Shell Sort vs Dual Pivot Quick Sort

YASAMIN VALISHARIATPANAHI

15 March 2024

—

Algorithm Analysis

—

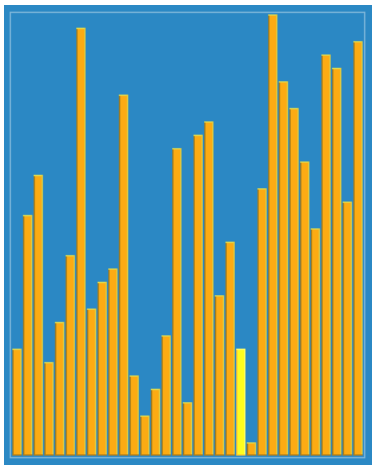
DOÇ. DR. ZERRİN IŞIK

Introduction

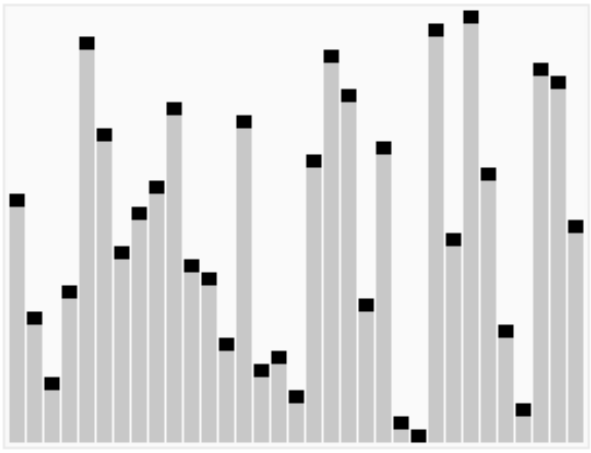
In this report we are going to examine how well these two-sorting algorithm, Shell Sort and Dual Pivot Quick Sort, perform in different scenarios.
We measure their speed in milliseconds, nanoseconds for different types of input. The goal is to see if there's a link between the expected speed, represented by Big O notation, and the actual results we get when we test them using Java Programming Language in Eclipse Application.
So, let's begin ....

THE PROCESS

Shell Sort[1]



Quick Sort[2](Dual Pivot Quick Sort[3])



Implementation

As the assignment said by creating a SortClass we invoked our two algorithms and test our algorithms with different input sizes to see how the algorithms will behave and how much time spends sorting these inputs. These input sizes are: 1.000, 10.000, 100.000 ; with Equal Integers (could be a worse case), Random Integers(Average Case), Increasing Integers(Best Case), Decreasing Integers(Worst Case).

For manage and better debugging I created classes for generating array inputs and running time of the sorts.
Fill the array with given input sizes and the way the inputs should be. Then sort them and measure the time of the sorting for each sorting algorithm and input cases.
"Sadly I had to remove the classes and their interfaces and make everything into inner classes, but since it was after everything worked there was no problem doing that since I had already debug the problems I encountered during writing this code."

[1] Shell Sort visualization (Accessing the Website you can check the references)

Runtime Analysis

For inputs in the given assignment: in milliseconds

DUAL PIVOT QUICKSORT VS SHELLSORT

	DUAL PIVOT QUICKSORT			SHELLSORT		
Input Size	1000	10000	100000	1000	10000	100000
Equal Integers	0	1	3	1	2	13
Random Integers	0	0	2	0	3	19
Increasing Integers	0	1	2	1	0	5
Decreasing Integers	0	1	2	0	0	4

For even larger input sizes: in milliseconds

DUAL PIVOT QUICKSORT VS SHELLSORT

	DUAL PIVOT QUICKSORT			SHELLSORT		
Input Size	100000	1000000	10000000	100000	1000000	10000000
Equal Integers	2	25	313	11	8	125
Random Integers	2	25	326	19	189	2509
Increasing Integers	2	25	326	2	26	325
Decreasing Integers	2	24	313	3	35	443

For inputs in the given assignment: in nanoseconds

DUAL PIVOT QUICKSORT VS SHELLSORT

	DUAL PIVOT QUICKSORT			SHELLSORT		
Input Size	1000	10000	100000	1000	10000	100000
Equal Integers	21627	281157	4281335	391198	4431701	20665939
Random Integers	25223	390435	5544449	219641	3905228	32486434
Increasing Integers	29116	393772	4325870	28709	392735	4876968
Decreasing Integers	27632	334452	4105765	39108	521397	5024800

