Visual Sorting: Visual Paradigm Implementation for Tree-based Sorting Algorithm

Achmad Ginanjar
Directorate General of Taxes
Gatot Subroto 40-42
Jakarta, Indonesia

ABSTRACT

In the world of sorting algorithms, the visual interpretation approach has not been explored yet. In the attempt at this approach, this paper will explore an algorithm that uses object unit values for sorting purpose. Those unit values represent how visual interpretation understanding an integer. The unit values were taken from an integer and split into values of unit, ten, hundred, etc. The integer was processed from a list of integers. Those integers in the list then sorted. The sorting process was done in two steps. The two processes are similar to the concept of divide and conquer in other sorting algorithm but having different logic. The results were shown to be accurate and stable. The result also indicated that the process could be done parallel. This indicates that the visual interpretation approach can be used as a sorting algorithm with the possibility of parallel execution.

Keywords

Tree sorting, visual paradigm, python sorting, integer sorting, tree object, time complexity, OOP

1. INTRODUCTION

The human problem-solving scenario has inspired many researchers to solve IT problems for decades. From Object-Oriented Concept to Artificial Neural Network (ARNN) algorithms are inspired on how a biological system works. One of which is how the visual (human eye) mechanism works to solve problems. When human see a cat or any object, human do not write any complex math to recognise it as a cat. Similarly, the way how a person drawing on paper that guided by visual interaction, inspired the foundation of drawing on a screen (Sutherland, 1938). Those paper starts a considerable jump for IT invention till now. Moreover, Elliot, E. (2018, 11 1). *The Forgotten History of OOP*[1] stated that the terminology of Object-Oriented Programming coined by Alan Kay in 1966 or 1967 also inspired by Sutherland's work.

In the world of sorting algorithm, the visual interpretation approach has not been explored yet. Many sorting algorithms implement the sorting concept solely based on an object as an object complete value. In contrast, this paper will explore an algorithm that uses an object unit value for sorting purpose. On the basis of this unit, the sorting algorithm evaluate the unit values and construct grouping like object. This object is the basis of sorting result retrieval.

2. METHODOLOGY

2.1 Definition

The experiment implements a few technical terms that frequently used to explained the method practised. The most important term used alongside the experiment was "tree object".

1.1.1. Definition 1

Let

$$T_{d\_max} = \{1{:}\, d, 2{:}\, d, \ldots, n{:}\, d\}$$
$$\{0 \leq n \leq 9 | n \in \mathbb{Z}, d < 0 \; | d \in \mathbb{Z}\}$$

Equation 1 Last Object

T be a dictionary object that has n keys. In this experiment n is an integer between 0 to 9, inclusive and without repetition. d_max represent the maximum size of integers in a List, where size is the count digit. d is an integer that count how many integers that similar. This T_{d_max} object is the last object of a tree.

1.1.2. Definition 2

Let

$$T_d = T_{d0}, T_{d1}, \ldots, T_{dn}$$
$$\{0 \leq n \leq 9 | n \in \mathbb{Z}, d \in \mathbb{Z}\}$$

Equation 2 Tree Object

T be a tree object that has n child where n between 0 to 9 inclusive and unique. d represent the level of a tree. In this experiment T_d is a tree object T with d as a name. Therefore, T_0 is a tree object that has string 0 as name.

1.1.3. Definition 3

Let

$$R = T_0$$

Equation 3 Root Object

R be root and a tree container. The evaluation of R is always at 0 level and require 1 complexity to build. Therefore, the minimum complexity to start the algorithm and sort nothing is always 1 or 9^0.

1.1.4. Definition 4

Let

$$L = [0, \ldots, n] \; \{n \in \mathbb{Z}\}$$

Equation 4 List of Integers

L be a list of integers. The member of the list is not unique. The size of the list is the count of their member.
The list was generated with bellow python code:

---

[1] Elliot, E. (2018, 11 1). The Forgotten History of OOP. Retrieved from Medium: https://medium.com/javascript-scene/the-forgotten-history-of-oop-88d71b9b2d9f#:~:text=%E2%80%9CObject%2DOriented%20Programming%E2%80%9D%20(,his%20Sketchpad%20Thesis%20in%201963.

```
def genList(n,n2=1000000):
    return [rd.randint(0, n2) for i in range(n)]
```

Pseucode 1 List Generation

2.2 Preliminaries

The current experiment involved splitting a value as a unit's level. The split values were values of unit, ten, hundred, etc. This method based on the understanding of how a human visual works. When comparing tens and hundreds, the human does not always take a complete value and compare them. However, by only look at the digit length, human know that hundreds are bigger than tens.

Fig 1 Hundreds vs Tens

Even without knowing the real value, hundred(left) is bigger than ten(right) in the above picture. Therefore, both objects can be sorted into [ten, hundred].

