Solution for Problem Set 1

Mianzhi Pan, 181240045

September 17, 2020

1 Problem 1

(a)

Algorithm 1 SELECTION-SORT(A)

```
1: for i = 1 → A.length − 1 do
2:     mini = i
3:     for j = i + 1 → A.length do
4:         if A[j] < A[mini] then
5:             mini = j
6:         end if
7:     end for
8:     swap(A[i], A[mini])
9: end for
```

(b) Best case: the array A has been sorted, so array elements $A[i](1 <= i <= n-1)$ needs comparing $n-i$ times while no swaping. Then the running time is $\sum_{i=1}^{n-1}(n-i) = \Theta(n^2)$.

Worst case: A is reverse ordered. In addition to the same comparion times as the best case, this algorithm needs swaping $n-1$ times, so the running time is $\sum_{i=1}^{n-1}(n-i) + (n-1)$, which is still $\Theta(n^2)$.

(c) Loop invariant: Every time the program reaches line 1, the subarray $A[1,...,i-1]$ consisting of $i-1$ smallest elements from the origin array A is sorted.

Initialization: When $i = 1$, the subarray $A[1,...,i-1]$ is empty, and so we can say that it contains 0 smallest element from A and it is sorted, which indicates that the loop invariant holds before the first loop iteration.

Maintenance: Line 2 to 8 choose the smallest element from subarray $A[i,...,n-1]$, then swap it and $A[i]$. Therefore, $A[i]$ would be the i^{th} smallest element from the origin array A after the i^{th} loop.As a result, $A[1,...,i-1]$ consists of $i-1$ smallest elements from A in sorted order.

Termination: The condition causing the for loop to terminate is that $i > A.length-1$, at which time we have $i = n$. In the last loop, we swap the smaller element in $A[n-1,n]$ and $A[n-1]$, thus the subarray $A[1,...,n-1]$ consists of

$n-1$ smallest elements from A in sorted order and $A[n]$ is the biggest element. We conclude the whole array A is sorted. Hence the algorithm is correct.

2 Problem 2

(a) In each iteration, `PolyEval` needs to do addition and mutiplication both one time. So the runtime is $2n = \Theta(n)$.

(b) Loop invariant: Every time the program reaches line 2, we have $y = \sum_{j=0}^{n-1-i} c_{i+1+j}x^j$

Initialization: Before the first loop, we have $i = n$. Calculating $y = \sum_{j=0}^{-1} c_{n+1+j}x^j = 0$, we have the same value of y as line 1.

Maintenance: Since $c_0, \cdots, c_n$ and x are fixed, the loop invariant is only related with i, and we can write it as $f(i)$. Before each loop, we have $y = f(i)$. We want to prove that after each loop $y = f(i-1)$.

$$
\begin{aligned}
y &= c_i + x * f(i) \\
&= c_i + x * \sum_{j=0}^{n-1-i} c_{i+1+j}x^j \\
&= c_j + \sum_{j=0}^{n-i-i} c_{i+1+j}x^{j+1} \\
&= c_j * x^0 + \sum_{j=1}^{n-i} c_{i+j}x^j \\
&= \sum_{j=0}^{n-i} c_{i+j}x^j \\
&= f(i-1)
\end{aligned}
$$

Termination: In the last loop, $i = 0$. As a result, when the program terminates, $y = f(0-1) = \sum_{j=0}^{n} c_j x^j$, which is equal to $P(x)$. Hence the algorithm is correct.

3 Problem 3

(a) For $f(n)$ and $g(n)$, we know that $f(n) > 0, g(n) > 0$. And we can easily find $n_0 \geq 0$,s.t. for all $n \geq n_0$, we have

$$
\begin{aligned}
f(n), g(n) \leq max\{f(n),g(n)\} \Rightarrow \frac{1}{2}(f(n) + g(n)) \leq max\{f(n), g(n)\} \\
max\{f(n), g(n)\} \leq f(n) + g(n)
\end{aligned}
$$

so there are $c_1 = 1/2$ and $c_2 = 1$, s.t.$0 \leq c_1(f(n) + g(n) \leq max\{f(n), g(n)\} \leq c_2(f(n) + g(n)$, i.e.$\Theta(f(n) + g(n) = max\{f(n), g(n)\}$.

(b) Use binomial theorem, we have $(n+a)^b = C_b^0 n^0 a^b + C_b^1 n^1 a^{b-1} + \cdots + C_b^b n^b a^0$. We can easily find $n_0 \geq 0$, s.t. for all $n \geq n_0$, we have $n^b \leq C_b^0 n^0 a^b + C_b^1 n^1 a^{b-1} + \cdots + C_b^b n^b a^0 \leq (C_b^0 a^b + C_b^1 a^{b-1} + \cdots + C_b^b a^0) n^b$, i.e.$0 \leq \frac{(n+a)^b}{C_b^0 a^b + C_b^1 a^{b-1} + \cdots + C_b^b a^0} \leq n^b \leq (n+a)^b$.Therefore, $(n+a)^b = \Theta(n^b)$.

