Payal Chavan

Summer 2024

CS 5800 Algorithms (Seattle)

Date: 05/30/2024

Question 1: Output of kth Element Sorted Array

```
payalchavan@Payals-MacBook-Air /Users/payalchavan/Documents/Algorithms

⚡ /usr/bin/python3
/Users/payalchavan/Documents/Algorithms/Assignment3_Payal_Chavan-KthInSortedArray.py

unionkth: 1 -- unionkthIterative: 1 -- unionKthCompare: 1

unionkth: 13 -- unionkthIterative: 13 -- unionKthCompare: 13

unionkth: 22 -- unionkthIterative: 22 -- unionKthCompare: 22

unionkth: 24 -- unionkthIterative: 24 -- unionKthCompare: 24

unionkth: 25 -- unionkthIterative: 25 -- unionKthCompare: 25

unionkth: 26 -- unionkthIterative: 26 -- unionKthCompare: 26

unionkth: 37 -- unionkthIterative: 37 -- unionKthCompare: 37

unionkth: 49 -- unionkthIterative: 49 -- unionKthCompare: 49

unionkth: 88 -- unionkthIterative: 88 -- unionKthCompare: 88

unionkth: 90 -- unionkthIterative: 90 -- unionKthCompare: 90

(base)

payalchavan@Payals-MacBook-Air /Users/payalchavan/Documents/Algorithms
```

Analysis:

The unionkth algorithm, which finds the kth element in the union of two sorted arrays, achieves a time complexity of O(log(m) + log(n)). This complexity arises from the binary search approach used in the algorithm.

We can analyze this by -

1. The algorithm starts by dividing the length of the arrays by 2. This step is typical in binary search, where the search space is halved at each iteration.
2. In binary search, you begin with the entire array and progressively reduce the search space by half. Depending on the condition, you choose either the left or the right half.
3. Consequently, the time complexity of binary search is **O(log(n))**, where **n** represents the size of the array.
4. In the case of **unionkth**, binary search is performed on two arrays. Hence, the overall time complexity becomes **O(log(m) + log(n))**, where **m** and **n** denote the sizes of the two arrays.
5. The algorithm compares the middle elements of the two arrays and discards half of the search space in each step, achieving the desired logarithmic time complexity.

Question 2: Output of Median Finding Code

```
payalchavan@Payals-MacBook-Air /Users/payalchavan/Documents/Algorithms

⚡ /usr/bin/python3
/Users/payalchavan/Documents/Algorithms/Assignment3_Payal_Chavan-Median.py

Selection::Random-Index Value: 520

Selection::Pseudo-Median Value: 520

Selection::Python-Builtin Value: 520


Array Size: 1

K: 0

Selection::Random-Index Value: 777

Selection::Random-Index Time: 0.38 microseconds

Selection::Pseudo-Median Value: 777

Selection::Pseudo-Median Time: 0.25 microseconds

Selection::Python-Builtin Value: 777

Selection::Python-Builtin Time: 0.29 microseconds


Array Size: 11

K: 3

Selection::Random-Index Value: 413

Selection::Random-Index Time: 5.67 microseconds

Selection::Pseudo-Median Value: 413

Selection::Pseudo-Median Time: 10.96 microseconds

Selection::Python-Builtin Value: 413

Selection::Python-Builtin Time: 0.54 microseconds


Array Size: 21

K: 16

Selection::Random-Index Value: 642
```

```
Selection::Random-Index Time: 9.71 microseconds

Selection::Pseudo-Median Value: 642

Selection::Pseudo-Median Time: 17.63 microseconds

Selection::Python-Builtin Value: 642

Selection::Python-Builtin Time: 0.79 microseconds


Array Size: 31

K: 9

Selection::Random-Index Value: 460

Selection::Random-Index Time: 18.38 microseconds

Selection::Pseudo-Median Value: 460

Selection::Pseudo-Median Time: 24.25 microseconds

Selection::Python-Builtin Value: 460

Selection::Python-Builtin Time: 1.29 microseconds


Array Size: 41

K: 32

Selection::Random-Index Value: 757

Selection::Random-Index Time: 26.21 microseconds

Selection::Pseudo-Median Value: 757

Selection::Pseudo-Median Time: 28.67 microseconds

Selection::Python-Builtin Value: 757

Selection::Python-Builtin Time: 1.79 microseconds


Array Size: 51

K: 28

Selection::Random-Index Value: 547

Selection::Random-Index Time: 28.08 microseconds

Selection::Pseudo-Median Value: 547

Selection::Pseudo-Median Time: 35.88 microseconds
```

