# COMP 5711: Advanced Algorithms
# Fall 2020 Midterm Exam Solutions

1. (a) Suppose the first $n_1$ operations are increments, where $m = 2^{\lfloor \log n \rfloor - 1} \le n/2$, so that afterwards, $A[i] = 1$ if and only if $i = \lfloor \log n \rfloor - 1$. The followed by $n - m$ operations are decrements, increments, decrements, increments... It can be verified that each of the followed operation has $\Omega(\log n)$ cost, so the total cost is at least $\Omega(\log n) \cdot n/2 = \Omega(n \log n)$, and the amortized cost is $\Omega(\log n)$.

   (b) Blanks are filled as follows.

**Increment** val($A$)

$i \leftarrow 0$
**while** $A[i] = \underline{\ 1\ }$ **do**
  $A[i] \leftarrow \underline{\ 0\ }$
  $i \leftarrow i + 1$
**end while**
$A[i] \leftarrow A[i] + 1$

**Decrement** val($A$)

$i \leftarrow 0$
**while** $A[i] = \underline{\ \text{-1}\ }$ **do**
  $A[i] \leftarrow \underline{\ 0\ }$
  $i \leftarrow i + 1$
**end while**
$A[i] \leftarrow \underline{\ A[i] - 1\ }$

   (c) We use accounting method to prove the amortized cost is at most 2. Let \$1 represent each unit of cost. Whenever a value changes from 0 to 1 or from 0 to -1, we will spend \$2: \$1 for the actual cost, and store another \$1 on the value as credit. When a bit is set to 0, the stored \$1 credit pays the cost. At most one value changes from 0 to 1 (changes from 0 to -1) in each Increment (Decrement) operation, so the amortized cost of \$2 is enough to cover all its changes.

   Alternatively, one can also use the potential method by defining the potential as $\Phi = \sum_i |A[i]|$.

2. (a) It is FPT repsect to $k$. This is because

$$mn + \text{poly}(k) \cdot (k!)^2 \cdot n \le (\text{poly}(k) \cdot (k!)^2 + 1) \cdot mn \le f(k) \cdot \text{IN}^2 = f(k) \cdot \text{poly}(\text{IN}),$$

   where $f(k) = \text{poly}(k) \cdot (k!)^2 + 1$.

   (b) Define $f(k) = \text{poly}(k) \cdot (k!)^2 + 1$ as before. It is easy to see that the algorithm runs in polynomial time if and only if $f(k) = \text{poly}(\text{IN})$. Note that $f(k) > k!$ and $f(k) < (k!)^3$, so this is equivalently to $k! = \text{poly}(\text{IN})$.

   **Sufficiency.** Assume $k = O(\log \text{IN} / \log\log \text{IN})$. Note that

$$k \log k = O\left(\frac{\log \text{IN}}{\log\log \text{IN}} (\log\log \text{IN} - \log\log\log \text{IN})\right) = O(\log \text{IN}),$$

   so $k! < k^k = 2^{k \log k} = O(2^{\log \text{IN}}) = O(\text{IN}) = \text{poly}(\text{IN})$.

   **Necessity.** Assume $k = \omega(\log \text{IN} / \log\log \text{IN}) = \omega(1) \cdot \log \text{IN} / \log\log \text{IN}$. Then for large enough IN, $\omega(1) > 2$ and $\log\log \text{IN} > 2 \log\log\log \text{IN}$. Therefore,

$$\log \frac{k}{2} > \log\log \text{IN} - \log\log\log \text{IN} > \frac{1}{2} \log\log \text{IN},$$

   and hence

$$k! \ge \prod_{i=k/2}^{k} i \ge (k/2)^{k/2} = 2^{k/2 \cdot \log(k/2)} \ge 2^{k/4 \cdot \log\log \text{IN}} = \text{IN}^{\omega(1)},$$

   which means $k! = \text{poly}(\text{IN})$ does not hold.

3. Define $k = (m+1)\ln n$. For $i = 1, 2, \cdots, k$, let $(w_{i1}, \cdots, w_{in})$ be a random permutation of $(1, 2, \cdots, n)$, and $G_i = (V, E_i)$ where $E_i = \{(u,v) \in E \mid w_{iu} < w_{iv}\}$. The algorithm will return $G' = (V, E')$, where $E' \in \{E_1, \cdots, E_k\}$ is the one with largest cardinality, i.e., $|E'| \geq E_i$ for all $i$.

   Recall that generating a random permutation has $O(n)$ cost. Since computing each $E_i$ takes $O(m)$ cost, the total running time of the algorithm is $O((m+n)\cdot(m+1)\ln n)$, which is polynomial.

   Now we prove its correctness. First we prove that each $G_i$ is acyclic. If this is not the case, then there exists a cycle, say, $\{(u_1,u_2),(u_2,u_3),\cdots,(u_k,u_1)\} \subseteq E_i$. By the construction of $E_i$, we have $w_{iu_1} < w_{iu_2} < \cdots < w_{iu_k} < w_{iu_1}$, which is a contradiction.

