Data Structures and Algorithms: Homework #3

Due on April 28, 2015 at 16:20

Instructors Hsuan-Tien Lin, Roger Jang

Tim Liou (b03902028)

3.1 Asymptotic Complexity

(1) Do Exercise R-4.28 of the textbook.

Consider $a_k \geq 0 \ for \ k \geq 0$

$$p(n) = a_0 + a_1 n + a_2 n^2 + a_3 n^3 + \cdots + a_m n^m$$

For $n \geq (a_0 + a_1 + a_2 + \cdots + a_m) \geq 1$, we have

$$\begin{aligned} p(n) &\leq (a_0 + a_1 + a_2 + \cdots + a_m) \times n^m \\ \Rightarrow \log p(n) &\leq \log(a_0 + a_1 + a_2 + \cdots + a_m) + m \log n \\ &\leq \log n + m \log n \\ &= (m+1) \log n \end{aligned}$$

Take $c = m + 1 > 0$, $n_0 = (a_0 + a_1 + a_2 + \cdots + a_m) \geq 1$

$$\log p(n) \leq c \log n \quad for \ n \geq n_0$$

That is, $\log p(n)$ is $O(\log n)$.

(2) Do Exercise R-4.34 of the textbook.

We have $f(n) > 1$ and $\lceil f(n) \rceil \leq f(n) + 1$ by definition. For $n \geq 1$,

$$\begin{aligned} \lceil f(n) \rceil &\leq f(n) + 1 \\ &\leq f(n) + f(n) \\ &= 2f(n) \end{aligned}$$

Take $c = 2 > 0$, $n_0 = 1 \geq 1$

$$\lceil f(n) \rceil \leq cf(n) \quad for \ n \geq n_0$$

That is, $\lceil f(n) \rceil$ is $O(f(n))$.

(3) Prove that $f(n) = \Theta(g(n))$.

By definition of limits at infinity,

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = A$$

means that for every $\epsilon > 0$ there is a corresponding N such that

$$\left| \frac{f(n)}{g(n)} - A \right| < \epsilon \quad for \ n > N$$

That is,

$$A - \epsilon < \frac{f(n)}{g(n)} < A + \epsilon \quad for\ n > N$$

Note that $g(n)$ is a strictly positive function. We have

$$(A - \epsilon)g(n) < f(n) < (A + \epsilon)g(n) \quad for\ n > N$$

Take $\epsilon \in (0, A)$, $c_1 = (A - \epsilon) > 0$, $c_2 = (A + \epsilon) > 0$, $n_0 > N$

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \quad for\ n > n_0$$

This shows that $f(n) = \Theta(g(n))$.

(4) Do Exercise R-4.8 of the textbook.

If A is better than B for $n \geq n_0$, n_0 satisfies the following statement.

$$2{n_0}^3 - 40{n_0}^2 > 0$$

We can easily find that $n_0 > 20$. We choose $n_0 = 21$. It is a possible value for n_0 satisfying the statement that A is better than B for $n \geq n_0$.

(5) Do Exercise C-4.16(b) of the textbook.

This is the pseudo code of the Horner's method.

```
1: function HORNER'S-METHOD(x, CoefficientsOfPolynomial, DegreeOfPolynomial)
2:     Sum ← 0
3:     for all CoefficientsOfPolynomial do
4:         Sum ← Sum × x + CoefficientsOfPolynomial
5:     end for
6:     return Sum
7: end function
```

Algorithm 1: Horner's method for computing polynomial

We can find there is only one **for loop** in this pseudo code, that is, the number of arithmetic operations is $O(n)$.

(6) Consider some $f(n)$ and $g(n)$ such that $\lg f(n) = O(\lg g(n))$ and $g(n) \geq 2$ for $n \geq 1$. Construct a counter-example to disprove that $f(n) = O(g(n))$.

