Assignment 6: Medians and Order Statistics & Elementary Data Structures

Safal Poudel

Algorithms and Data Structures (MSCS-532-B01)

University of the Cumberlands

GitHub Link: https://github.com/poudels54443/Assignment6

Implementation and Analysis of Selection Algorithms

The deterministic median-of-medians algorithm divides the input into groups of five, sorts each group, and collects their medians. By recursing on the list of medians to choose a pivot, it guarantees that at least thirty percent of the elements lie on each side of the pivot, satisfying the recurrence

$T(n) \leq T(n/5) + T(7n/10) + O(n)$,

which resolves to O(n) in the worst case (Rauh & Arce, 2012). The randomized Quickselect algorithm chooses a pivot uniformly at random and partitions the array into elements less than, equal to, and greater than that pivot. In expectation the pivot splits the data into roughly equal halves, yielding

$E[T(n)] = E[T(n/2)] + O(n)$,

and thus $E[T(n)] = O(n)$ (Martínez, Panario, & Viola, 2004). Both implementations validate index bounds and correctly handle duplicate values by grouping equal-to-pivot elements before recursing.

Empirical Performance Comparison

Input Size (n)	Distribution	Deterministic Select (ms)	Randomized Quickselect (ms)
1000	random	0.523	0.229
1000	sorted	0.431	0.296
1000	reverse	0.450	0.297
5000	random	3.195	1.159
5000	sorted	2.144	0.742

5000	reverse	2.256	0.619
10000	random	5.472	2.548
10000	sorted	4.322	2.173
10000	reverse	4.384	3.779

These measurements confirm that randomized Quickselect consistently outperforms median-of-medians in wall-clock time while both algorithms scale linearly. Quickselect's lower constant factors make it preferable when average performance matters, whereas median-of-medians remains the choice for strict worst-case guarantees.

Elementary Data Structures Implementation and Discussion

The DynamicArray class begins with capacity one and doubles its size when full, halving when elements fall below one quarter of capacity. This resizing strategy yields amortized O(1) appends while preserving O(1) element access. The Matrix class stores data in a list of lists, offering constant-time cell access and O(r×c) time for row or column insertions and deletions. A Stack wraps the dynamic array to support push, pop, peek, and emptiness checks in amortized constant time. A Queue uses a circular buffer over a Python list to achieve amortized constant-time enqueue and dequeue operations with automatic resizing. The LinkedList class implements singly linked nodes with head and tail insertion in O(1) and value-based deletion or full traversal in O(n). Finally, the TreeNode class models a rooted tree whose preorder traversal visits each node exactly once in O(n) time.

Dynamic array amortized costs and performance characteristics have been studied in fine-grained algorithm analyses that integrate high-precision timing into compiler frameworks

(Haslbeck et al., 2022). Succinct and dynamic tree representations demonstrate how child-pointer lists can be stored and traversed efficiently within tight memory bounds while supporting updates in logarithmic time (Navarro & Sadakane, 2014). Stacks and queues built on dynamic arrays benefit from contiguous locality, and linked lists provide guaranteed constant-time head operations at the cost of pointer overhead and linear-time tail operations or searches. These trade-offs guide developers in choosing the most suitable structure for workloads ranging from memory-constrained embedded systems to high-throughput server applications.

References

Martínez, C., Panario, D., & Viola, A. (2004). Adaptive sampling for Quickselect. In Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA ’04) (pp. 440–448).

Rauh, A., & Arce, G. R. (2012). Optimal pivot selection in fast weighted median search. IEEE Transactions on Signal Processing, 60(8), 4108–4121.

Haslbeck, M. P. L., et al. (2022). Verified fine-grained algorithm analysis down to LLVM. Proceedings of the 43rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’22), 1011–1026

Navarro, G., & Sadakane, K. (2014). Fully-functional static and dynamic succinct trees. ACM Transactions on Algorithms, 10(3), Article 16.