   Define $p_i = \Pr(|E_i| \geq m/2)$. Then we show $p_i \geq 1/(m+1)$, so that the algorithm fails with probability at most $\prod_{i=1}^k (1-p_i) \leq (1-1/(m+1))^k < (1/e)^{\ln n} = 1/n$, as required. First note that for each edge $(u,v) \in E$, $\Pr((u,v) \in E_i) = \Pr(w_{iu} < w_{iv}) = 1/2$. Therefore, $\mathbb{E}[|E_i|] = \sum_{(u,v)\in E} \Pr((u,v) \in E_i) = m/2$. On the other hand, since $m$ is an integer, an integer $j < m/2$ implies $j \leq (m-1)/2$. Therefore,

$$\begin{aligned}\frac{m}{2} = \mathbb{E}[|E_i|] &= \sum_{j=1}^{m} j \cdot \Pr(|E_i| = j) = \sum_{j<m/2} j \cdot \Pr(|E_i| = j) + \sum_{j \geq m/2} j \cdot \Pr(|E_i| = j) \\ &\leq \frac{m-1}{2} \sum_{j<m/2} \Pr(|E_i| = j) + m \sum_{j\geq m/2} \Pr(|E_i| = j) = \frac{m-1}{2}(1-p_i) + mp_i,\end{aligned}$$

   which implies $p_i \geq 1/(m+1)$.

4. Recall that in the random permutation problem, we are given an array $A$ with size $n$, indexed by $1, 2, \cdots, n$, and we want to permute the array such that each permutation appears with the same probability.

   (a) In the $i$-th step of the for-loop, there are exactly $i-1$ values that have been taken from $\{1, 2, \cdots, n\}$, so the success probability of each trial in line 5 to 7 is $p_i = 1-(i-1)/n$. Recall the waiting time for first success in class, the expected running time of the $i$-th step is $n/(n-i+1)$. Therefore, by the linearity of expectation, the total expected running time is

$$O\left(\sum_{i=1}^{n} \frac{n}{n-i+1}\right) = O\left(n \cdot \sum_{i=1}^{n} \frac{1}{i}\right) = O(n \log n).$$

   (b) The algorithm fails if and only if there are duplicates in $n$ independent uniform random values, taken from a domain with $n^2$ values. Let $E_i$ be the event that $W_j \neq W_i$ for all $j < i$. Then the success probability is

$$\begin{aligned}p(n) &= \Pr(E_2) \cdot \Pr(E_3 \mid E_2) \cdot \cdots \cdot \Pr(E_n \mid \bigcap_{i=2}^{n-1} E_i) \\ &= \prod_{i=0}^{n-1} \left(1 - \frac{i}{n^2}\right) \geq \left(1 - \frac{n}{n^2}\right)^n = \left(1 - \frac{1}{n}\right)^n \geq \frac{1}{4}.\end{aligned}$$

   On the other hand, when $n$ is even,

$$p(n) = \prod_{i=0}^{n-1} \left(1 - \frac{i}{n^2}\right) \leq \prod_{i=n/2}^{n-1} \left(1 - \frac{i}{n^2}\right) \leq \left(1 - \frac{n/2}{n^2}\right)^{n/2} = \left(1 - \frac{1}{2n}\right)^{2n/4} < e^{-\frac{1}{4}}.$$

When $n$ is odd, the proof is similar. Finally we choose $c_1 = 1/4$ and $c_2 = e^{-1/4}$ to complete the proof.

*Alternative Solution.* Define $X_{ij} = 1$ if $W_i = W_j$, and 0 otherwise, and let $X = \sum_{1 \le i < j \le n} X_{ij}$ be number of pairs of that have the same values. Recall the analysis of the birthday paradox in class, we have $\mathbb{E}[X] = n(n-1)/(2n^2) = (n-1)/(2n)$. Since $X$ takes non-negative values, by Markov inequality,

$$p(n) = 1 - \Pr(X \ge 1) = 1 - \Pr(X \ge \frac{2n}{n-1} \cdot \mathbb{E}[X]) \ge 1 - \frac{n-1}{2n} > \frac{1}{2}.$$

On the other hand, since $\ln(1+x) \le x$ and $n \ge 2$,

$$p(n) = \prod_{i=0}^{n-1} \left(1 - \frac{i}{n^2}\right) = \exp\left(\sum_{i=0}^{n-1} \ln\left(1 - \frac{i}{n^2}\right)\right) \le \exp\left(-\sum_{i=0}^{n-1} \frac{i}{n^2}\right) < e^{-\frac{1}{2}} \cdot e^{\frac{1}{2n}} \le e^{-\frac{1}{4}}.$$

Finally we choose $c_1 = 1/2$ and $c_2 = e^{-1/4}$ to complete the proof.

(c) To turn it into a Las Vegas algorithm, we just repeat until success. Since the success probability $p(n) \ge 1/4$, the expected running time is $O(n \log n / p(n)) = O(n \log n)$.