Question 3.5 from CTCI: Sort Stack

February 28, 2018

Question

Write a program to sort a stack such that the smallest items are on the top. You can use a additional temporary stack, but you may not copy the elements into any other data structure (such as an array). The stack supports the following operations: *push*, *pop*, *peek*, and *isEmpty*.

Explanation and Algorithm

While there is a different approach to sorting stacks in this manner, we are limited to using one additional stack to aid us in sorting the input stack. We can use this additional stack to maintain a sorted stack in reverse order and then push the values back onto our input stack, achieving the desired result. This can be accomplished in the following way:

For each value in the input stack, we use the *pop* operation to remove it and store the value in a temporary variable. We then push this value onto the temporary stack so that it is sorted with the largest value at the top and the smallest value at the bottom. If the temporary stack is empty or the value removed from the input stack is larger than the value at the top of the temporary stack, we can just push the value onto the temporary stack and continue removing the next value from the input stack. However, if the value removed is smaller than the value at the top of the temporary stack, we will then need to sort it within that stack. Since the removed value is stored in a temporary variable, we can pop the value at the top of the temporary stack and push it onto the input stack again. We continue doing this until we have found the correct position for our stored value and then we push that onto the temporary stack. We continue this process until the temporary stack is completely sorted and the input stack is empty. We then push the values from the temporary stack back onto the initial stack, producing a sorted stack such that the smallest items are on top.

Hints

1. Consider using the temporary stack to hold the values in sorted order. What order should these values be sorted in? Where should the smallest elements be?
2. Try maintaining the temporary stack in descending order, such that the largest values are at the top and the smallest values are at the bottom. Why might you want to sort the temporary stack this way?
3. How would you handle inserting a value that is smaller than the value at the top of the temporary stack?
4. If you haven't already, try using a temporary value to hold each value popped from the input stack. Is there a way you can remove the values that are larger than this in the temporary stack so that you can insert it in its proper position?

Code

```
/* Solution */
public void sort(Stack<Integer> s){
   Stack<Integer> r = new Stack<Integer>();
   while(!s.isEmpty()){
      int tmp = s.pop();
      while(!r.isEmpty() && r.peek() > tmp){
         s.push(r.pop());
      }
      r.push(tmp);
   }
   while(!r.isEmpty()){
      s.push(r.pop());
   }

   return;
}
```

Big O Analysis

1. The running time is $O(n^2)$ where n is the size of the input stack.

Sources

Question, answer and other material taken from Cracking the Coding Interview 6th edition by Gayle Laakmann McDowell.