CS584
HW#1

DUE: Wednesday, September 17

1. (10 points) Suppose you are given a list of *n* numbers, $a_1,...,a_n$, and you want to test if any number appears twice in the list.

a Give an algorithm to solve this problem using worst-case $O(n^2)$ time and $O(1)$ space (beyond the space needed to store $a_1,...,a_n$).

b Give an algorithm to solve this problem using worst-case $O(n\log n)$ time and $O(n)$ space.

c Prove a $\Omega(n)$ bound on the complexity of solving this problem.

2. (5 points) Define $O_{1\text{var}}$ by

$$O_{1\text{var}}(g(n)) = \{ f(n) : \text{there exists positive constant } c \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \in \mathbb{N} \}$$

Either prove or give a counter-example to the following:

CONJECTURE: For all $g : \mathbb{N} \to \mathbb{R}^+$, $O(g(n)) = O_{1\text{var}}(g(n))$.

3. (18 points) Let $A[1..n]$ be an permutation of $(1,...,n)$. An *inversion* is a pair (i, j) such that $1 \le i < j \le n$ but $A[i] > A[j]$. For each $1 \le j \le n$ we define $\iota(j)$ to be the number of i such that (i, j) is an inversion. So for $A = (5,2,4,1,3)$ we have $\iota(1) = 0$, $\iota(2) = 1$, $\iota(3) = 1$, $\iota(4) = 3$ and $\iota(5) = 2$. For array *A*, the *inversion table*, $\tau(A)$, is an array $(\iota(1),...,\iota(n))$. The inversion table of $A = (5,2,4,1,3)$ is $(0,1,1,3,2)$.

a For each $j, 1 \le j \le n$, what are the possible values of $\iota(j)$?

b What are the possible inversion tables?

c How many inversion tables are there?

d Describe a simple function which associates with each inversion table the number of permutations which can give rise to it?

e For each inversion table, describe an algorithm to construct the set of permutations of *A* which can give rise to that inversion table. For example, one permutation of *A* which could give rise to the inversion table $(3,1,2,1,0)$ would be

n	1	2	3	4	5
$f(n)$	4	2	5	3	1

, meaning the permutation of five distinct numbers in which the smallest element is in the $f(1)$ =4th position, the 2nd smallest element is in the $f(2)$ =2nd position,…

f Analyze the work done by INSERTIONSORT(*A*) as a function of $\tau(A)$.

4. (7 points) Let $A[1..n]$ be drawn from a uniform distribution over the set of all permutations of n distinct numbers. Analyze the expected number of pairwise comparisons of the following algorithm. The comparisons we are counting are $A[1]<A[2]$, $Big < A[i]$ and $Little > A[i]$. Give an exact answer (not using asymptotic notation), and show the analysis (do not simply give the answer).

```
if A[1]<A[2] then {Big ← A[2]; Little ← A[1]}
             else {Big ← A[1]; Little ← A[2]}
for i ← 3 to n do
        if Big < A[i] then Big ← A[i]
        else if Little > A[i] then Little ← A[i]
```

5. (10 points) Assume you want to allocate the rental of an electron microscope for a time between t=0 and $t=\delta$, among a series of n requests, $\{(\sigma_1,\tau_1),...,(\sigma_n,\tau_n)\}$, where each request satisfies $0 \le \sigma_i \le \tau_i \le \delta$ for $1 \le i \le n$. If a request (σ_i,τ_i) is granted, then it has sole access to the microscope for the time interval between $t=\sigma_i$ and $t=\tau_i$.

a Describe an algorithm to maximize the number of requests which are granted. The worst-case time complexity of your algorithm should be bounded from above by a polynomial in n.

b Prove that your algorithm works. That is, prove that it always grants a maximal number of requests.

C.S.584
Solutions for H.W. #1

a *Duplicate?* $\leftarrow$ false
for $i \leftarrow 1$ to *n*-1 do
for $j \leftarrow i+1$ to *n* do
if $a_i = a_j$ **do** *Duplicate?* $\leftarrow$ true
return Duplicate

b HEAPSORT($a_1, ..., a_n$)
Duplicate? $\leftarrow$ false
for $i \leftarrow 1$ to *n*-1 do
if $a_i = a_{i+1}$ **do** *Duplicate?* $\leftarrow$ true
return Duplicate?

c Assume there exists an algorithm *A* to test for duplicates in $o(n)$ time. When presented with $a_i = i$ for $1 \le i \le n$, algorithm *A* returns false. But there must be some $a_j, 1 \le j \le n$, such that *A* never examines a_i. We keep all the values of $a_1, ..., a_n$ the same except that $a_j = \begin{cases} j-1, \text{if } j > 1 \\ 2, \text{if } j = 1 \end{cases}$. This will not change the behavior of *A* (it never examined a_j before, and it won't examine a_j in the new data set), so *A* must still return false, though now it's incorrect. So we have contradicted the existence of *A*, and by contradiction the complexity of the problem must be $\Omega(n)$.