[2] Quick Sort visualization (Accessing the Website you can check the references)

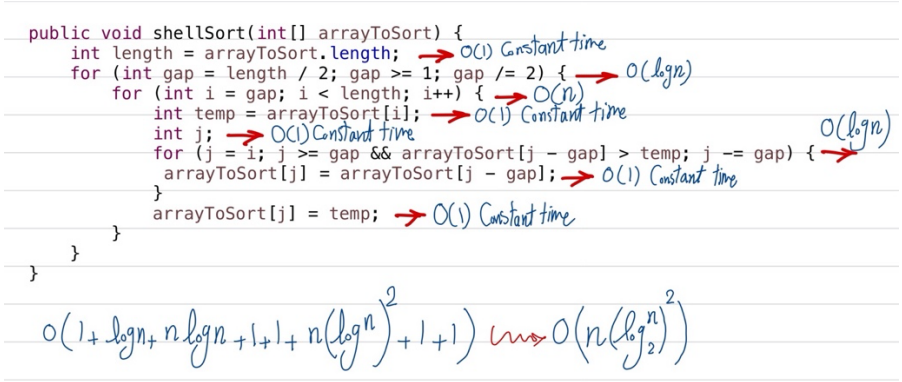
[3] For visualization of Dual Pivot Quick Sort please check the references.

Experiment Result

Based on what we obtain from the Runtime Analysis of our two algorithms, we understand that the running times of both of these sorting algorithms are depending on the input scenarios and array size. However, to make sure we need to establish a connection between the asymptotic running time complexity and the result of our output in our code.
But firstly I want to talk about the inputs that I get from the program.
So as you see the 1000, 10000, 100000 wasn't clearly enough since there are latterly lots of 0's in the result of program so I couldn't make any decision so I make it even larger 100000, 1000000, 10000000
Now I began to understand what is going on here. The running of the program took about nearly 6 seconds with these new entries but before the result would have come right away.
Also I added the inputs in the nanoseconds which the results will show clearly as before but the only differences for showing in milliseconds I just needed even more larger data but now in nanoseconds we can clearly see the changes of inputs effecting our sorts.

- **Shell Sort**: Shell Sort has an average time complexity of O(n $(\log n)^2$), in the worst case. It performs better than simple insertion sort for larger arrays due to its gap sequence, which reduces the number of comparisons and swaps needed to sort the array.
  Even in the video of explaining how Shell Sort[4] actually work took the narrator a bit time to sort the entire array in shell sort. We keep making the gap smaller and smaller in the mean time they value of indexes will compare to each other; this process will take place till the gap size reaches 1 and with Insertion Sort the array will be finally sorted as we want. This process of starting with large gap and make it smaller until the gap size between these indexes reaches 1 to do the Insertion Sort moving these smaller elements from left side of array to the right side and also moving the larger elements from right side of array to the left side, in the worst case the Shell Sort is basically an Insertion Sort. Our average time complexity depends on these gaps that we have so different gap sizes will change the complexity because when we have more gaps then we also have to do more swaps in order to finish the sort. This sort is not stable because we keep sorting these little pieces till we finally get the last sort that we want.

[4] YouTube video of Explaining Shell Sort in an example



Best Case: Θ(n) _ Average Case: Ω(nlogn) _ Worst Case: O(n$logn^2$)
In average case depends on the gap[7].

[7] Shell Sort's Advantages and Disadvantages

- **Dual Pivot Quick Sort**[5]: Dual Pivot Quick Sort has an average time complexity of O(n log n) in the worst case. It improves the original Quick Sort by using two pivots to partition the array into three sections, leading to better performance in some scenarios.
  When array is sorted (increasing) sub-arrays are unbalanced so it has the worst case. In the video[3] of ordering large number of data we can clearly see that how fast the algorithm sort in different scenarios.

```
public void dualPivotQuickSort(int[] arrayToSort) {
        if (arrayToSort == null || arrayToSort.length <= 1) {
            return;
        }
        dualPivotQuickSort(arrayToSort, 0, arrayToSort.length - 1);
    }

    private void dualPivotQuickSort(int[] array, int left, int right) {
        if (left < right) {
            int[] pivots = partition(array, left, right);
            dualPivotQuickSort(array, left, pivots[0] - 1);
            dualPivotQuickSort(array, pivots[0] + 1, pivots[1] - 1);
            dualPivotQuickSort(array, pivots[1] + 1, right);
        }
    }

    private int[] partition(int[] array, int left, int right) {
        if (array[left] > array[right]) {
          swap(array, left, right);
        }

        int smallerPivot = left + 1;
        int biggerPivot = right - 1;
        int i = left + 1;

        while (i <= biggerPivot) {
            if (array[i] < array[left]) {
            swap(array, i, smallerPivot);
            smallerPivot++;
            } else if (array[i] >= array[right]) {
                while (array[biggerPivot] > array[right] && i < biggerPivot) {
                  biggerPivot--;
                }
                swap(array, i, biggerPivot);
                biggerPivot--;

                if (array[i] < array[left]) {
                  swap(array, i, smallerPivot);
                  smallerPivot++;
                }
            }
            i++;
        }

        swap(array, left, --smallerPivot);
        swap(array, right, ++biggerPivot);

        return new int[]{smallerPivot, biggerPivot};
    }

    private void swap(int[] array, int i, int j) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }
```