Selection::Python-Builtin Value: 547

Selection::Python-Builtin Time: 2.21 microseconds

Array Size: 61

K: 31

Selection::Random-Index Value: 483

Selection::Random-Index Time: 27.83 microseconds

Selection::Pseudo-Median Value: 483

Selection::Pseudo-Median Time: 41.00 microseconds

Selection::Python-Builtin Value: 483

Selection::Python-Builtin Time: 2.79 microseconds

Array Size: 71

K: 51

Selection::Random-Index Value: 759

Selection::Random-Index Time: 26.87 microseconds

Selection::Pseudo-Median Value: 759

Selection::Pseudo-Median Time: 43.67 microseconds

Selection::Python-Builtin Value: 759

Selection::Python-Builtin Time: 3.50 microseconds

Array Size: 81

K: 75

Selection::Random-Index Value: 939

Selection::Random-Index Time: 28.63 microseconds

Selection::Pseudo-Median Value: 939

Selection::Pseudo-Median Time: 49.54 microseconds

Selection::Python-Builtin Value: 939

Selection::Python-Builtin Time: 3.71 microseconds

```
Array Size: 91

K: 72

Selection::Random-Index Value: 770

Selection::Random-Index Time: 46.46 microseconds

Selection::Pseudo-Median Value: 770

Selection::Pseudo-Median Time: 54.87 microseconds

Selection::Python-Builtin Value: 770

Selection::Python-Builtin Time: 4.29 microseconds
```

Analysis:

The time complexity of **finding a random index** is **O(1)**. This is because it's a straightforward operation that doesn't rely on the size of the input. In other words, regardless of the input size, the time required to find a random index remains constant.

When it comes to finding the **pseudo median**, we are utilizing an approach like the "median of medians," the time complexity is **O(n)**. This is due to the following reasons-

1. The "median of medians" algorithm divides the array into groups of 5 elements each.
2. Within each group, finding the median can be done in constant time (since there are only 5 elements).
3. Next, the algorithm recursively computes the median of these medians.
4. Since the number of groups decreases by a constant factor (5) at each recursion level, the overall time complexity remains linear.

Overall, as the pseudo median code divides the elements into chunks/quintets it's time complexity i.e., O(n) is greater than the time complexity of choosing random index for splitting array, which is O(1). From our above output analysis, for each of the cases, we see that the time to find median using random index value is faster than the time to find the pseudo median value.

We can observe that the Python built-in median value takes much less time as compared to the median using random index value and pseudo median value. This is because Python's built-in sort() function uses a sorting algorithm called "Timsort". This makes it one of the most efficient sorting algorithms in practical use. Hence, the Python built-in median value method is faster compared to the other two methods.

Reflection:

In the course of these coding exercises, I acquired knowledge on three different approaches for finding the median value: using a random index, calculating the pseudo median, and leveraging Python's built-in functionality. Additionally, I conducted an analysis of their respective time complexities to determine which approach is more efficient. Furthermore, I delved into methods for finding the kth element in an array, exploring both recursive and iterative techniques.

Acknowledgements:

I would like to express my sincere gratitude to the following individuals and resources for their invaluable contributions to this assignment:

1) Prof. Bruce Maxwell: Thank you, Professor Bruce Maxwell for your guidance in this assignment. Your expertise in algorithms greatly influenced my work.
2) Classmates and TA's: I appreciate the discussions of my classmates, and TA's who clarified my doubts.
3) DPV Algorithms Textbook: This textbook was a useful resource for me to understand the basics of algorithms.
4) https://geeksforgeeks.org/timsort/: To understand use of "timsort" functionality in Python.
5) https://people.computing.clemson.edu/~goddard/texts/algor/A5.pdf: To learn how to find median using pseudo median approach.