ALG Problem Set 1

Problem 1

We show that the following loop invariant holds: At the beginning of each iteration, $m = \max\{A[1..i-1]\}$.

Initialization: At the beginning of the first iteration, $i = 2$, and we have $m = A[1] = \max\{A[1..1]\}$.

Maintenance: Assume that at the beginning of iteration $i = k$, $m = \max\{A[1..k-1]\}$. In this iteration, if $A[k] > m$, then we set $m = A[k]$, which is the maximum of $A[1..k]$. Otherwise, we keep m unchanged, which is still the maximum of $A[1..k]$. At the beginning of the next iteration, $i = k+1$, we have $m = \max\{A[1..k]\}$.

Termination: When the loop terminates, we have $i = n+1$ and $m = \max\{A[1..n]\}$.

Problem 2

Algorithm 1 INSERTION SORT

1: **for** $j = 2$ **to** n **do**
2:   $key = A[j]$
3:   $i = j - 1$
4:   **while** $i > 0$ **and** $A[i] > key$ **do**
5:     $A[i+1] = A[i]$
6:     $i = i - 1$
7:   **end while**
8:   $A[i+1] = key$
9: **end for**

For the outer loop of lines 1-9, we show that at the beginning of each iteration, the subarray $A[1..j-1]$ consists of the elements originally in $A[1..j-1]$, but in sorted order.

Initialization: It is true prior to the first iteration, where $j = 2$.

Maintenance: Assume at the beginning of iteration j, the subarray $A[1..j-1]$ consists of the elements originally in $A[1..j-1]$, but in sorted order. Then for the inner loop of lines 4-7, we show that:

At the end of the inner loop, $i \geq 0$ and $A[i+2] > key$, the subarray $A[1..i, i+2..j]$ consists of the elements originally in $A[1..j-1]$, but in sorted order.

Initialization: If $j - 1 > 0$ and $A[j-1] > key$, the program enters the first iteration. At the end of the first iteration, we have $i = j - 2 \geq 0$. $A[j] \leftarrow A[j-1]$. Hence $A[i+2] = A[j] > key$. $A[1..j-2], A[j]$ consists of the elements originally in $A[1..j-1]$ in sorted order.

Maintenance: Assume at the beginning of the inner loop $i = k$, $k > 0$, $A[k] > key$ and $A[1..k, k+2..j]$ is the sorted sequence of original $A[1..j-1]$. We copy $A[k]$ to $A[k+1]$. So at the end of the inner loop, we have $i = k-1 \geq 0$ and $A[i+2] = A[k+1] = A[k] > key$, the subarray $A[1..k-1, k+1..j]$ consists of the elements originally in $A[1..j-1]$ in sorted order.

Termination: When the inner loop terminates,

1. If $i \leq 0$, then $i = 0$ and $A[2] > key$, $A[2..j]$ consists of the elements originally in $A[1..j-1]$ in sorted order.

2. If $i \geq 1$, then $A[i] \leq key < A[i+2]$. $A[1..i], A[i+2..j]$ consists of the elements originally in $A[1..j-1]$ in sorted order.

After $A[i+1] \leftarrow key$ at L8, $A[1..j]$ consists of the elements originally in $A[1..j]$ in sorted order.

Termination: When the outer loop terminates, $j = n+1$ and $A[1..n]$ consists of the elements originally in $A[1..n]$ in sorted order.

Problem 3

(a)

$$n^3 \geq r = \sum_{i=1}^{n/2} \sum_{j=i+1}^{n+1-i} (n-i-j+2) \geq \frac{1}{2} \sum_{i=1}^{n/2} (n-2i+1)^2 \geq \frac{1}{4} \sum_{i=1}^{n-1} i^3 = \frac{(n-1)n(2n-1)}{24}$$

$\boxed{\Theta(n^3)}$

(b) The number of '*'s $= \sum_{k=1}^{n} \lceil \lg \frac{n}{k} \rceil \geq \lg(\frac{n^n}{n!})$ and $\leq n + \lg(\frac{n^n}{n!})$.

By Stirling's approximation,

$$n! = \Theta(\frac{n^n \sqrt{n}}{e^n})$$

Hence

$$\frac{n^n}{n!} = \Theta(n^{-1/2} e^n)$$

$$\lg \frac{n^n}{n!} = \Theta(n - \lg n) = \Theta(n)$$

$\boxed{\Theta(n)}$

Problem 4

(a) [1] is correct.

Proof: By the definition of big O notation, we have:

- $\exists c_1 > 0, n_1 \geq 0$ such that $\forall n \geq n_1, f_1(n) \leq c_1 g(n)$.
- $\exists c_2 > 0, n_2 \geq 0$ such that $\forall n \geq n_2, f_2(n) \leq c_2 g(n)$.

Let $c_0 = \max\{c_1, c_2\} > 0, n_0 = \max\{n_1, n_2\}$. Then for all $n \geq n_0$, we have $f_1(n) \leq c_0 g(n)$ and $f_2(n) \leq c_0 g(n)$, which implies $f_1(n) + f_2(n) \leq c_0(g_1(n) + g_2(n))$.

[2] is correct.

Proof: By definition,

- $\exists c_1 > 0, n_1 \geq 0$ such that $\forall n \geq n_1, f_1(n) \geq c_1 g(n)$.
- $\exists c_2 > 0, n_2 \geq 0$ such that $\forall n \geq n_2, f_2(n) \geq c_2 g(n)$.

