D0012E Lab1 Grupp 11:

Elias Grundberg	eligru-3@student.ltu.se	010314
Hjalmar Norén	hjanor-3@student.ltu.se	041029
Pablo Zepada Garcia	pabzep-3@student.ltu.se	021107

2024-11-21

Theory Behind the Algorithm:

The program is meant to sort a stack of numbers using a temporary empty stack. It starts with the second currently empty stack taking elements from the first unsorted stack and inserting them in a descending order ensuring the correct placement of elements. The algorithm essentially pops elements from the first stack and inserts them in the second at the correct position; larger elements in the second stack will be temporarily moved back to the first stack to make space for smaller elements. This results in a worst case time complexity of $O(n^2)$ due to nested loops and a space complexity of O(n) for the temporary stack.

The algorithm used here is Insertion Sort. Bubble Sort was tried but found to not be as fast. Merge sort was also tried but requires more than 1 stack to be properly used so it does not satisfy the requirements of the Lab. Hence Insertions Sort was chosen due to satisfying the needed criteria and being the fastest option we found. While relatively fast it is still extremely slow when using large datasets, sorting 100k numbers in a worst case scenario takes about an hour depending on the speed of the computer and other small factors.

C is a constant that represents the speed of which the computer computes.
Worst case:

$$T(n) = c_1 n + c_2 \sum_{i=1}^{n-1} i + c_3 n = n + \frac{(n-1)(n-2)}{2} + n$$

```
while not stack1.isEmpty():                                          cost    times
    temp = stack1.pop()                                              c1      n
    while not stack2.isEmpty() and stack2.content.data > temp:
        stack1.push(stack2.pop())                                    c2      sum_{i=1}^{n-1} i
        compares += 1
    stack2.push(temp)
while not stack2.isEmpty():
    stack1.push(stack2.pop())                                        c3      n
```

Run cases:

Size	Worst Time
1000	0.329102
10 000	33.809793s
50 000	846.096449s
100 000	3360.619308s

Aside from the code itself the run times are dependent on the performance of the hardware and the allocation of resources so each run can have some minor differences.

Code Components:

Node Class: Represents a single element in a stack implemented as a linked list. There are two attributes, Data stores the value of the node and Next points to the next node in the stack. It is a class that provides the necessary building blocks for the Stack class.

Stack Class: Implements a stack using a linked list. Stack contains the attribute Content that represents the top of the stack and four methods; Push that adds an element to the top of the stack, Pop that removes and returns the top element of the stack, IsEmpty that checks if the stack is empty and SortList that returns the elements of the stack as a list (Used only for Visualization and Debugging, not sorting).

Sorting Algorithm: A function that sorts the elements of stack1 using a second empty stack2 as temporary storage. The process goes:

- Continuously pops elements from stack1 and compares them to the top of stack2.
- Moves larger elements from stack2 back to stack1 to maintain correct order.
- Once stack1 is empty, all elements are stored in sorted order in stack2.
- Moves the elements back to stack1 for the final result.

Main Function: Initialises the first stack and fills it with data then outputs the stack's content before and after the sorting. It also measures and prints runtime performance eg. the number of operations that have been used (Push, Pop and Compares).

Potential Improvements:

Add validation: Instead of looking through the results to see if they are sorted we could add some validation code that does it automatically.
Explore more sorting techniques: We have only looked at 3 different sorting algorithms, 1 of which wouldn't be acceptable for this Lab but there is potentially a better sorting algorithm than Insertion Sort that still only uses 2 stacks to sort all numbers.