CS525DA
HW#2

DUE: Tuesday, September 30

1. (8 points) Assume you are given an array A of n integers, and the desired output is an integer x which does not belong to A. Assume the benchmark computational operation is a standard C operation.

(a)Prove that the complexity of this problem is in $O(n)$.

(b)Prove that the complexity of this problem is in $?(n)$.

2. (4 points) Assume you are given an array A of n integers, and the desired output is the median of A. You may assume that n is odd. For example, if n=5 and $A = \langle -25, 14, 22, -10, -558 \rangle$, then the answer is –10. Assuming the benchmark computational operation is a pairwise comparison, describe an algorithm to solve this problem which in the best-case uses n-1 pairwise comparisons.

3. (9 points) Assume you are throwing n labeled balls into m labeled bins, where each bin has unlimited capacity (each bin can hold at least n balls). The balls are thrown independently (knowing the the cell to which any ball goes gives no information on where any other ball goes) and according to a uniform distribution over the bins (the probability that any ball goes to any bin is $1/m$).

(a) What is the probability that the first bin is empty (receives no balls) after n throws? Make sure your answer holds for n=0.

(b) What is the expected number of empty bins?

4. (6 points) Recall that for function $g: \mathbb{N} \to \mathbb{R}^+$, the set $O(g)$ was defined as

$$\{ f \mid (\exists c)(\exists n_0)(\forall n \geq n_0)\ f(n) \leq cg(n) \}$$

Prove or give a counterexample to the following conjecture which simplifies the definition.

CONJECTURE: For any $g: \mathbb{N} \to \mathbb{R}^+$, $O(g) = \{ f \mid (\exists c)(\forall n \in \mathbb{N})\ f(n) \leq cg(n) \}$.

5. (4 points) Do **EXERCISE *5.1-3*** on pg. 94 of our text. (Hint: Find two events that occur with equal probabilities.) The second part of the problem (computing the expected running time) counts as 5 points of extra credit.

CS525DAs
HW#2 SOLUTIONS

1. (a) The following algorithm solves the problem in time $O(n)$, thus establishing this upper bound on the complexity of the problem.

NONMEMBER(A)
    $big \leftarrow 1$
    for $j \leftarrow 2$ **to** n
        do if $A[big]<A[j]$ **then** $big \leftarrow j$ /* $A[big]$ is a largest element of $A[1..j]$ */
    return $A[big]+1$

(b) To establish the linear time lower bound, we show by contradiction that no algorithm can solve the problem is less than linear time. Assume some algorithm solves the problem for input array A is less than linear time, and assume that it returns some value x. The algorithm can not have examined every element of A (or else it would have used at least linear time). Let $i?$ be an index such that the algorithm never examined $A[i?]$. Modify A by putting x into $A[i?]$. The algorithm will behave in the same way (every element it examined before is unchanged), and it will still return x, which is now incorrect.

2. Guess (and verify) in linear time that A is sorted. If A is sorted, return the middle value, otherwise sort it and return the middle value.

    $sorted \leftarrow$ true
    for $k \leftarrow 1$ **to** n-1 **do if** $A[k]>A[k+1]$**then** $sorted \leftarrow$ false
    if not $sorted$ **then** INSERTIONSORT(A)
    return $A[(n+1)/2]$

3. (a) For any ball, the probability that it does not go to the first bin is $(m-1)/m$. Since the throws are independent, the probability that no balls go to the first bin is $\left(\frac{m-1}{m}\right)^n$.

(b) We define n random variables $X_i = \begin{cases} 1 \text{ if bin } i \text{ is empty} \\ 0 \text{ if bin } i \text{ is not empty} \end{cases}$, $1 \le i \le n$, and rv $X = \sum_{1 \le i \le n} X_i$ counts the number of empty bins. From part (a) we get $E[X_i] = 1*\Pr\{X_i = 1\} + 0*\Pr\{X_i = 0\} = \left(\frac{m-1}{m}\right)^n$. From the linearity of expectation, $E[X] = E\left[\sum_{1 \le i \le n} X_i\right] = \sum_{1 \le i \le n} E[X_i] = \sum_{1 \le i \le n}\left(\frac{m-1}{m}\right)^n = n\left(\frac{m-1}{m}\right)^n$. Note that if $n=m$, then $E[X] = n\left(1-\frac{1}{n}\right)^n$, and $\lim_{n\to\infty} n\left(1-\frac{1}{n}\right)^n = \frac{n}{e}$, so about 1/3 of the bins are empty.

4. Assume that $(\exists c)(\exists n_0)(\forall n \geq n_0)\, f(n) \leq cg(n)$, and let $c^* = \max\left(c, \frac{f(0)}{g(0)}, \ldots, \frac{f(n_0)}{g(n_0)}\right)$.

There are two cases:

- $n \leq n_0$ Since $c^* \geq \frac{f(n)}{g(n)}$, then $cg(n) \geq \frac{f(n)}{g(n)} g(n) = f(n)$.
- $n > n_0$ Since $c^* \geq c$, then $c^* g(n) \geq cg(n) \geq f(n)$.

In either case, it follows that $(\forall n \in \mathbb{N}) c^* g(n) \geq f(n)$.

Assume that $(\exists c)(\forall n \in \mathbb{N})\, f(n) \leq cg(n)$. Choosing $n_0 = 0$ it follows that $(\exists c)(\exists n_0)(\forall n \geq n_0)\, f(n) \leq cg(n)$

5. Noting that for a pair of flips, if *x*=*flip*() and *y*=*flip*(), then $\Pr\{x = Head \wedge y = Tail\} = \Pr\{x = Tail \wedge y = Head\}$. This translates into the algorithm

repeat forever
  x ← *flip*()
  y ← *flip*()
  if *x*?*y* **then return** *x*

Extra Credit: The probability of "succeeding" (returning *Head* or *Tail*) is $2p(1-p)$, and the probability of "failing" (getting 2 *Head*s or 2 *Tail*s, and having to continue) is $p^2 + (1-p)^2$. To simplify the notation, let $p^* = 2p(1-p)$ denote the probability that a pair of trials suffices, and $q^* = p^2 + (1-p)^2$ denote the probability that a pair of trials doesn't suffice and we must continue. Hence, the probability of needing exactly 1 pair of trials is p^*, the probability of needing exactly 2 pairs of trials is $q^* p^*$, the probability of needing exactly 3 pairs of trials is $(q^*)^2 p^*$, and the probability of needing exactly $k \geq 1$ pairs of trials is $(q^*)^{k-1} p^*$, which is the probability of k-1 failures followed by a success. (By the way, this is the geometric distribution, which is discussed in SECTION C.4, PG. 112, of our text.) The expected number of trials is

$$\sum_{1 \leq k} k (q^*)^{k-1} p^* = p^* \sum_{1 \leq k} k (q^*)^{k-1} = \frac{p^*}{q^*} \sum_{1 \leq k} k (q^*)^k$$

There are a couple of ways to derive a closed form for this summation. By equation C.31 of our text, the expected number of pairs of *flip*s called by the program is $1/p^* = 1/2p(1-p)$, and obviously the expected number of *flip*s is $1/p(1-p)$.