Solution for Problem Set 1

20130005 方盛俊

Problem 1

(a) We need to prove that A' is a permutation (reordering) of A.

(b)

- Loop invariant: After the j-th loop, $A[j-1]$ is the smallest element of $A[j-1...n]$.
- Proof:
  - Initialization: There is only one element, $A[n]$, when $i = 1$.
  - Maintain: $A[j]$ is the smallest element of $A[j...n]$. After exchange, $A[j-1]$ is the smaller than $A[j]$ and all elements in $A[j+1...n]$ so that $A[j-1]$ is the smallest element of $A[j-1...n]$, when $j \leftarrow j-1$.
  - Termination: $A[i]$ is the smallest element of $A[i...n]$, when $j = i$.

(c)

- Loop invariant: After the i-th loop, subarray $A[1...i]$ is sorted and $A[i]$ is the smallest element of $A[i...n]$.
- Proof:
  - Initialization: There is only one element, $A[1]$, when $i = 1$.
  - Maintain: $A[i-1]$ is the smallest element of $A[i-1...n]$. After loop in lines 2-4, with the loop invariant proved in part (b), $A[i]$ is the smallest element of $A[i...n]$ and $A[1...i]$ is still sorted, when $i \leftarrow i+1$.
  - Termination: $A[1...n]$ is sorted, when $i = n$.
- Correctness: Elements are exchanged only and $A[1...n]$ is sorted. So inequality (1) holds.

Problem 2

(a) $T(n) = c_1 + c_2(n+2) + c_3(n+1) = (c_2 + c_3)n + (c_1 + 2c_2 + c_3) = \Theta(n)$

(b)

- Loop invariant: After thee i-th loop, $y = \sum_{k=i}^{n} c_k x^{k-i}$.
- Proof:
    - Initialization: $y = 0$, before the loop.
    - Maintain: The old one $y = \sum_{k=i+1}^{n} c_k x^{k-i-1}$, so the new one $y' = c_i + xy = c_i + \sum_{k=i+1}^{n} c_k x^{k-i} = \sum_{k=i}^{n} c_k x^{k-i}$, when $i \leftarrow i - 1$.
    - Termination: $y = \sum_{k=0}^{n} c_k x^k$, when $i = 0$.
- Correctness: The algorithm will be terminated within $n + 1$ times of loop and the result $y = \sum_{k=0}^{n} c_k x^k$ is equal to $P(x)$.

Problem 3

(a) $f \in \Theta(g)$

(b) $f \in O(g)$

(c) $f \in \Theta(g)$

(d) $f \in \Theta(g)$

(e) $f \in \Theta(g)$

(f) $f \in \Theta(g)$

(g) $f \in \Omega(g)$

(h) $f \in \Omega(g)$

(i) $f \in \Omega(g)$

(j) $f \in \Omega(g)$

(k) $f \in \Omega(g)$

(l) $f \in O(g)$

(m) $f \in O(g)$

(n) $f \in \Theta(g)$

(o) $f \in \Omega(g)$

(p) $f \in \Omega(g)$

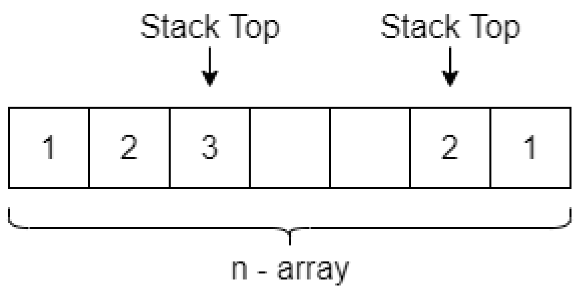
Problem 4

$$1 = n^{\frac{1}{\lg n}} \ll \lg(\lg^* n) = \lg^*(\lg n) \ll \lg^* n \ll 2^{\lg^* n} \ll \ln\ln n \ll \sqrt{\lg n} \ll \ln n \ll \lg^2 n \ll n = 2^{\lg n} \ll n\lg n = \lg(n!) \ll n^2 = 4^{\lg n} \ll n^3 \ll 2^{\sqrt{2\lg n}} \ll (\sqrt{2})^{\lg n} \ll \left(\frac{3}{2}\right)^n \ll 2^n \ll e^n \ll n \cdot 2^n \ll (\lg n)! \ll (\lg n)^{\lg n} = n^{\lg\lg n} \ll n! \ll (n+1)! \ll 2^{2^n} \ll 2^{2^{n+1}}$$

Problem 5

Overview:

We save two index variances that indicate the positions of two stack tops. The 1-based one increases with push operation, and the n-based one decreases with push operation. Let S, T be two stacks and i, j be two indices.



Algorithm:

Algorithm 1 Two Stacks with One Array

function INITIATE()
  $i \leftarrow 0$
  $i \leftarrow n + 1$
end function
function S.PUSH(x)
  $i \leftarrow i + 1$
  $A[i] \leftarrow x$
end function

function S.pop()
    $i \leftarrow i - 1$
    return $A[i+1]$
end function
function T.push(x)
    $j \leftarrow j - 1$
    $A[j] \leftarrow x$
end function
function T.pop()
    $j \leftarrow j + 1$
    return $A[j-1]$
end function

Problem 6

Overview:

Let Q, U be two FIFO queues. When the operation is *push*, we enqueue x to U. When the operation is *pop*, dequeue element from queue U and enqueue it to queue Q, until the size of queue U is 1. Then we dequeue the last element from queue U and save it as r, then we restore all elements from Q to U. Finally, return r.

Algorithm:

Algorithm 2 Stack Using Two FIFO Queues

function Push(x)
    U.enqueue(x)
end function
function Pop()
    while U.size() > 1 **do**
        Q.enqueue(U.dequeue())
    end while
    $r \leftarrow U$.dequeue()
    while Q is not empty **do**
        U.enqueue(Q.dequeue())
    end while
    return r
end function

Time Complexity:

Let n be the size of Q.

- Push: $T(n) = c_0 = \Theta(1)$

- Pop: $T(n) = (c_0 + c_1)(n-1) + c_2 + (c_0 + c_1)(n-1) + c_3 = (2c_0 + 2c_1)n + (c_2 + c_3 - 2c_0 - 2c_1) = \Theta(n)$

Bonus Problem

Overview:

Let $A[1...N]$ be an array, n be size or index of queue. When the operation *add*, save it in the tail of array and let n increases. When the operation *remove*, get an element randomly, swap it with the tail element, let n decreases and return the element.

Algorithm:

Algorithm 3 Stack Using Two FIFO Queues

```
function Add(x)
    n ← n + 1
    A[n] ← x
end function
function Remove()
    i ← random(n)
    r ← A[i]
    A[i] ← A[n]
    n ← n − 1
    return r
end function
```

Time Complexity:

- Push: $T(n) = c_0 + c_1 = O(1)$
- Pop: $T(n) = c_2 + c_3 + c_4 + c_5 = O(1)$

Correctness:

- Add: The used subarray is $A[1...n]$, and the new index $n' = n + 1$, we save x in $A[n']$, and then the used subarray will be $A[1...n+1]$.
- Remove: We get an element $A[i]$ randomly and save it in variance r. Then we move the tail element $A[n]$ to the position $A[i]$, which ensures that all remaining elements are still in subarray $A[1...n-1]$. Index n decreases so that the used subarray will be $A[1...n-1]$. Finally we return variance r, the element originally in position $A[i]$, which was chosen uniformly at random among all the elements. So we have proved the *remove* operation holds.