Let $c_0 = \min\{c_1, c_2\} > 0, n_0 = \max\{n_1, n_2\}$. Then for all $n \geq n_0$, we have $f_1(n) \geq c_0 g(n)$ and $f_2(n) \geq c_0 g(n)$, which implies $f_1(n) + f_2(n) \geq c_0(g_1(n) + g_2(n))$.

(b) $\boxed{\text{Only [2] is correct}}$

[1] is incorrect. Let $f(n) \equiv 1, g(n) = n, \min\{f(n), g(n)\} = 1 = o(n+1)$.

[2]: $\frac{f(n)+g(n)}{2} \leq \max\{f(n), g(n)\} \leq f(n) + g(n)$.

Problem 5

$1 = n^{1/\lg n} \ll \lg(\lg^* n) \ll \lg^* n = \lg^*(\lg n) \ll 2^{\lg^* n} \ll \sqrt{\lg\lg n} \ll \ln\ln n \ll \ln n \ll \lg^2 n \ll 2^{\sqrt{2\lg n}} \ll n = 2^{\lg n} \ll n\lg n = \lg(n!) \ll (\sqrt{5})^{\lg n} \ll n^2 = 4^{\lg n} \ll n^3 \ll (\lg n)! \ll (\lg n)^{\lg n} = n^{\lg\lg n} \ll (4/3)^n \ll 2^n \ll n \cdot 2^n \ll e^n \ll n! \ll (n+1)! \ll 2^{2^n} \ll 2^{2^{n+1}}$

Problem 6

In this problem, we will implement a stack using 2 queues Q_1, Q_2 and $O(1)$ additional variables (including a global variable t). We maintain the following invariant: After each operation, all elements in the stack are stored in Q_t in the order from bottom to top. Q_{3-t} is empty and used as a temporary storage.

Algorithm 2 PUSH(S, x)

```
GLOBAL t
ENQUEUE(Q_t, x)
```

Algorithm 3 POP(S)

```
GLOBAL t
x = NULL
while Q_t is not empty do
    x = DEQUEUE(Q_t)
    if Q_t is not empty then
        ENQUEUE(Q_{3-t}, x)
    end if
end while
t = 3 - t
return x
```

The running time of PUSH is $O(1)$.

Assume there are n elements in the stack. The running time of POP is $O(n)$, as it needs to dequeue all elements from one queue and enqueue all but one of them into the other.

By carefully manipulating the size of Q_1 and Q_2, we can obtain an amortized $O(1)$ PUSH and $O(\sqrt{n})$ POP. The efficient implementation satisfies the following invariant: After each operation, Q_1 contains the bottom part of the stack elements and Q_2 contains the top part, both in the order from bottom to top. And $|Q_2|^2 \leq |Q_1|$.

Algorithm 4 PUSH'(S, x)

```
ENQUEUE(Q_2, x)
if |Q_2|^2 > |Q_1| then
    y = DEQUEUE(Q_2)
    ENQUEUE(Q_1, y)
end if
```

Algorithm 5 POP'(S)

```
1: if Q_2 is not empty then
2:    for i = 1 to |Q_2| - 1 do
3:       x = DEQUEUE(Q_2)
4:       ENQUEUE(Q_2, x)
5:    end for
6:    return DEQUEUE(Q_2)
7: end if
8: if Q_1 is empty then
9:    return NULL
10: end if
11: len = |Q_1|
12: for i = 1 to len - 1 do
13:    x = DEQUEUE(Q_1)
14:    if (|Q_2| + 1)^2 <= |Q_1| then
15:       ENQUEUE(Q_2, x)
16:    else
17:       ENQUEUE(Q_1, x)
18:    end if
19: end for
20: return DEQUEUE(Q_1)
```

In amortized analysis, the running time of PUSH’ is $O(1)$ and that of POP’ is $O(\sqrt{n})$.

First we prove that $|Q_2|^2 \leq |Q_1|$.

- Initially, $|Q_1| = 0, |Q_2| = 0$.
- When we push an element: After enqueueing into Q_2, if $|Q_2|^2 > |Q_1|$, we move one element from Q_2 to Q_1. $|Q_1|$ increased by 1 and $|Q_2|$ remains unchanged. $|Q_2|^2 \leq |Q_1|$ holds.
- When we pop an element, if $|Q_2| = 0$, by checking before ENQUEUE at line 14, we ensure that $|Q_2|^2 \leq |Q_1|$ always holds.

Thus, we have $|Q_2|^2 \leq |Q_1|$ at all times.

Then let the potential function be $\Phi = |Q_1| - |Q_2|^2 \geq 0$.

The amortized cost of PUSH’: $O(1) + O(|Q_1| + 1 - |Q_2|^2 - (|Q_1| - |Q_2|^2)) = O(1) + O(1) = O(1)$.

Let $n = |Q_1| + |Q_2|$ be the total number of elements. The amortized cost of POP’:

- When $|Q_2| > 0$, $|Q_2| - (|Q_2| - 1)^2 + |Q_2|^2 = O(|Q_2|) = O(\sqrt{n})$.
- When $|Q_2| = 0$, $|Q_1| + (|Q_1| - t) - t^2 - |Q_1| = |Q_1| - t - t^2$, where t is the size of new Q_2. $(t+1)^2 > |Q_1| - t - 1$. Hence $|Q_1| - t - t^2 < \frac{|Q_1|}{t+2} < \sqrt{|Q_1|} = O(\sqrt{n})$.