Strange Stack

Riko Jacob

Thu Mar 7 11:10:49 2019 +0100, rev. 8ca5f11

Description

This assignment is an exam question about understanding a given piece of code. The points for the different parts of the questions are irrelevant for the grading of the assignment, they are merely for you orientation the weights in the exam. In the exam situation, you will have to answer this kind of question **without access to a computer**, so we strongly suggest that you don't cheat yourself and solve the questions with only pen and paper. Observe that Question (f) is almost pointless if you have a computer to execute the code, and is hence not relevant for passing the assignment.

Deliverable

A. A PDF with your answers. Please add all the names and itu-email addresses of the group members as authors. Resembling the exam situation, hand-written and scanned answers are particularly welcome.

***Question 1**: from exam 180528 Question 2*

The next few questions all concern the class defined in fig. 1 and 2.

(a) (1Pt) Class *S* is some kind of fixed-capacity stack-like data structure. When an object of class *S* runs out of capacity, it

A. forgets the oldest element.

B. ignores the next push.

C. resizes itself so as to make room for more elements.

D. returns the smallest element.

```java
import edu.princeton.cs.algs4.*;

public class S
{
    public int[] a;
    int sz;

    public S(int cap)
    { a = new int[cap]; sz = 0;}

    public void push(int item)
    {
        if (sz == a.length) shift();
        a[sz] = item;
        sz++;
    }
    public int pop()
    {
        sz--;
        return a[sz];
    }

    private void shift()
    {
        int[] tmp = new int[a.length];
        for (int i = 0; i<a.length - 1; i++) tmp[i] = a[i+1];
        a = tmp;
        sz--;
    }
}
```

Figure 1: Class S (for Strange Stack), java version.

```python
class S:
    def __init__(self, capacity):
        self.a = [0] * capacity;
        self.sz = 0

    def push(self, item):
        if self.sz == len(self.a):
            self.shift()
        self.a[self.sz] = item
        self.sz += 1

    def pop(self):
        self.sz -= 1
        return self.a[self.sz]

    def shift(self):
        tmp = [0] * len(self.a)
        for i in range(0, len(self.a) - 1):
            tmp[i] = self.a[i+1]
        self.a = tmp
        self.sz -= 1
```

Figure 2: Class S (for Strange Stack), python version.

(b) (1Pt) What is the result of the following operations?

```
// java                                    # python3
int N = 4;                                 N = 4;
S s = new S(N);                            s = S(N)
for (int i = 0; i < N    ; i++)            for i in range(N):
    s.push(i);                                 s.push(i)
for (int i = 0; i < N    ; i++)            for i in range(N):
    StdOut.print(s.pop());                     stdio.write(s.pop())
for (int i = 0; i < N + 1; i++)            for i in range(N + 1):
    s.push(i);                                 s.push(i)
for (int i = 0; i < N    ; i++)            for i in range(N):
    StdOut.print(s.pop());                     stdio.write(s.pop())
```

(Ignore whitespace such as blanks or newlines.)

(c) (1Pt) Draw the data structure after the following operations: (This means "*at the end* of the operations," not "*after each* operation," so you need to draw only a single picture. Make sure you draw the entire data structure. Make sure to include all instance variables.)

```
// java                    # python 3
S s = new S(3);            s = S(3)
s.push(5);                 s.push(5)
s.push(6);                 s.push(6)
s.pop();                   s.pop()
```

(d) (1Pt) Assume I initialised an *S*-object with capacity *K*. Now I perform a sequence of *N* pushes. What is the worst-case (also the worst possible dependency of *K* on *N*) running time of a single `push`? (Choose the smallest correct estimate.)

A. $O(\log N)$, i.e., logarithmic in N

B. $O(N)$, i.e., linear in N

C. $O(N \log N)$, i.e., linearithmic in N

D. $O(1)$, i.e., constant, independent of N

(e) (1Pt) Assume I initialised an *S*-object with capacity *K*. Now I perform a sequence of *N* pushes, followed by *N* pops. What is the worst-case (also the worst possible dependency of *K* on *N*) running time of a single `pop`? (Choose the smallest correct estimate.)

A. $O(\log N)$, i.e., logarithmic in N

B. $O(N)$, i.e., linear in N

C. $O(N \log N)$, i.e., linearithmic in N

D. $O(1)$, i.e., constant, independent of N

(f) (1Pt) Add a method with the following signature that checks if the data structure is empty.

```
// java                          # python 3
public boolean isEmpty()         def is_empty(self):
```

Don't change any other methods in class `S` *and don't introduce new instance variables to* `S`*. Write actual code. In the exam situation, minor syntax mistakes will not count heavily.*

(g) (1Pt) Extend class `S` with the following signature that removes the element that was first pushed (i.e., the 'oldest' element). The method returns nothing. *Don't change any other methods in class* `S` *and don't introduce new instance variables to* `S`*. Write actual code. In the exam situation, minor syntax mistakes will not count heavily.*

```
// java                          # python 3
public void remove_oldest()      def remove_oldest(self):
```

(h) (2Pt) Explain how to change the implementation of the data structure so that push and pop take constant time in the worst case. The functionality must be unchanged. If K is the capacity, then the constructor should take $O(K)$ time and the data structure should take $O(K)$ space.