The latter step was building tree object to represent level and group level. The tree object was built per level. The final tree object will have a nested tree inside a Root. The depth of a tree depends on how much level a number has. For example, an integer of 435 will have three nested trees representing a unit of 5, a ten of 3 and a hundred of 4.

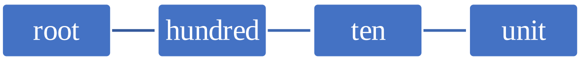


Fig 2 Unit levels

2.3 Splitting an Integer

In this experiment, one thousand lists (L) of different length of integer were generated randomly. Each list L had member size from 0 to 100.000 integer. Each list L was processed separately in ascending order by the size of the list. A 10 step were used to get 10.000 lists out of the maximum 100.000 size. For example: 0,10, 20,….100.000

Each list L then iterated through its member. Each member of the list L that is an integer value, then split into the unit, tens, hundred etc. The split process was started from the biggest unit of each integer. An integer of 146 was split into 1, then 4, and lastly, 6. In the case of integer 146, 1 had 4 as its member, and 4 has 6 as its member.

2.4 Constructing Tree Object

Using tree object implementation, each unit was grouped by its unit level. In the case there was an integer that having a smaller maximum unit, compared with the previous integer, then 0 is added to represent a higher unit. For example, integer of 34 represented with 0034 to represent 0 hundred and 0 thousand.

2	3	4	5	➔	2	3	4	5
		3	4	➔	0	0	3	4
	1	4	6	➔	0	1	4	6
3	5	6	7	➔	4	5	6	7
3	2	1	0	➔	3	2	1	0
			3	➔	0	0	0	3

Fig 3 Conversion

The construction of the tree object was started by initiating an empty tree object. That object was used as a root container to maintains the whole nested tree object. This object was the same object like any other tree object in this paper. The only difference was the user names the object.

The next step was to insert each integer into the tree. Each integer that had been split as a result of the previous procedure, was inserted one by one. Similar to splitting integer, inserting the integer was started from the biggest unit. Following the previous example, using an integer of 146, 1 was inserted first, then followed by 4, then 6. This process will create an object such as

$$T_0 = T_{10}, T_{11}, \dots, T_{1n}$$

Equation 5 A level

The least T object was different from its roots. The last T object was a dictionary that records the occurrences of an integer. For example, if a list consists of two integers of 146 like

$$T_{dmax} = [\dots,146,\dots 146]$$

Then

$$T_{dmax} = \{\dots,6{:}2,\dots\}$$

Fig 4 Similar Integers

This represent that there are two 146 integer value in the list L.

2.5 Obtaining sorted integer

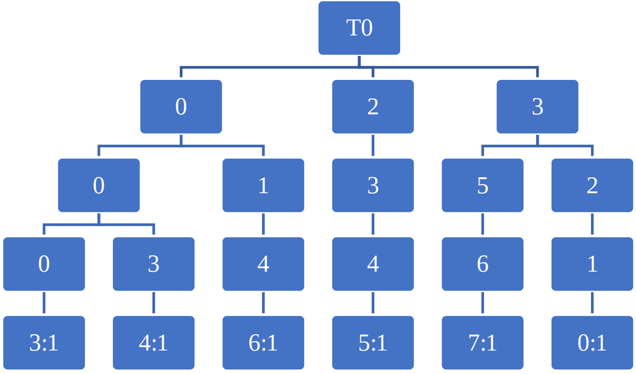


Fig 5 Tree Object

The tree T_o object then reviewed to get the sorted integers. From the last object, which represents the last or smallest unit, the integer is evaluated. Started from the last child, up to the parent T_{d_max} of each tree, then continue to the upper parent T_{d_max-1} until reach the top root T_0 .

For each tree t_n, the collection process were made by ascending walk throughout a list of integer from 0-9 ([0,1,2,3,4,5,6,7,8,9]) for each child. In the least tree T_{-1}, multiple numbers were returned when the original list L had more than one value that equal.

2.6 Proof of Concept

Each group were sorted with two different algorithms. The algorithms were the Visual Sorting algorithm (the subject of this paper) and the python default sorting algorithm, *Timsort* algorithm that proposed by Peters, T. (2002, 07 20)[2]. The

---

[2] Peters, T. (2002, 07 20). [Python-Dev] Sorting. Retrieved from Python Mailinglist:

results then compared in both values and time executions. The experiment was done only with integer type object. Another type of objects like string or float, is not studied yet. However, the implementation will be similar. In addition, the experiment was done with the python language. The performance might not be the best compared to low-level language such as C. Nevertheless, the algorithm was done in such a way that might be the best in the python way.