Consider $f(n) = 4^n$, $g(n) = 2^n$, for $n \geq 1$, we can find

$$\begin{aligned}\lg f(n) &= n2 \lg 2 \\ &\leq n4 \lg 2 \\ &= 4 \lg 2^n \\ &= 4 \lg g(n)\end{aligned}$$

Take $c = 4 > 0$, $n_0 = 1 \geq 1$

$$\lg f(n) \leq c \lg g(n) \quad for\ n \geq n_0$$

That is, $\lg f(n)$ is $O(\lg g(n))$. Note that $g(n) \geq 2$ for $n \geq 1$.

If $f(n) = O(g(n))$, $\exists n_0 > 0, c > 0 \ni$

$$4^n \leq c2^n \quad for\ n \geq n_0$$

Take $\log_2$ on both sides,

$$\begin{aligned}&2n \leq \log_2 c + n \quad for\ n \geq n_0 \\ &\Rightarrow \log_2 c \geq n\end{aligned}$$

That is, take $n' = max(n_0, \lceil \log_2 c + 1 \rceil)$

$$\begin{aligned}n' \geq n_0 &\Rightarrow 4^n \leq c2^n \\ n' > \log_2 c &\Rightarrow 4^n > c2^n\end{aligned}$$

This is a contradiction. Therefore, we disprove that $f(n) = O(g(n))$.

3.2 Stack, Queue, Deque

(1) Do Exercise C-5.2 of the textbook.

Pop out the elements in the stack one by one and check if it is equal to element x. After that, enqueue the elements into the queue one by one. Use a varible to store the number of the elements we poped from the stack. Once we find the certain element or the stack is empty, we push the elements into stack from queue, and then enqueue the same number of element into queue from stack. Finally push all these elements from queue into stack again. This will maintain elements' original order.

