**CS 344 - Sections 1,2,3 - Fall 2018**
**Homework 4.**
**Due Tuesday, November 13, 11:55pm**
**80 points total plus extra credit**

# 1 Problem 1: Random Binary Search (20 points total)

Consider the Following Problem:

- Input: a *sorted* array $A$ of length $n$, and a target number $k$
- Output: index $i$ such that $A[i] = k$, or no solution if none exists.

Previously, we showed that binary search solves this problem in $O(\log(n))$ time by repeatedly looking at the middle and recursing to one side. Let's say that instead of picking the middle element, we picked a *random* element in the array. Here is the pseudocode.

**Algorithm: RandomBinarySearch(A,k,left,right)** – in the initial call, left = 0 and right = n-1

1. If left > right return "no solution"
2. i ← Rand(left....right)
3. If $A[i] = k$ return $i$
4. If $A[i] > k$
   - return RandomBinarySearch(A,k,left, i-1)
5. If $A[i] < k$
   - RandomBinarySearch(A,k,i+1,right)

In this question, we will figure out the expected running time of this RandomBinarySearch. Note that all the work done by the algorithm consists of comparing some $A[i]$ to $k$ (and then recursing and comparing some other $A[i]$ to k). In particular, each $A[i]$ is compared to $k$ either 0 or 1 times. The expected running time is thus the expected number of comparisons made by the algorithm.

- For now, let assume that $k$ does exist somewhere in the array, and figure out the expected running time for that case.
- Define indicator random variables $X_1 ... X_n$, where $X_i = 1$ if $A[i]$ is compared to $k$, and 0 otherwise. For example, $X_5 = 1$ if 5 is chosen as the random index in line two for some recursive call, and $X_5 = 0$ if 5 is never chosen.
- **Part 1 (5 points):** What is E$[X_1]$? Do you see why it depends on the value of $k$? In particular, let $j$ be index such that $A[j] = k$, and express your answer in terms of $j$.
- **Part 2 (5 points):** Derive a general formula for $E[X_i]$ (again in terms of $j$)
- **Part 3 (5 points:)** Use part 2 to figure out the expected running of RandomBinarySearch.
- **Part 4 (5 point:)** What is $E[X_i]$ if $k$ is NOT in the array? Note that we can no longer define a $j$ such that $A[j] = k$, because $k$ not in the array. Think about how you want to define $j$ instead, and make sure to be explicit in your solution about how you are defining $j$.

Note that in the analysis above, the algorithm doesn't actually know $j$. But that's ok because our analysis showed that no matter what $j$ happens to be, the expected running time is O(insert your answer to part 3).

# 2 Problem 2 (10 points total)

Let's say you have a set $S$ of $n$ keys, all of which are in the universe $U = [1....2^{64}]$. (As always, assume $n$ much less than $|U|$.) You want to build a dictionary for the set $S$. You decide to pick your hash function $h : [1...2^{64}] \to [1...n]$ in the following way:

- Let $b \leftarrow$ Rand(1....n)
- For all $x \in U$ let $h(x) = \left\lceil \frac{x}{b} \right\rceil \bmod n$.

**The Problem:** For every question below, you should briefly explain your answer.

- **Part 1:** (5 points) Show that this is not a universal hash function by showing two keys that have a high probability of collision. (state what that probability is.)
- **Part 2 (3 points):** Show a set $S$ of $n$ keys such that with probability at least 1/3, $n/2$ keys in $S$ all collide in the same spot (i.e. there are n/2 keys in $S$ that all hash to the same number.)
- **Part 3 (2 points):** For the set $S$ in part 2, what would be expected time to do a search in this hash table in big-O notation?

# 3 Problem 3 (10 points total)

You have a set $S$ of $2^{20}$ (about a million) black and white images. Each image is 30x30 pixels, and each pixel is either white or black. You want to build a dictionary that will allow you to search whether a given image is already in $S$.

- **Part 1 (1 points):** How large is the universe size $U$?
- **Part 2 (9 points):** You decide to use the following hash function $h : U \to [1....2^{20}]$
  - Pick twenty random pixels $p_1, p_2, ..., p_{20}$. So for example, $p_1$ might end up being the pixel at position (10,20).
  - Given any image $I$, compute $h(I)$ as follows
    - we will construct a 20-bit number from $I$. Check whether $I$ is black or white at pixel $p_1$: if black, make the leading digit a 0, if white make it a 1. Check whether $I$ is black or white at $p_2$ to determine the second digit, and so forth. Let $h(I)$ be the resulting 20-bit number. Note that because $h(I)$ is 20 bits, we indeed have $h(I) \in [1....2^{20}]$

  **The question:** Argue that $h$ is not universal by exhibiting a pair of images $I_1$, $I_2$, for which $\Pr[h(I_1) = h(I_2)]$ is significantly larger than $1/2^{20}$. (State what the probability is.)

# 4 Problem 4 (20 points)

Consider the following problem:

- Input: an array $A$ with $n$ distinct numbers, and an array $B$ with the same $n$ numbers in a different order.
- Output: the goal is find the number whose position in $A$ is most similar to its position in $B$. Formally, say that a pair $(i, j)$ is related if $A[i] = B[j]$; we want to find the related pair that minimizes $|j - i|$. If there multiple pairs that minimize $|j - i|$, you only have to return one of them.

Write pseudocode for an algorithm that solves this problem in $O(n)$ expected time.

# 5 Problem 5 (20 points)

Consider the following problem, where we are given an array $A$ with some duplicate elements, and want to find the number of distinct elements in each interval of size $k$.

- Input: a positive integer $k$, and an unsorted $A$ with $n$ numbers, some of them repeating.
- Output: an array $B$ of length $n - k$, where $B[i]$ should contain the number of *distinct* elements in the interval $A[i], A[i+1], ..., A[i+k]$.

Write pseudocode for an algorithm for this problem with expected running time $O(n)$. Note that your expected running time should be $O(n)$ even if $k$ is large.

HINT: similar to the algorithm for sorting a k-sorted array, you will want to take advantage of the fact that two consecutive intervals $A[i]...A[i+k]$ and $A[i+1]...A[i+k+1]$ only differ by a small number of elements.

# 6 Problem 6: extra credit

I have a set $S$ of $n$ numbers, all in the universe $U = [1...2^{64}]$. I want to build a *static* dictionary for this set – no insertions or deletions. But I want to make sure that searching in this dictionary is *always* fast, not just fast in expectation. In particular, I want to design a dictionary with the following properties:

- Build-Dictionary(S) should take $O(n)$ time *in expectation*
- Dictionary.search(key k) should *always* take $O(1)$ time – no expectation here
- I am ok with using $O(n^2)$ space for my dictionary.

Write pseudocode for an algorithm that satisfies the 3 conditions above. You can assume that for any $t$ you can construct a universal hash function $h : U \rightarrow [1...t]$ in just $O(1)$ time. *Make sure to analyze your algorithm*, and in particular to argue why search is worst-case $O(1)$, and Build-Dictionary is expected $O(n)$.

HINT 1: it will not be enough to pick a single universal hash function $h$, because you might get unlucky and have a lot of collisions. You will thus want to use an approach similar to the select algorithm in class, and keep picking a universal function until you find a good one.

HINT 2: Say you have a random variable $X$ that takes some non-negative integer value, and you know that $E[X] \leq .1$. Given the definition of expectation, think about what kind of lower bound does this gives you on $\Pr[X = 0]$. The number .1 was made up, but you will likely need a similar argument in your analysis.