3. RESULTS AND DISCUSSION

3.1 Tree Object

Implementation of tree object for sorting from other studies resulted in a similar tree object with apparent differences. According to Skylarof et al. (2005), the value of the tree object was the complete value of its own number. For example, for an integer value like 789. In Skylarof works, A single node of a tree object that contains the integer is having 789 as its value. In contrast, In this experiment, a value like 789 was split into its unit to represent levelling. The figure below shows the differences between both methods.

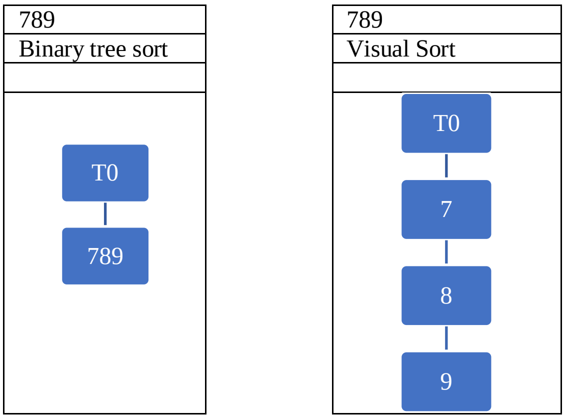
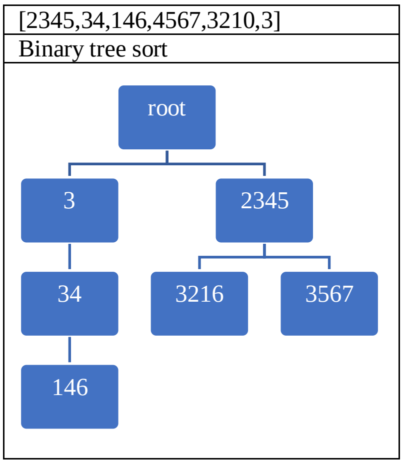


Fig 6 Binary Sort vs Visual Sort

As can be seen, the visual sort object size is more significant than the binary tree sort object. This was because visual sort breaks a value into units, while on the other hand, binary tree sort treats a value as a complete value. However, in the experiment, with more value, the visual sort was resulting in similar objects with a binary tree sort. The figure below shows the final object of both algorithms.



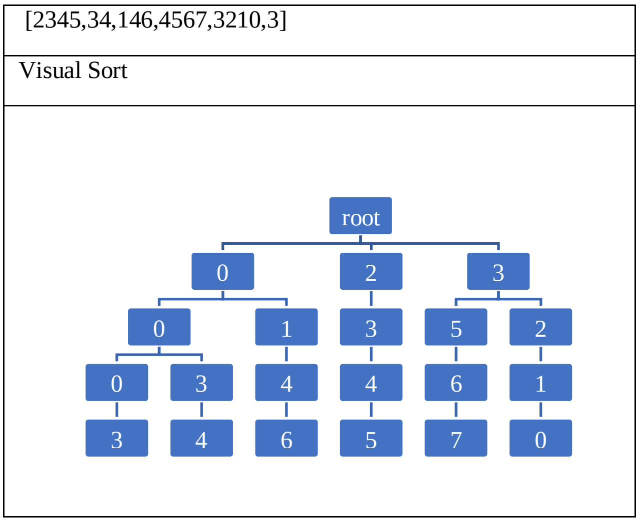


Fig 7 Final Tree Result

Using this approach for object creation, the visual model results in better computation consumption. This is because there was not any comparison done to determine bigger or smaller value. Unlike a binary tree sort that compares two value in each step, the visual model breaks units only. The behaviour is similar to the merge sort algorithm (Skiena, 2008) that breaks a list into sub list until only a single value left.

3.2 Obtaining Sorted Value

The final step of the visual sorting algorithm in this experiment was sorted value retrieval. The results of the visual sorting algorithm were sorted value like the one produced by binary tree sort. However, the time used to execute visual sorting was more significant than the time achieved by the default python sorting algorithm.

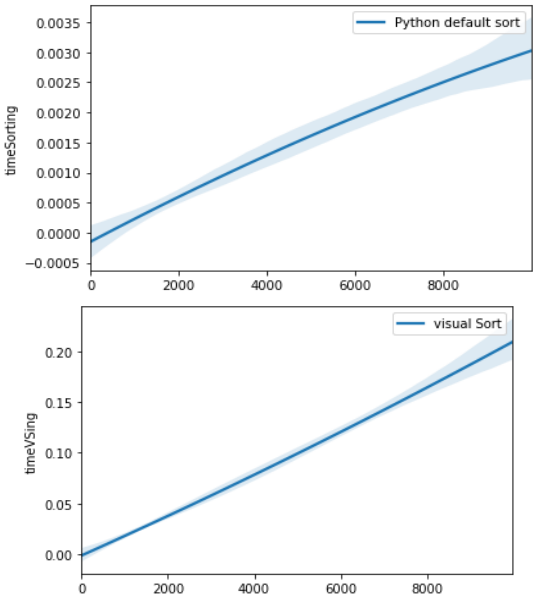


Fig 8 Time Comparison, Time Sort vs VIsual Sort

---

https://mail.python.org/pipermail/python-dev/2002-July/026837.html

Sorted values:

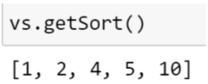
			3
		3	4
	1	4	6
2	3	4	5
3	2	1	0
3	5	6	7

Fig 9 Sorted Values

Nevertheless, the result suggests that visual sorting can be used as an alternative to the sorting algorithm that available now.

