CS3230 Assignment 7

Zhuang Jianning
A0245061M

1. Let $t(i)$ be the time complexity of the ith operation

$$t(i) = \begin{cases} i & \text{if } i \text{ is a power of 2} \\ 1 & \text{otherwise} \end{cases}$$

$$T(n) = \sum_{i=1}^{n} t(i) = \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j + \sum_{i \le n \,\wedge\, i \text{ not power of 2}} 1$$

$$\le \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j + n$$

$$\le 2^{\lfloor \lg n \rfloor + 1} - 1 + n$$

$$\le 3n \qquad \boxed{T(n) = O(n)}$$

$$T(n) = \sum_{i=1}^{n} t(i) = \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j + \sum_{i \le n \,\wedge\, i \text{ not power of 2}} 1$$

$$\ge \sum_{i=1}^{n} 1$$

$$\ge n \qquad \boxed{T(n) = \Omega(n)}$$

$$\therefore T(n) = \Theta(n) \Rightarrow \text{amortized cost per operation} = \frac{\Theta(n)}{n} = \Theta(1)$$

$$n \le T(n) \le 3n \Rightarrow 1 \le \frac{T(n)}{n} \le 3 \qquad \Theta(1)$$

2. similar to the aggregate analysis of the binary increment problem

let $f(i)$ = number of times the ith bit is flipped (rightmost bit is 0th bit)

$f(0) = n$

$f(1) = \left\lfloor \frac{n}{2} \right\rfloor$

$f(2) = \left\lfloor \frac{n}{4} \right\rfloor$

$f(i) = \left\lfloor \frac{n}{2^i} \right\rfloor$

* Assume k-bit binary number where $k = \lfloor \lg n \rfloor$

let $t(i)$ be time complexity from flipping ith bit

$$t(i) = \left\lfloor \frac{n}{2^i} \right\rfloor \cdot 2^i$$

$$\begin{aligned} T(n) = \sum_{i=0}^{k-1} t(i) &= \sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor \cdot 2^i \\ &\le \sum_{i=0}^{k-1} \frac{n}{2^i} \cdot 2^i \\ &\le kn \\ &\le n\lg n \end{aligned}$$

$$\boxed{T(n) = O(n\lg n)}$$

$$\begin{aligned} T(n) = \sum_{i=0}^{k-1} t(i) &= \sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor 2^i \\ &\ge \sum_{i=0}^{k-1} \left(\frac{n}{2^i} - 1 \right) 2^i \\ &\ge \sum_{i=0}^{k-1} n - 2^i \\ &\ge kn - 2^k + 1 \\ &\ge n(\lg n - 1) - 2^{\lg n - 1} + 1 \\ &\ge n\lg n - \frac{3}{2}n + 1 \ge \frac{1}{2} n\lg n \quad \text{for } n \ge 8 \end{aligned}$$

$$\boxed{T(n) = \Omega(n\lg n)}$$

$\therefore T(n) = \Theta(n\lg n) \Rightarrow$ amortized cost per operation $= \dfrac{\Theta(n\lg n)}{n} = \Theta(\lg n)$ //

3. Let A be the array of bits with A.length storing length
and a new field A.max storing index of highest order 1-bit

initially set A.max = -1

```
INCREMENT(A)
i = 0
while i < A.length and A[i] = 1
    A[i] = 0
    i++
if i < A.length
    A[i] = 1
    if i > A.max
        A.max = i
```

```
RESET(A)
for i = 0 to A.max
    A[i] = 0
A.max = -1
```

For INCREMENT, set amortized cost to 6.

\$1 to set a bit from 0 → 1 } similar to lecture
\$1 as credit on bit set to 1 to be used when 1 → 0 }
\$1 as credit on bit set to 1 to be used when examining if A[i] = 1 (while loop)
\$ 1 as credit on bit set to 1 to be used when examining A[i] = 0
\$1 when while loop terminates after examining A[i] = 0
\$1 to update A. max (assume updating A.max cost unit time)
\$1 as credit on new highest order 1 to be used during RESET since
each bit up to A.max must have been highest order at some time

For RESET, set amortized cost to 1.

\$1 to update A.max to -1

zeroing of bits paid by credit stored when updating A.max as highest order bit

Amortized cost for each operation = $O(1)$

Amortized cost for n operations = $O(n)$