# COMPSCI330 Design and Analysis of Algorithms
# Assignment 8

Due Date: Wednesday April 17, 2019

## Guidelines

- **Describing Algorithms** If you are asked to provide an algorithm, you should clearly define each step of the procedure, establish its correctness, and then analyze its overall running time. There is no need to write pseudo-code; an unambiguous description of your algorithm in plain text will suffice. If the running time of your algorithm is worse than the suggested running time, you might receive partial credits.
- **Typesetting and Submission** Please submit the problems to GradeScope. You will be asked to **label your solution for individual problems**. Failing to label your solution can cost you 5% of the total points (3 points out of 60 for this homework).
- LaTeX is preferred, but answers typed with other software and converted to pdf is also accepted. Please make sure you submit to the correct problem, and your file can be opened by standard pdf reader. **Handwritten answers or pdf files that cannot be opened will not be graded.**
- **Timing** Please start early. The problems are difficult and they can take hours to solve. The time you spend on finding the proof can be much longer than the time to write. If you need more time for your homework please use this form and submit a STINF.
- **Collaboration Policy** Please check this page for the collaboration policy. You are **not** allowed to discuss homework problems in groups of more than 3 students. **Failure to adhere to these guidelines will be promptly reported to the relevant authority without exception.**

**Problem 1** (SlowSort)**.** (20 points) Consider an algorithm that is very similar to QuickSort (which we will call SlowSort). This algorithm splits the array in the exact same way as the QuickSort algorithm. However, it does some additional operations so the total split and merge time is $n^2$ for an array of length $n$.

Let $X_n$ be the running time of SlowSort on an array of size $n$. We assume in the base case $X_0 = X_1 = 0$. The goal of this problem is to analyze the running time of SlowSort.
(a) (5 points) Use the law of total expectations, write a recursion for $\mathbb{E}[X_n]$.
(b) (15 points) Use induction to prove that $\mathbb{E}[X_n] \leq Cn^2$ for some constant $C$. Determine the smallest possible $C$ that works for your proof.
(Hint: for this problem you can use the approximation $\sum_{i=1}^{n}(i-1)^2 \approx n^3/3$.)

**Problem 2** (Birthday Paradox and Hashing)**.** (20 points) Birthday paradox says that for a classroom of 23 people, the probability that two of them have the same birthday is more than 50%. In this problem we are going to consider a generalized version of this problem.
(a) (10 points) Let $x_1, x_2, ..., x_k$ be $k$ numbers chosen independently from $1, 2, ..., n$, and $k$ is much smaller than $n$. Estimate the probability that all these $k$ numbers are different from each other. How large does $k$ need to be for this probability to be below 50%? (In this problem, you can use the approximation $(1-x) \approx e^{-x}$ when $x \leq k/n$.)
(b) (10 points) Suppose $a_1, a_2, ..., a_k$ are $k$ distinct numbers, $f$ is a random function from a pairwise independent hash family that maps $a_i$ to $\{1, 2, ..., n\}$. Let $x_i = f(a_i)$. For every pair $(i, j) : 1 \leq i < j \leq k$, if $x_i = x_j$ then we say there is a collision. What is the expected number of collisions? How large does $k$ need to be for the expected number of collisions to be more than 1?

**Problem 3** (Expensive Counters)**.** (20 points) In order to represent large integers, we can store them in a binary array. As an example, the number 11 can be represented by an array $A[] = [1, 1, 0, 1]$. The first element in the array represents the least significant bit, and in general the $k$-th element in the array represents $2^{k-1}$. The array $A[]$ is a representation for 11 because $11 = 1+2+8$.

You are implementing a binary counter on a strange hardware. Your counter consists of $n$ binary bits. The counter starts at 0, and has a function called "increase". If the counter is representing number $x$ now, after calling "increase" it should represent the number $x+1$ (counter resets to 0 if $x$ becomes $2^n$).

The hardware is very strange, so flipping the $k$-th bit in the counter actually takes $k$ units of time. Show that the amortized cost for the increase operation is $O(1)$.

You can use any method that we talked about in class, charging method is easier in terms of calculation, aggregate method is easier conceptually but you need to deal with a difficult sum.)

Hint 1 (for aggregate): If you are using aggregate method, you can focus on the case where you take the average over $m = 2^n$ add operations. You can also use the following formula: for any sequence $a_i$ we have

$$\sum_{i=1}^{n} a_i \times (n - i + 1) = \sum_{i=1}^{n}(\sum_{j=1}^{i} a_j).$$

Hint 2 (for charging): flipping the $k$-th bit requires $k$-units of time, if would be good if the previous $k-1$ bits can pay for 1 unit each.