Assignment 6: Medians and Order Statistics & Elementary Data

Shrisan Kapali

Student Id: 005032249

MSCS 532 – Algorithms and Data Structures

Satish Penmatsa

February 09, 2025

Part 1: Implementation and Analysis of Selection Algorithms

GitHub Link

https://github.com/ShrisanKapali-Cumberlands/MSCS532_Assignment6

Implementation of Medians of Medians algorithm – Deterministic Algorithm

Many deterministic and randomized algorithms are used to find the kth smallest element in an unordered array. Median-of-Medians is a deterministic algorithm that helps find the k^{th} smallest element in an unsorted array. In this deterministic algorithm, the array is divided into subarrays and sorted, and the median of each subarray is found. The median of the array formed using the medians of all the subarrays is found using recursive methods. The median acts as the pivot, and recursively applying this algorithm, the k^{th} smallest element in the array is found (Mao, 2024).

As the steps involved in the median-of-medians algorithms are,

1. Diving the array into subgroups of 5 – O (n)
2. Finding the median of each subgroup – O (n)
3. Finding the median of the new median groups – O (n)
4. Partitioning the array – O (n)
5. Recursively searching the correct partition – O (0.7n)

Overall = O (n) + O (n) + O (n) + O (n) + O (0.7n) = O (n)

The recurrence relation is given by:

$T(n) = T(n/5) + T(0.7n) + O(n)$

As in this median-of-median deterministic algorithm, at each level, O(n) work is done regardless of the best or worst case, and the overall time complexity remains O (n). The disadvantage of this algorithm is that it recursively sorts the arrays in groups of 5 and creates a new array of medians, which can add up memory overheads (“What are the advantages and disadvantages of median-of-median algorithms,” 2023).

Implementation of Randomized Quickselect – Randomized Algorithm

Randomized quickselect is a divide-and-conquering algorithm that is like quicksort but only recurs in the direction where the kth smallest element is present (“Quickselect Algorithm,” 2024). It is implemented by choosing a pivot at random, avoiding worst-case run time $O(n^2)$. The time complexity of randomized quickselect is as follows:

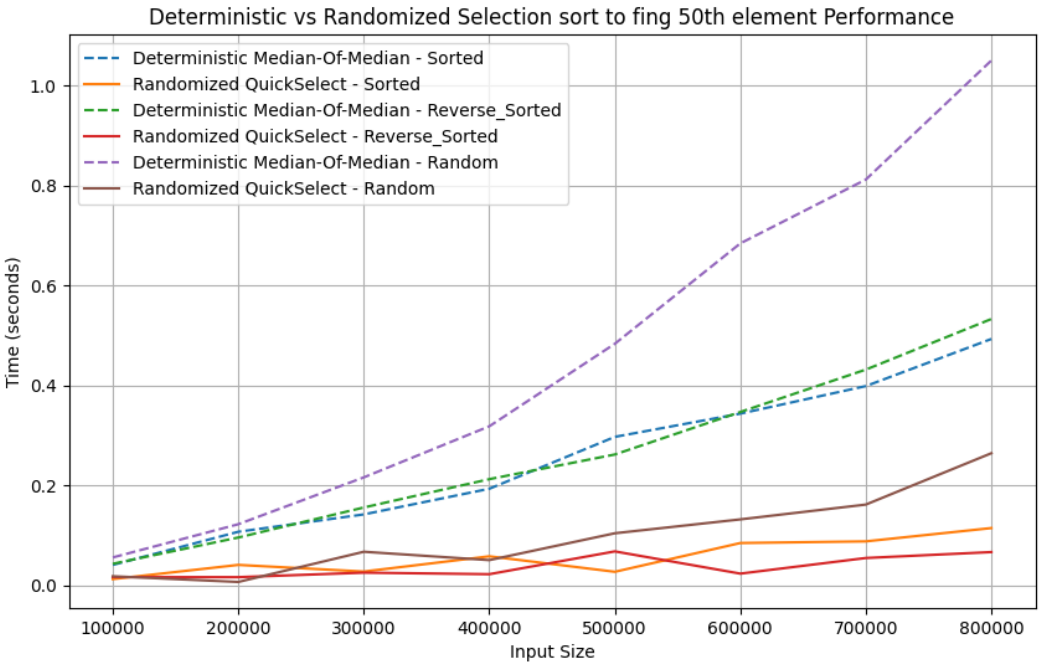
Worst Case: $O(n^2)$

Average and Best case: O (n)

As the pivot is always selected at random, the likelihood of the pivot always being the highest or lowest is very unlikely, thus it has the expected linear run time of O (n).

Performance Analysis

In the Python program, we took a data set of varying sizes between [100000, 800000] in sorted, reverse sorted, and random order. We took position 50 as our kth element and ran the selection algorithm against median-of-medians and randomized quickselect. The following figure gives the results.



The above figure shows that the randomized quickselect algorithm is much faster than the median-of-medians algorithm when the data size gets bigger. As the median algorithm's median needs to recursively sort and create a new array of medians to find the kth element, it does have additional memory overhead.

Part 2: Elementary Data Structures Implementation and Discussion

Arrays

In the Python program, an array was implemented by creating an array of fixed size and default value as None. It has methods such as insert, which inserts the element at the given index; delete, which deletes the element at the given index; and access, which returns the value of the array at the given index.

