Turkish and Social Studies and far right skewed distribution on fields like Math and Science. If we assume all the students must solve from each field's questions, our graph will have a right skewed distribution. So, our mode is smaller than our mean which means there will be much more students in the lower grades dedicated to this information our computer has to make more switch operations in the mode area. According to my opinion as I written, if we want to list all student we should consider less switch operations because students will concentrated in one region of the graph so I would rather Dual Pivot Quick Sort over Shell Sort because as I mentioned we have to make less switch operations and if we choose Shell Sort algorithm we have to control every before element in the array recursively and the cost of this control will be grater than the Dual Pivot Quick Sort's controls.

Scenario B:

In practical terms, when implementing the sorting for the translation of a large dictionary, Dual-Pivot Quick Sort would result in faster processing times because the use of two pivots can potentially reduce the number of comparisons needed to sort the array, making it faster for sorting thousands of words. When we look at the table above, we can see Dual Pivot Quick Sort performs better in random elements. Although our Tur-Eng dictionary is sorted, when we translate all the words into Turkish it will be random and unordered so Dual Pivot Quick Sort will perform better run time rather than Shell Sort.

References

1. https://cdn.osym.gov.tr/pdfdokuman/2023/YKS/sayisalbilgiler20072023.pdf
2. https://www.geeksforgeeks.org/dual-pivot-quicksort/
3. https://www.geeksforgeeks.org/time-complexity-and-space-complexity/
4. https://www.programiz.com/dsa/master-theorem
5. https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/
6. https://www.geeksforgeeks.org/time-complexity-and-space-complexity/
7. https://www.youtube.com/watch?v=HFjJgSguqUA
8. https://www.youtube.com/watch?v=IViqgakt-Eg