What is your programming partner's name:XIAOHAN CHEN

which of you submitted the program files to Gradescope?: ME, YIJUN

Briefly discuss the pair programming experience, including:

1. Technique Application: Xiaohan and I effectively used pair programming techniques. She completed Part 1,tests of p1,p2, I did Part 2,3 and we jointly debugged and did tests.

2. Contribution: We split our tasks well, complementing each other's work.

2. Time and Evaluation: We spent approximately 8 hours completing the programming and testing portion of this assignment. Our sessions were spread over 3 days, with each session lasting about 3h with balanced contributions.

4. Future Plans: Open to partnering with Xiaohan again, our collaboration was productive.

PUSH METHOD PERFORMANCE

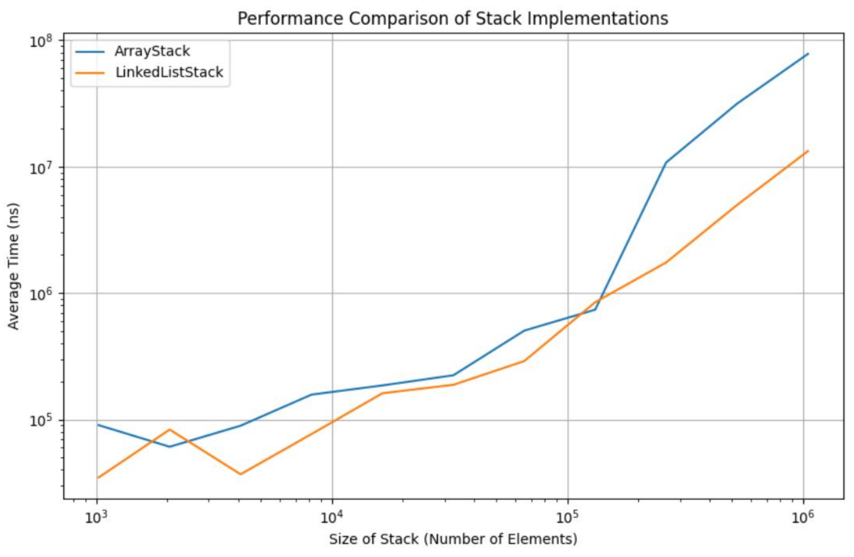

Table 1 Push method

ArrayStack:

The performance curve for ArrayStack shows non-linear growth in the average time required as the size of the stack increases, particularly noticeable at larger scales. This behavior is typically due to the resizing operation that occurs when the array reaches its capacity limit. Resizing involves allocating a new, larger array and copying over the existing elements, which is an O(n) operation. Consequently, each resizing operation causes a spike in the performance time.

Additionally, as the stack size continues to increase, the frequency of resizing decreases, but the cost of each resize is higher due to the larger number of elements that must be copied.

LinkedListStack:

The performance curve for LinkedListStack is relatively smoother, suggesting a more linear relationship between performance and stack size. This is because each push operation in a linked list is O(1), independent of the number of current elements. However, as the stack size increases, the overall running time also increases gradually, possibly due to the overhead associated with memory allocation and garbage collection.

PEEK METHOD PERFORMANCE

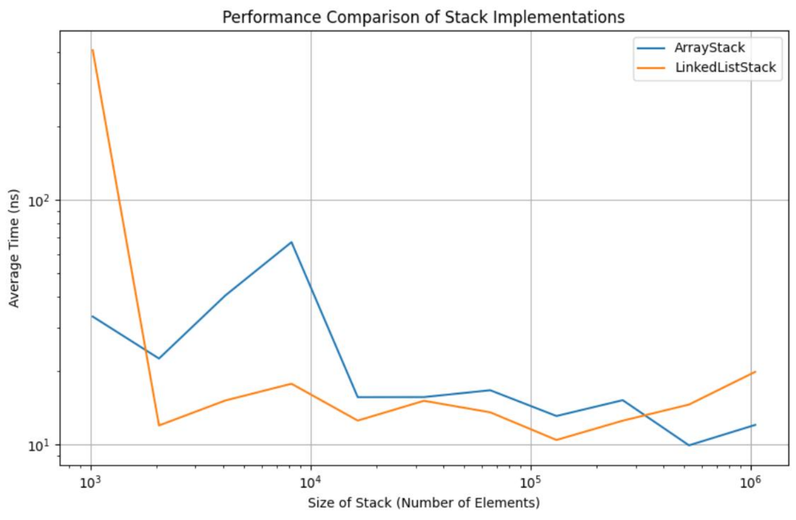

Table 2 Peek method

The peek() operation should typically be constant time O(1) for both types of stacks because it only involves accessing the top element without any modifications to the stack's structure.

However, the graph exhibits some fluctuation, especially with LinkedListStack. Here are a few points to consider:

1. JVM Warm-up: The Java Virtual Machine (JVM) optimizes the code as it runs, which can cause variations in the early stages of performance testing. This is why a warm-up period is often used before taking measurements.

3. Garbage Collection: The JVM performs garbage collection at indeterminate times, which can momentarily affect performance measurements.

4. System Load: Other processes running on the system can interfere with the timing of the code, leading to spikes and drops in the measured times.

POP METHOD PERFORMANCE

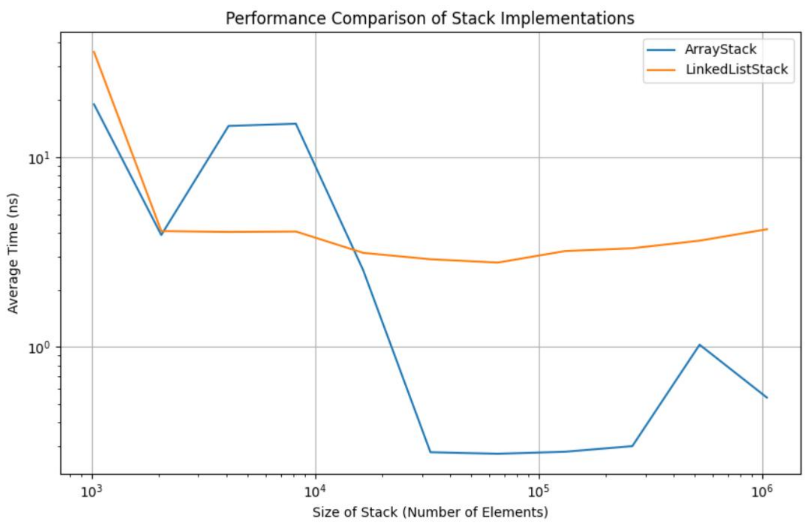

Table 3 Pop method

Both in arrayStack and Linked Stack, pop( ) is O(1). Because this operation involves removing the top element from the stack, which does not require traversing the entire structure or resizing the underlying array or list.

But some factors can affect actual performance:
Because this operation involves removing the top element from the stack, which does not require traversing the entire structure or resizing the underlying array or list. Here's a breakdown:

Memory Allocation: In some LinkedListStack implementations, pop() might involve deallocating the node which was at the top of the stack, depending on the garbage collection strategy and how memory allocation is handled in the language's runtime environment. This doesn't change the theoretical complexity but can affect practical performance.

Data Locality: ArrayStack might still have better real-world performance due to better data locality and caching behavior. Access patterns that are sequential in memory (like those in an array) are generally faster due to the way modern CPUs cache memory.