As for GeeksForGeeks (2023), an array's time and space complexity can vary depending on its dimensions. For a one-dimensional array, accessing an element by index takes O (1) time and space complexity. The index is directly linked to a memory location, so it takes constant time to access. If elements are inserted at the index, it takes O (1) constant time. Adding the element at the end of the array takes O (1) constant time; however, if the new element is to be added at the beginning of the array, it takes O (n) as it requires shifting all existing elements by index 1. The worst-case time to search for an element in the array is given by O (n), as it may require a loop through the whole array to obtain the element. Deleting an element from an array has the time complexity of O (n) as it may require shifting all the elements by one position.

An array can be used to store a list of values. In practical applications, an array can store a collection of objects. For example, all the company's customers can be stored in an array.

Stacks

Stacks are data structures where elements are stored in a last in, first out (LIFO) pattern. In the Python program, basic stack methods, such as push, pop, and peek, were implemented.

Pushing a new record to a stack takes O (1) time and space complexity. The pop method removes the topmost element, taking a constant time and space complexity as O (1). The peek refers to viewing the element at the top, which also happens in constant time of O (1) (“Time and Space Complexity analysis of Stack operations,” 2023).

Stacks are widely used while building function calls and recursive functions (Menon, 2020). The basic Windows commands, such as undo/redo, use stacks.

Queues

Queues are data structures where elements are stored in a first in, first out (FIFO) pattern. Enqueue, dequeue, and peek methods were implemented in the Python program.

The enqueue method, which refers to inserting a new element in the queue, takes a constant time O (1), as it simply inserts the new value at the last position. As queues are FIFO, the dequeue method removes the element at the index 0, so it also has the time complexity of O(1) (“Time and Space Complexity analysis of Queue operations,” 2024).

In practical applications, queues can be used in ticketing systems or tasks that must be performed sequentially.

Linked Lists

A linked list is a collection of nodes where each node has a value and a reference to the next node in sequence ("Time and Space Complexity of Linked List," 2024). The Python program has implemented linked list methods such as insert, delete, and traverse.

The time complexity of a linked list while inserting a new value depends on the position of the insertion. If inserted at the beginning, it has a time complexity of O (1), but if inserted at the end, it has a complexity of O (n). This is because, if inserted at the beginning, only the new reference has to be updated; however, if added to the last, it needs to go through all the existing reference nodes and add the value at last ("Time and Space Complexity of Linked List," 2024). The same goes for deletion; deleting from the beginning takes O (1), while from the last takes O(n). The search has the time complexity of O(n), as it may need to traverse all the nodes.

Linked lists are used in systems where dynamic memory allocation is needed. It is ideal for implementing stacks, queues, and graphs ("Applications of linked lists data structure," 2023). In the real world, it is used in applications to view images, play music, task scheduling, and file systems ("Applications of linked lists data structure," 2023).

Trade-offs between data structures

Choosing the correct data structure is essential for performance and memory efficiency. Depending on the tasks, one data structure may be preferred over another. For example, we have to store a list of products in an inventory management system. In this case, the product list can be stored using an array, linked list, or dictionary. Dictionaries may be the most applicable as they provide a constant time O (1) for insertion, deletion, and accessing the element using the key. However, the insertion takes O (1) for an array or linked list, but searching and accessing the element can take O (n). However, the same key cannot be used in a dictionary, so each product needs to have a unique key identifier. Linked lists take up less memory space when compared to other data structures; however, the complexity of search and retrieve increases when the data size is huge ("Data Structure Selection and Trade-offs," 2024). In conclusion, choosing the correct data structure by analyzing their trade-offs is necessary to achieve optimal time and space complexity.

References

Data Structure Selection and Trade-offs. All 38 AP subjects. (2024, July 19). https://library.fiveable.me/data-structures/unit-1/data-structure-selection-trade-offs/study-guide/4082pmaoQQ9sPH4L

GeeksforGeeks. (2023a, September 14). *Time and space complexity analysis of Stack Operations*. GeeksforGeeks. https://www.geeksforgeeks.org/time-and-space-complexity-analysis-of-stack-operations/

GeeksforGeeks. (2023b, November 3). *Time and space complexity of 1-D and 2-D array operations*. GeeksforGeeks. https://www.geeksforgeeks.org/time-and-space-complexity-of-1-d-and-2-d-array-operations/

GeeksforGeeks. (2024a, January 31). *Time and space complexity analysis of queue operations*. GeeksforGeeks. https://www.geeksforgeeks.org/time-and-space-complexity-analysis-of-queue-operations/

GeeksforGeeks. (2024b, July 31). *Time and space complexity of linked list*. GeeksforGeeks. https://www.geeksforgeeks.org/time-and-space-complexity-of-linked-list/

GeeksforGeeks. (2024c, August 14). *Quickselect algorithm*. GeeksforGeeks. https://www.geeksforgeeks.org/quickselect-algorithm/?ref=ml_lbp

Mao, L. (2024, February 18). *Median-of-medians selection algorithm*. Lei Mao's Log Book. https://leimao.github.io/blog/Median-of-Medians-Select-Algorithm/

Menon, A. (2020, October 6). *Stack Data Structure: Practical Applications & Operations*. Medium. https://medium.com/swlh/stack-data-structure-practical-applications-operations-e4e308008752

www.linkedin.com. (2023, March 29). *What are the advantages and disadvantages of median-of-medians algorithm?*. Median-of-Medians Algorithm: Pros and Cons for Quicksort. https://www.linkedin.com/advice/0/what-advantages-disadvantages-median-of-medians-algorithm