3.3 Correctness and Time Complexity

The implementation of visual sorting was straightforward. It did not need any complex mathematical theory and implementation. Due to this fact, the correctness of the visual sorting algorithm result can be guaranteed. This theory is proofed by this experiment that gave consistent and ordered result for each list L.



```
vs.getSort()

[1, 2, 4, 5, 10]
```

Pseucode 2 Get Sorted Value

One of the downsides of this method was time complexity. Due to its nature, the Visual Sorting algorithm time complexity mostly depends on the maximum integer size in a list. In this paper, maximum size was known with d notation. The maximum d value was also translated as the depth of a tree. Below is a comparison with other soring method stated in this paper:

Table 1 O Time Complexity

No	Name	Best	Worst
1.	Timsort	n	$n\ Log\ n$
2.	Merge Sort	$n\ Log\ n$	$n\ Log\ n$
3.	Visual Sorting	$n + d + 1$	$n + 9^{d!} + 1$

Due to the nature of visual sorting works, there is a big opportunity for parallel sorting implementation. This is because each integer was processed individually without actually comparing with other value. With a better coding algorithm, the visual sorting algorithm can be done parallelly.

4. CONCLUSION

Prior works have documented the successfulness of sorting algorithms using a tree-based algorithm; Skylarof for example, introduces the Binary Tree Sort algorithm for sorting by building a tree object with two legs each node. Sutherland did another major work that is important to mention. His study starts the merging of visual concept into a computer model. However, these studies have never been implemented together.

In this study, the implementation of a visual concept is applied in a sorting algorithm. The algorithm sorts the value base on its single unit. This study found that in all experiment, the values are sorted correctly. Although the time is not better than the Tim Sort algorithm, visual sorting can be helpful in some cases. These findings merge the sorting tree algorithm and visual concept, confirming that a unit value like units, tens, hundreds, etc., can be very meaningful in computer algorithm as to how the human brain interprets it. Also, a new sorting algorithm is introduced. This study, however, introduces that the benefit of the sorting algorithm may address more time and resource to finish. Most notably, this is the first study to our knowledge to implement a sorting algorithm with a visual concept.

The results provide compelling evidence for long term collaboration between researchers to explore how the human vision works and implement it in a computer model. However, some limitations are worth addressing. Although the concept is easy to draw, the experiment was assessed with Python language; there might be an improvement if low-level programming language was being used. Another limitation arises because the study was done with only integer-base values. Future work should therefore include other low-end language and other data type.

In the future, this study can be extended with a different data type. This study focuses on the integer data type. The string data type that commonly uses can be implemented with a slight adjustment. Another extension can also implement low-level programming like C, C#, C++ that can guarantee the robustness.

5. DECLARATION

Funding

This research received no external funding.

Conflicts of interest/Competing interests

The authors declare no conflict of interest.

Availability of data and material

All data and material were produced in the lab by code supplied, therefore can be done repeatedly. No data sharing available except the code.

Code availability

The codes are attached in the paper and were produced by author.

6. REFERENCES

[1] Zhang, Jian & Jin, Rui. (2021). An Improved Bubble Sort Method - Marking Bubble Sort. 10.1007/978-3-030-70042-3_121.

[2] Liu, Yu-Zhe & Tang, Shyue-Ming & Chang, Jou-ming. (2021). On the Decycling Number of Bubble-sort Star Graphs.

[3] Akhter, Naeem & Idrees, Muhammad & Furqan-ur-Rehman,. (2016). Sorting Algorithms – A Comparative Study. International Journal of Computer Science and Information Security,. 14. 930-936.

[4] Skiena, S. S. (2008). 4.5: Mergesort: Sorting by Divide-and-Conquer. In *The* Algorithm *Design Manual (2nd ed.)* (pp. 120-125). Springer.

[5] Sklyarov, V., Skliarova, I., & Pimentel, B. (2005). PGA-based implementation and comparison of recursive and iterative algorithms.

[6] Sutherland, I. E. (1938). *Sketchpad, a man-machine graphical communication* system. Massachusetts Institute of Technology.