2. To show that $O(g(n)) \subseteq O_{1\text{var}}(g(n))$, assume that $f(n) \in O(g(n))$. There exist c, n_0 such that $f(n) \le cg(n)$ for all $n \ge n_0$. Choose $c^* = \max\left\{c, \frac{f(1)}{g(1)}, ..., \frac{f(n_0)}{g(n_0)}\right\}$, and $f(n) \le c^* g(n)$ for all $n \in \mathbb{N}$.

To show that $O_{1\text{var}}(g(n)) \subseteq O(g(n))$, assume that $f(n) \in O_{1\text{var}}(g(n))$. There exists *c* such that $f(n) \le cg(n)$ for all $n \in \mathbb{N}$. Choosing $n_0 = 1$ it follows that $f(n) \le cg(n)$ for all $n \ge n_0$.

3. ***a*** Any subset of the *j*-1 elements which are smaller than *j* can be to its left, so $0 \le \iota(j) \le j-1$.

b For inversion table $(\iota(1),...,\iota(n))$, each $\iota(i)$ is independent of all the other values. In fact, $A[j]$ can be chosen to fall into any of the previous places/intervals. So any array of the form $(x_1,...,x_n)$, where each x_j is an integer satisfying $0 \le x_j \le j-1$, is an inversion table.

c Since each x_j can assume any of *j* values, independent of all the others, the number of possible inversion tables is $\prod_{1\le j\le n} j = n!$

d 1 There's a bijection between permutations and inversion tables.

e **for** $k \leftarrow n$ **downto** 1 **do**
place *k* to the left of $\iota(k)$ numbers already listed

f INSERTIONSORT(*A*) does $\sum_{1\le k\le n} \iota(k)$ swaps, so it's the work is $\Theta\left(n + \sum_{1\le k\le n} \iota(k)\right)$.

4. $\mathbb{A}[1] < \mathbb{A}[2]$ is executed 1 time, and $\mathbb{B}ig < \mathbb{A}[i]$ is executed *n*-2 times. $\mathbb{L}ittle > \mathbb{A}[i]$ is executed once for every time that $\mathbb{B}ig < \mathbb{A}[i]$ is false. This occurs whenever *A*[*i*] is not the largest of the first *i* numbers. Since the numbers are distinct and the permutation of the first *i* numbers is random, the probability of this event is $1-\frac{(i-1)!}{i!} = 1-\frac{1}{i}$. For $3 \le i \le n$,letting rv X_i be 1 if $\mathbb{L}ittle > \mathbb{A}[i]$ is executed and 0 otherwise (this is called an *indicator rv*), we see that $E[X_i] = 1-\frac{1}{i}$, and, by linearity of expectation, the expected total number of executions of that test is
$E\left[\sum_{3\le i\le n} X_i\right] = \sum_{3\le i\le n} E[X_i] = \sum_{3\le i\le n}\left(1-\frac{1}{i}\right) = n-2-\left(H_n-\frac{3}{2}\right) = n-H_n-\frac{1}{2}$. So the expected total number of comparisons of the program is $2n-H_n-\frac{3}{2}$.

5. ***a*** HEAPSORT $\{(\sigma_1,\tau_1),...,(\sigma_n,\tau_n)\}$ in order of increasing τ_i $O(n\lg n)$
for each request, grant it if it starts no earlier than the previously granted request finishes $\Theta(n)$

So the time complexity of the algorithm is in $O(n\lg n)$.

b Assume that the algorithm of part ***a*** grants a set of requests $\{(\sigma_{i_1},\tau_{i_1}),...,(\sigma_{i_p},\tau_{i_p})\}$ and the set of requests $\Lambda = \{(\sigma_{j_1},\tau_{j_1}),...,(\sigma_{j_q},\tau_{j_q})\}$ is feasible (none of the intervals overlap), and that *q*>*p*. By the nature of the algorithm, $\tau_{i_1} \le \tau_{j_1}$, and $(\sigma_{j_1},\tau_{j_1})$ can be replaced by $(\sigma_{i_1},\tau_{i_1})$ in Λ and Λ will still be a feasible schedule. Iteratively, $(\sigma_{j_2},\tau_{j_2})$ can then be

replaced by $\left(\sigma_{i_2}, \tau_{i_2}\right)$ in Λ,…, until $\left(\sigma_{j_p}, \tau_{j_p}\right)$ can then be replaced by $\left(\sigma_{i_p}, \tau_{i_p}\right)$ in Λ. But after the algorithm granted requests $\left\{\left(\sigma_{i_1}, \tau_{i_1}\right),...,\left(\sigma_{i_p}, \tau_{i_p}\right)\right\}$, it would have granted $\left(\sigma_{j_p+1}, \tau_{j_p+1}\right)$, which is a contradiction. So the greedy algorithm of ***a*** must grant a maximum number of requests.