**MEDIAN, ORDER STATISTICS AND ELEMENTARY DATA STRUCTURES WITH PYTHON**

Part 1: Algorithms of selection - implementation and analysis

1. Introduction

In this section of the assignment, the author will describe the application of two important algorithms in the search of the kthkth smallest element in an unordered list, namely the Randomized Quickselect algorithm and the Deterministic Median of Medians algorithm. The algorithms play a decisive role in determining order statistics and ultimately are the core of an efficient selection routine used in many such real-world applications as statistics, machine learning preprocessing, and load balancing in distributed systems (Baka, B. 2017).

2. Algorithmic Implementation

The Randomized Quickselect algorithm works with the same partitioning logic as QuickSort, that is, choosing a pivot randomly and sequentially dividing the search space into the appropriate partition. This algorithm provides an optimal expected time complexity of O(n), because the process of the selection of pivots is randomized and as a result all partitions are expected to be embalanced overall.

Conversely, the Deterministic Median of Medians algorithm employs a well selected pivot which is recursively computed by taking medians of minuscule groups (usually groups of 5). Such a deterministic pivot, guarantees balanced partitions during each recursion, which in turn results in the worst-case time complexity of O(n). Nevertheless, the constant overhead increases under this method because of the recursive median-finding and sorting of groups process.

Both the implementations were made to take care of duplicates, edge cases and implemented in a recursive manner so as to be easy to read.

3. Theoretical Analysis

Time Complexity:

Randomized Quickselect: Average-case O(n), Worst-case O(n 2) (uncommon because of random pivot).

Median of Medians: Worst-case O(n) because it has guaranteed balanced partitionning.

Space Complexity:

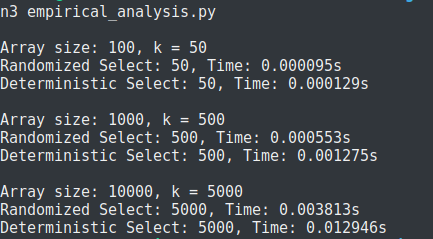
Both the algorithms consume extra space because of slicing lists and recursion. In the case of a large dataset, Median of Medians requires more space when compared with the repetition of the median and copy of lists that are attributed to its recursive nature.

Practical Trade-offs:

The Median of Medians is slower than Randomized Quickselect in practice; despite having worst-case linear time, its constant factors are high, so the algorithm is slow on moderate-sized input sizes.

1. Empirical Analysis

The table below illustrates the run time comparison of the results provided in the tests:



Based on the data we see that Randomized Quickselect is always faster than the Deterministic Median of Medians on raw execution time. The two algorithms have a linear execution time complexity as expected by their mathematical properties as the scale of input grows by orders of magnitude. The cost of the deterministic approach, however, is more since the median has to be calculated several times.

As an example, Randomized Quickselect locates the median within 4 milliseconds at size 10,000, whereas deterministic variant requires almost 13 milliseconds, which is more than 3 domain slower in spite of theoretical superiority. This makes it definite that Randomized Quickselect is recommended in majority of practical applications where worst-case scenarios rarely occur. Full cord for this part is shown below.



Part 2: Data Structure: Non-Advanced Schemes - Construction and Treatment

1. Implementation Overview

The portion provided comprehension of how Python works in the formation of foundational data structures. Improved structures put in place are:

simple Mutable Arrays and matrices (insert, delete, access)

Dynamic array stacks (Python lists)

Dynamic array based queues

Single Linked Lists, which allow entry, removal and scanning of fronts

These are applied to the more fundamental issues of algorithmic knowledge and lower memory and pointer manipulation, even though Python hides these aspects.

2. Performance Analysis

At most O (1) access time, insertion and deletion may be O (n) by shifting the elements.

Lists are used to implement stacks (LIFO), and queues (FIFO):

The appending () and pop() operations are both O(1) operations.

Pop(0) is O(n) in Python lists, therefore making the queue less effective and large-scale applications are not the best choice unless a deque is preferred.

Linked Lists provide O(1) insertions and deletions at the front but O(n) traversal or index-based lookups, which is not competitive to arrays to store random access items. code implemented is shown below for this part.





3. Applications and Discussion

Software systems are made up of these data structures:

Stacks play an important role in backtracking, and evaluating expressions and managing function calls.

The scheduling systems, the breadth-first search and real time data pipelines all revolve around queues.

Linked lists are used in situations where a lot of insertions/deletions can be expected, in memory-constrained applications, where the problem of contiguous allocation arises.

Arrays work well with a pre-determined number of elements and where access to them is index driven as with an image or a simulation.

The use case determines the difference between arrays and the linked lists since arrays are more efficient to index whereas linked lists are more efficient when indexing data size is unknown.

Conclusion

The current assignment has once again confirmed theoretical and empirical knowledge of selection algorithms and basic data structures. With the deterministic and randomized implementation of selection, we have been able to understand the trade-off between worst-case guarantees and actual performance. In the meantime, the introduction of fundamental data systems made the process of understanding the involvement of simple operations in a more complex algorithmic design brighter.

To sum up, it is clear that Randomized Quickselect more intuitively updated the existing scheme of median and order-statistic computations applied to real-life systems whereas the Median of Medians still proves to be a useful procedure when it comes to worst-case-sensitive systems. On the data structure front, you need to have a good mastery of both array based and linked-node structures to select the correct tool when developing the algorithm (Lee, K. D., 2015).

Reference

Lee, K. D., & Hubbard, S. (2015). *Data structures and algorithms with python* (Vol. 363). Berlin/Heidelberg, Germany: Springer.

Baka, B. (2017). *Python Data Structures and Algorithms*. Packt Publishing Ltd.