Master method: $T(n) = 3T(n/3) + O(n)$, $a > 1$, $b > 1$

Comparing $n^{\log_3 3}$ vs n: $n^1 = n$ Case 2 $\Theta(n^{\log_3 3} \log^d n) = \Theta(n \log n)$ $O(n \log n)$

Best Case: $\Theta(n log_3^n)$ _ Average Case: $\Omega(n log_2^n)$ _ Worst Case: $O(n^2)$

In the best case it's not $\Theta(n log_2^n)$ since each partition is the third part of the entire current range. Also, if one pivot is not good (it is close to one of the ends of the current range), there is a good chance that the second pivot will be better.[8]

---

[5] YouTube video of Explaining Dual Pivot Quick Sort in an example

[3] Visualization of Dual Pivot Quick Sort

[8] stackExchange Website

Conclusion

In conclusion, based on the asymptotic time complexities and the experimental results, the appropriate sorting algorithm for each scenario can be determined like this:

- **Scenario A** (Placing students at universities): In placing (3 million) Students case, we have a very larger set of data which according to the StackOverflow[6]; Dual Pivot Quick Sort can handle a large data much better with its average time complexity of O(nlog_2^n) also in the program result we can see the differences that Shell Sort took so much time to sort so for placing these students at universities I will choose to use Dual Pivot Quick Sort which take less time that Shell Sort.
- **Scenario B** (Translating dictionaries): For translating dictionaries with thousands of words, we can use both algorithms. However, some dictionaries could be large as I said before since Dual Pivot Quick Sort can handle larger data better and faster because in this algorithm it keep slicing the array and find the right index of pivot so our performance can be improve if we choose this algorithm instead of the Shell Sort for larger dictionaries because the more data we have the gap between data are bigger so means more swapping work for Shell Sort and we want to find the words fast in dictionary so it's not a right choice for this scenario.

References

- [1]: Wikipedia – Shellsort: https://en.wikipedia.org/wiki/Shellsort
- [2]: Wikipedia – Quicksort: https://en.wikipedia.org/wiki/Quicksort
- [3]: Dual Pivot Quick Sort: https://www.youtube.com/watch?v=qXY2gSEcueQ
- GeeksforGeeks – Sorting Algorithms (Shellsort): https://www.geeksforgeeks.org/shellsort/
- GeeksforGeeks – Sorting Algorithms (Quicksort): https://www.geeksforgeeks.org/quick-sort/
- [4]: Shell Sort YouTube video: https://www.youtube.com/watch?v=ddeLSDsYVp8
- Quick sort: https://www.youtube.com/watch?v=Vtckgz38QHs
- [5]: Dual Pivot Quick Sort YouTube Video: https://www.youtube.com/watch?v=XYVbjQXkmiI
- [6]: Differences between Shell Sort & Dual Pivot Quick Sort: https://stackoverflow.com/questions/20917617/whats-the-difference-of-dual-pivot-quick-sort-and-quick-sort
- [7]: Shell Sort's Advantages and Disadvantages: https://www.shiksha.com/online-courses/articles/shell-sort-advantages-and-disadvantages/
- [8]: StackExchange: https://cs.stackexchange.com/questions/76967/quicksort-dual-pivot-or-single#:~:text=Basically%20Dual%2Dpivot%20Quicksort%20buys,of%20the%20entire%20current%20range.

[6] StackOverflow Website the difference between Shell Sort & Dual Pivot Quick Sort Algorithm