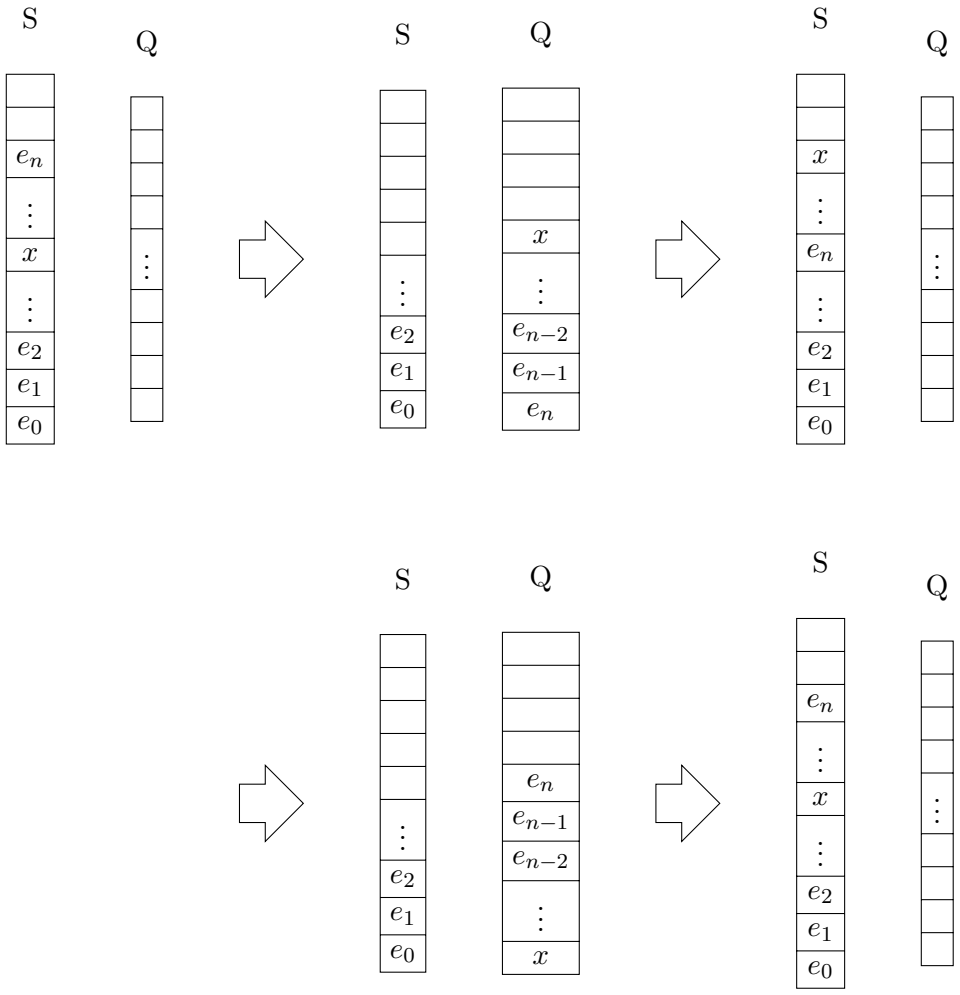

Figure 1: How this algorithm works

(2) Do Exercise C-5.9 of the textbook.

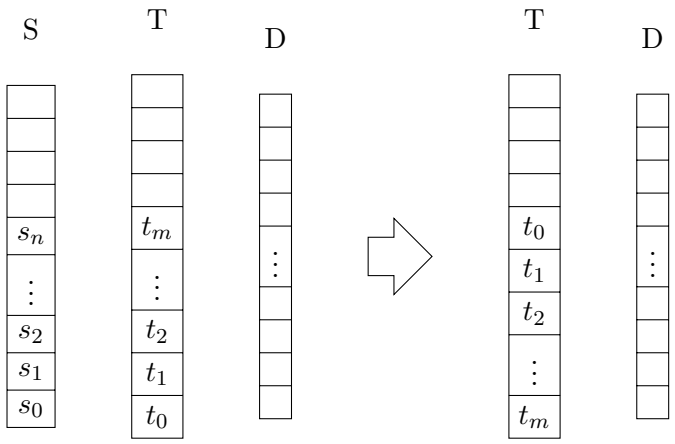

Figure 2: **Pop** all elements from T and **Push_front** them to D. Then **Pop_back** them back to T.

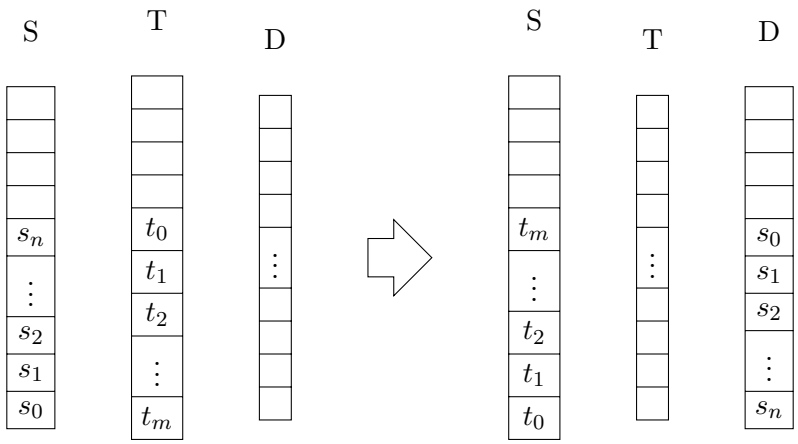

Figure 3: **Pop** all elements from S and **Push_front** them to D. Then **Pop** all elements from T to S.

