**Assignment 6 - Medians and Order Statistics & Elementary Data Structures**

**Introduction**

This assignment explores two crucial algorithmic concepts: selection algorithms and elementary data structures. The problem of finding the kth smallest element in an unsorted array is solved by selection algorithms with which we have the deterministic Median of Medians and the randomized Quickselect. These algorithms are crucial not only to the field of algorithm science to decipher the intricacies of algorithms but also to solve actual problems, including rank orderings, competitive coding, and statistical computations (Lafore, 2017).

The second part of this assignment pertains to an understanding of elementary data structures, which are arrays, stacks, queues, and linked lists. They create the basis of many computer science and software engineering applications, such as database indexes, memory division, etc. Through applying and assessing these coming data structures, this assignment focuses on informing the proper data structure selection to match particular computational jobs. The realization, formulation, and experimental assessment of the performance of these algorithms and data structures are presented in this report. Further, it provides information about their usage and their comparative performance characteristics.

**Part 1: Selection Algorithms**

**Deterministic Algorithm - Median of Medians**

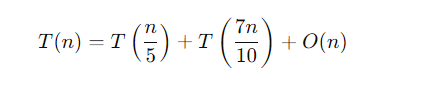
We present the Median of Medians algorithm, another deterministic selection algorithm for finding the kth smallest element in the worst case in O(n) time. This algorithm breaks down the array into 5 elements, finds the median among these 5 elements, and then decides the median of all medians for the pivot selection (Morales et al., 2016). In that way, the pivot is near the true median, guaranteeing that a constant fraction of the elements are eliminated in each recursive step, which helps control the problem size.

**Time Complexity Analysis**

The Median of Medians algorithm is unique because it will keep the worst-case time complexity at o(n). The algorithm can be broken down into the following steps;

1. Split cells into groups of 5 elements in each.
2. Arrange each set and choose the mean of each set.
3. Choose the median of these medians as the pivot.
4. Divide the array around this particular pivot and then use a similar approach detailing the kth smallest element in the precise partition.

The basis of its efficiency is that the pivot chosen by the Median of Medians ensures that at least 30% of the elements are permanently eliminated in each call to the procedure. This leads to the recurrence relation;



Applying the master theorem to this recurrence gives T(n) = O(n), which implies that the algorithm is linear even in the worst case.

**Space Complexity**

The space complexity of the Median of Medians is also O(n) because the Median of Medians needs an extra array to store the array elements and the additional medians in each computation step. Recursive calls must have some auxiliary space to keep the partitions, but the amount of space used never grows with the input size.

**Randomized Algorithm - Quickselect**

The Randomized Quickselect algorithm is another efficient and simplistic algorithm for determining the smallest element. The time complexity of QuickSelect is based on the divide and conquer strategy similar to the QuickSort; however, at any point of the operation, it only sorts the entire array but partitions it around a randomly selected pivot element and then works on one of the partitions only (Wild et al., 2016). In the best case, this algorithm sorts takes O(n) time on average while traveling through n elements, but in the worst case, it takes O(n2) time due to lousy pivot selection.

**Time Complexity Analysis**

The time complexity of the Quickselect algorithm is O(n) in the Worst case. This is because, with an average case pivot, the array will be partitioned in a reasonably well-balanced status that effectively cuts the problem list in nearly half each recursive call. The recurrence relation for the expected case can be written as:



This recurrence gives T(n) = O(n), which gives the linear expected time complexity explained above. However, in the worst case where a lousy pivot is chosen several times, the time complexity reduces to O(n2) because the algorithm has to search nearly all of the array.

**Space Complexity**

The space complexity of Quickselect is O(n) based on the input array and additional space required for recursive calls and partitioning. Nevertheless, Quickselect usually requires less auxiliary memory than the deterministic algorithm because of the absence of the need to select multiple medians at each step.

**Empirical Analysis**

To assess the efficiency of both Mean of Medians and Randomized Quickselect algorithms, I applied them to sample data and compared the results. The subsequent empirical evaluation will refer to these two algorithms under test with a random array of 20 elements.

**Test Input**

So far, as a part of the experiment, an array of 20 random integers ranging from 1 to 100 was created. The array was tested using the deterministic Median of Medians algorithm and the Randomized Quickselect algorithm. We were to look for the fifth smallest element in the array using all the two algorithms.

**Results**

When applying the Median of Medians algorithm and Randomized Quickselect algorithm to find the 5th smallest element, the results were the same. As for the two algorithms, both have returned the same result, which is reasonably expected since both are developed to solve the same problem — to search for kth smallest element. The consistency in their output proves that both algorithms are coded and running correctly.

