**COMP4804 W2005 Midterm Exam - 1h20m**

Answer all questions in the exam booklet. No notes, books or calculators are allowed.

## 1 List Ranking by Independent Set Removal

We have $n$ distinct points arranged on a circle and we perform the following experiment: We color each point black or white, independently, each with probability 1/2. We say that a point $v$ is good if $v$ is colored black and $v$'s counterclockwise neighbour is colored white.

1. Let $p$ be the probability that a point $v$ is good. What is the value of $p$? What is the expected number of good points?
2. Suppose that we repeat this experiment $k$ times and after each iteration we tag all the good points. What is the probability that a point $v$ remains untagged after $k$ iterations? What is the expected number of untagged points after $k$ iterations?
3. What is the expected number of untagged points after $\log_{1/(1-p)} n + c$ iterations? Give an upper bound on the probability that, after $\log_{1/(1-p)} n + c$ iterations, there are any untagged points. (Hint: Don't use Chernoff's bounds)

## 2 Monte-Carlo Landslide Finding

We are given an array $A_1, \ldots, A_n$ and we are told that some element $x$ occurs $2n/3$ times in the array, but we are not told the value of $x$. Our goal is to use a fast Monte-Carlo algorithm (that may report the incorrect value) to find $x$.

1. Describe a constant-time Monte-Carlo algorithm to find $x$ that is correct with probability 2/3.
2. Suppose we sample $k$ elements at random (with replacement) from the array to obtain $k$ sample values $S_1, \ldots, S_k$. Give a good upper-bound on the probability that $x$ occurs less than $(1-\epsilon)2k/3$ times in this sample.
3. Give a good upper bound on the probability that $x$ occurs less than $k/2$ time in the sample. (Hint: This is the same as the previous question except we are using a specific value of $\epsilon$.)
4. Describe a Monte-Carlo algorithm that runs in $O(k)$ time and reports $x$ with probability at least $1 - 1/e^{\Omega(k)}$. (Just describe the algorithm. The error probability follows from the previous questions.)

## 3 Dynamically Growing and Shrinking Arrays

Consider the following memory-efficient implementation of a stack as an array $A$. Initially, we allocate an array $A$ of size 1. Let $n$ denote the size (number of elements on the stack) at some point in time.

If the user pushes an element on to the stack but the array is already full (because it has size $n$, i.e, $|A| = n$) then we allocate a new array $A'$ of size $\lceil 3n/2 \rceil$, copy $A$ into $A'$, and set $A \leftarrow A'$.

Conversely, if the user pops an element from the stack, but the stack has size $2n$ ($|A| = 2n$) then we allocate a new array $A'$ of size $\lceil 3n/2 \rceil$, copy the first $n$ elements of $A$ into $A'$ and then set $A \leftarrow A'$.

1. Define a non-negative potential function, $\Phi(n, |A|)$, that will allow you to prove that both push and pop take $O(1)$ amortized time. (Hint: Having $|A| = 3n/2$ is good, but having $|A| = n$ or $|A| = 2n$ is bad.)
2. Using your potential function, show that push takes $O(1)$ amortized time.
3. Using your potential function, show that pop takes $O(1)$ amortized time.