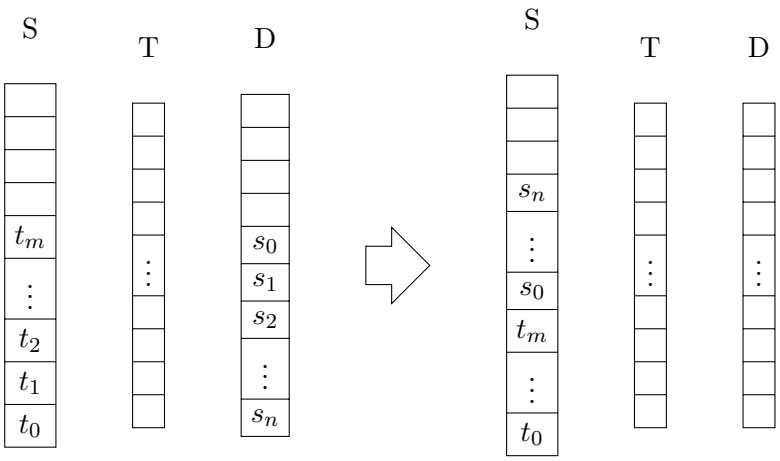

Figure 4: **Pop_front** all elements from D to S

(3) Use any pseudocode to write down an algorithm that uses two stacks (with push, pop and isempty operations but no others) to simulate one deque (for push/pop front and push/pop back operations). What is the total running time after N operations?

====== Pending ======

(4) Do Exercise C-5.9 of the textbook, but with three stacks instead of two stacks and one deque.

Suppose three stacks are big enough for all elements.

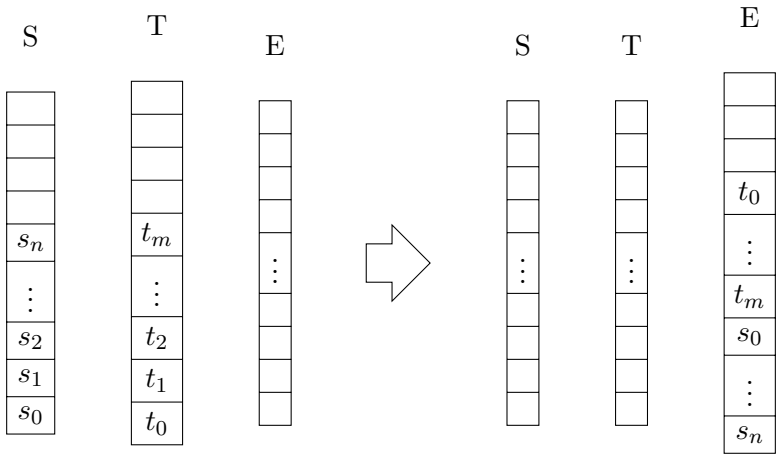

Figure 5: **Pop** all elements from S to E and then **Pop** all elements from T to E.

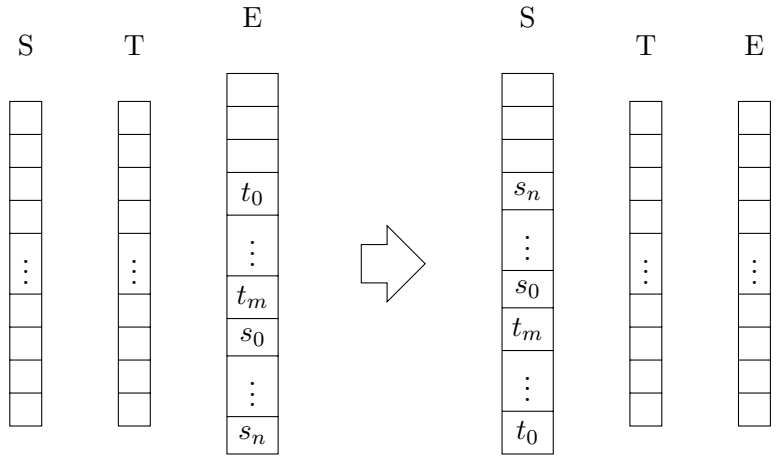

Figure 6: **Pop** all elements from E to S

3.3 List, Iterator

(1) Do Exercise C-6.7 of the textbook.

====== Pending ======

(2) Do Exercise C-6.13 of the textbook by Googling the *Knuth Shuffle*.

```
1: function KNUTH-SHUFFLE(V, LengthOfV)
2:     for i ← 0 to LengthOfV − 1 do
3:         r ← RANDOMINTGER(i + 1)
4:         Exchange V[i] and V[r]
5:     end for
6: end function
```

Algorithm 2: Knuth-Shuffle

Knuth Shuffle guarantees that every possible ordering is equally likely. The running time of this function is $O(n)$, n is the number of cards.

3.4 Calculators