(c) Θ

4 Problem 4

$$
\begin{aligned}
&1 < n^{1/lgn} < lg(lg^* n) < lg^*(lgn) < lg^* n < 2^{lg^* n} \\
&< lnlnn < \sqrt{lgn} < lnn < lg^2 n < 2^{\sqrt{2lgn}} < (\sqrt{2})^{lgn} \\
&< 2^{lgn} = n < nlgn = lg(n!) < 4^{lgn} = n^2 \\
&< n^3 < (lgn)^{lgn} = n^{lglgn} < (3/2)^n < 2^n < n \cdot 2^n \\
&< e^n < (lgn)! < n! < (n+1)! < 2^{2^n} < 2^{2^{n+1}}
\end{aligned}
$$

5 Problem 5

Suppose we have two stacks A and B. When do `enqueue` operation, we `push` an element n into A. When we do `dequeue` operation, we `pop` n from B. The elements in B is from A. If A is full or B is empty, we `pop` elements in A and *mathttpush* them to B.

The following pseudocode doesn't consider overflow and underflow.

Algorithm 2 enqueue(x)

if $A.top == n$ **then**
  repeat
    $a = POP(A)$
    $PUSH(B, a)$
  until $B.top == n$ or $STACK - EMPTY(A)$
end if
$PUSH(A, x)$

Algorithm 3 dequeue

if $STACK - EMPTY(B)$ **then**
  repeat
    $a = POP(A)$
    $PUSH(B, a)$
  until $B.top == n$ or $STACK - EMPTY(A)$
end if
$POP(B)$

best-case:the running time for both `dequeue` and `enqueue` is $\Theta(1)$.

worst-case: If A is full and B is empty, the running time is both $\Theta(n)$.

6 Problem 6

Use two stacks A and B, the former to store real elements and the latter to store minimum element.When we do *push*, first $PUSH$ it to A, and then compare it with the top element b of B. If it is smaller than b,$PUSH$ it to B as well. Otherwise, $PUSH$ b to B again. So the top element of B is always the minimum of A.When we do *pop*, POP both two stacks and return the result from A.When we do *min*,return the top element of B.

Algorithm 4 push(x)

$PUSH(x, A)$
if not $STACK - EMPTY(B)$ **then**
    $b = POP(B)$
    $PUSH(b, B)$
    if $x < b$ **then**
        $PUSH(x, B)$
    else
        $PUSH(b, B)$
    end if
else
    $PUSH(x, B)$
end if

Algorithm 5 pop()

$a = POP(A)$
$POP(B)$
return a

Algorithm 6 min()

$b = POP(B)$
$PUSH(b, B)$
return b

We design the MINISTACK data structure with two stacks, each of their space complexity is $O(n)$. So the total space complexity is $O(n)$.

7 Problem 7

Use a queue A.When doing *remove* operation, we first choose a random number x, which means we will remove the x^{th} element in A. Then we do $DEQUEUE$

for $x-1$ times and $ENQUEUE$ them all. After that, we do $DEQUEUE$ once more and return the value.

Algorithm 7 add(x)

$ENQUEUE(A, x)$

Algorithm 8 remove

$x = random(N)$
while $x > 1$ **do**
    $a = DEQUEUE(A)$
    $ENQUEUE(A, a)$
    $x = x - 1$
end while
$a = DEQUEUE(A)$
return a

It is obvious that the time complexity of *add* is $O(1)$.When we do *remove* each time, we need to do $DEQUEUE$ for x times and $ENQUEUE$ for $x-1$ times while $1 \leq x \leq N$ is a constant. Then the time complexity of *remove* is $O(1)$ as well.

8 Problem 8

We scan the input expression from left to right. If we come across an operand, output it directly. Otherwise, push the operator we meet to a stack A. After scanning the whole expression, pop the elements in A and output them.

Algorithm 9 InToPost(E)

for $i = 1 \rightarrow E.length$ **do**
    if $E[i]$ is an operand **then**
        output $E[i]$
    else
        $PUSH(A, E[i])$
    end if
end for
repeat
    $a = POP(A)$
    output a
until $STACK - EMPTY(A)$

Considering the worst case(all elements are operators), the algorithm has to scan the whole expression and do n times' $PUSH$ and n times' POP. Total operations number is $3n$, so the time complexity is $O(n)$.