COMP 5711: Advanced Algorithms
Fall 2020 Midterm Exam

1. (30 pts) Recall the binary counter algorithm, which works as follows. It keeps an array A of bits, all initialized to 0, to represent an integer $\text{val}(A) = \sum_{i \geq 0} A[i] \cdot 2^i$. To increment $\text{val}(A)$, it sets $A[i] \leftarrow 0$ for $i = 0, 1, \ldots, k$, where $k = \max\{i \mid A[i] = 1\}$. Then set $A[k+1] \leftarrow 1$. We proved that this algorithm has an amortized cost of $O(1)$. We now extend the counter to allow both increments and decrements. Note that we don't allow decrements when $\text{val}(A) = 0$.

   (a) (10 pts) We do increments as above. For a decrement operation, we do the opposite, i.e., set $A[i] \leftarrow 1$ for $i = 0, 1, \ldots, k$, where $k = \max\{i \mid A[i] = 0\}$, then set $A[k+1] \leftarrow 0$. Show that the amortized cost of this algorithm is $\Omega(\log n)$, where n is the total number of increment/decrement operations.

   (b) (5 pts) We now design a new algorithm to reduce the amortized cost to $O(1)$. The entries of A can take 3 possible values 0, 1, or -1 (instead of just 0 and 1). The value stored in the counter is still represented by $\text{val}(A) = \sum_{i=0}^{k-1} A[i] \cdot 2^i$. The increment and decrement algorithms are given below. Please fill in the 5 blanks, so that it works correctly.

Increment

```
i ← 0
while A[i] = ______ do
    A[i] ← ______
    i ← i + 1
end while
A[i] ← A[i] + 1
```

Decrement

```
i ← 0
while A[i] = ______ do
    A[i] ← ______
    i ← i + 1
end while
A[i] ← ______
```

   (c) (15 pts) Show that the algorithm in (b) has amortized cost $O(1)$.

2. (20 pts) Recall that the running time of dynamic programming algorithm for circular arc coloring problem is $O(mn + \text{poly}(k) \cdot (k!)^2 \cdot n)$, where n is the number of vertices and m is the number of edges.

   (a) (6 pts) Show that this algorithm is FPT with respect to k. You must use the definition, i.e., an algorithm is FPT if its running time is $O(f(k) \cdot \text{poly}(\text{IN}))$ for some function $f(k)$, where IN is the input size. For this problem, we take $\text{IN} = m + n$.

   (b) (14 pts) Prove that the algorithm runs in polynomial time if and only if $k = O(\log \text{IN} / \log \log \text{IN})$.

3. (20 pts) Design a randomized polynomial-time algorithm for the following problem. You need to **describe** your algorithm, show that it runs in **polynomial time**, and prove its **correctness** (i.e., the success probability). Given a *directed* graph $G = (V, E)$ where V is the set of vertices and E is the set of edges. Let $n = |V|, m = |E|$. We want to find a subgraph $G' = (V, E')$, where $E' \subseteq E$, such that G' is acyclic. Design an algorithm that, with probability at least $1 - 1/n$, finds such a subgraph G' with $|E'| \geq m/2$. [*Hint: Assign a random permutation to the vertices.*]

4. (30 pts) 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. Here we consider two other algorithms for this problem.

(a) (10 pts) Give an asymptotic expected running time of Algorithm 1 and justify your answer (assuming line 6 has $O(1)$ cost). Make your bound as tight as possible.

Algorithm 1 Permute A by sampling

1: $n \leftarrow$ the size of A
2: $B \leftarrow$ an array with size n
3: $Z \leftarrow$ an array with size n and all elements initialized to be 0
4: **for** $i \leftarrow 1$ **to** n **do**
5:     **repeat**
6:         $j \leftarrow$ an uniformly random value taken from $\{1, 2, \cdots, n\}$
7:     **until** $Z[j] = 0$
8:     $B[i] \leftarrow A[j]$
9:     $Z[j] \leftarrow 1$
10: **end for**
11: **return** B

(b) (14 pts) Suppose Algorithm 2 succeeds with probability $p(n)$. Prove that there exist constants $0 < c_1 < c_2 < 1$ such that $c_1 \leq p(n) \leq c_2$ for any $n > 1$.
[*Hint: For* $p(n) \geq c_1$, *use the inequality* $\prod_{i=1}^{k}(1 - i/n) \geq (1 - k/n)^k$. *For* $p(n) \leq c_2$, *use Markov inequality.*]

Algorithm 2 Permute A by sorting

1: $n \leftarrow$ the size of A
2: $W \leftarrow$ an array with size n
3: **for** $i \leftarrow 1$ **to** n **do**
4:     $W[i] \leftarrow$ an uniform random value taken from $\{1, 2, \cdots, n^2\}$
5: **end for**
6: Sort A, using W as sort keys
7: **if** there exist $i < j$ such that $W[i] = W[j]$ **then**
8:     *Halt with failure*
9: **end if**
10: **return** A

(c) (6 pts) Algorithm 2 is a Monte Carlo algorithm. How to turn it into a Las Vegas algorithm? Note that you cannot just use the permutation algorithm in the textbook/slides; you must use Algorithm 2 as a black box for the reduction. What is the asymptotic expected running time of the Las Vegas algorithm (assuming line 4 has $O(1)$ cost, and line 6 and 7 have $O(n \log n)$ cost)? Make your bound as tight as possible.