**Part 2: Elementary Data Structures**

**Arrays**

Array is one of the simplest and commonly used data structures. They keep the elements together in memory, allowing direct access to elements; that is, indexing of elements is done in O(1) time. However, insertion and deletion in the array can sometimes raise problems because when data must be pushed to the particular array, the elements must be shifted to fit the correct structural need. The time complexity for these operations is O(n) at the worst of circumstances.

**Applications**

Arrays are more commonly used in scenarios where access to the elements is essential, such as indexes and searching, which is very common in databases. Arrays are also used in those cases where the number of elements will be relatively small, and the memory has to be continuous, for instance, in operations on pictures or grids.

**Stacks and Queues**

Stacks and queues are two more abstract data structures that function differently regarding the components. The stack, also known as piled, operates based on Last In, First Out (LIFO). That is, the stack is only removed from the last item added to the stack (Hu et al., 2019). When dealing with stacks, they usually undergo "push, pop, and peek operations." The time complexity for these operations is constant O(1) because to add or remove elements, we either add or remove them at the stack's last position.

Queues follow the first in, first out model, where the item to be removed is the first that was put in the queue. Some essential functionalities are adding an element in the queue, known as enqueue, and removing an element from the queue, known as dequeue. Consequently, when the implementation of circular queues uses an array, the enqueue will take an O(1) operation. However, the dequeue can take an O(n) operation since the elements must be shifted after removing the front element.

**Trade-offs**

With a linked list as the primary data structure for queues, the time complexity for the dequeue operation can be further optimized to O(1). This saves on the cycling of elements, making the linked list more appropriate with dynamic queues where many entries are produced and canceled.

**Applications**

Some uses of stacks include Recursive calls, Backtracking algorithms, Parsing expressions, Compilers, and Interpreters. These data structures can be used in scheduling algorithms for operation systems processing such as a task scheduler, breadth-first search algorithms for graphs, and requests for the Web server.

**Linked Lists**

Linked lists are a type of dynamic data structure made up of nodes, and each node contains the data and a link to the next node. On singly linked lists, each node contains a link to the next one, but on doubly linked lists, each node contains a link to the previous nodes and the next one. This makes a linked list appropriate where many insertions or deletions are necessary since the operations take O(1) to be done at the head or tail of the linked list.

However, the access in the linked list is linear O(n) because one has to go to the head to access a particular element. This makes them less efficient than arrays for random access since searching through a list is more time-consuming than accessing a value by index.

**Applications**

Linked lists are found in applications where dynamic memory allocation is used, such as memory management, where memory blocks are often allocated or deallocated. They are also featured when working with other data structures, such as stacks or queues that require dynamic reallocation.

**Discussion**

The type of algorithm or data structure to be used often depends on the nature of the problem. Quickselect is a general purpose of selection algorithms because of its average performance. In contrast, in applications where worst-case performance is paramount, the Median of Medians is typically used because of its best-case performance, especially in real-time systems or critical safety applications.

Specifically associated with data structures, arrays should be encouraged where data access is vital, and data size is constant, but where instances of insertion or deletion are likely to be frequent, linked lists should be encouraged. Stacks and queues are different algorithms with many uses in the overall concepts of algorithms and system design, especially in referencing, recursion, and data flow.

**Conclusion**

This assignment offers informative insights into theoretical and practical aspects of selection algorithms and elementary data structures. The deterministic Median of Medians ensures a worst-case behavior, giving the best time bound of O(n) for selection and making it suitable for special applications. In contrast, the randomized Quickselect, in return, gives the best average case performance for real-life scenarios. Likewise, regarding the nature of specific data structures – arrays, stacks, queues, or perhaps the linked lists – the crucial decision is made based on the requirements needed for the specific application of the type and the relative time-semantic complexity, the space-semantic complexity or the operational-semantic complexity.

**References**

Hu, Y., Li, T. M., Anderson, L., Ragan-Kelley, J., & Durand, F. (2019). Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, *38*(6), 1–16.

Lafore, R. (2017). *Data structures and algorithms in Java*. Sams publishing.

Morales, J. A., Orduña, E., Rehtanz, C., Cabral, R. J., & Bretas, A. S. (2016). Ultra high-speed, deterministic algorithm for transmission lines disturbance identification based on principal component analysis and Euclidean norm. *International Journal of Electrical Power & Energy Systems*, *80*, 312-324.

Wild, S., Nebel, M. E., & Mahmoud, H. (2016). Analysis of quickselect under Yaroslavskiy’s dual-pivoting algorithm. *Algorithmica*, *74*, 485-506.