Introduction to Algorithms

Exercise Class

Homework 2

Class Exercise 1

Homework 3

2021-03-26

Homework 2

Exercise 3.1-4

Is $2^{n+1} = O(2^n)$? Is $2^{2n} = O(2^n)$?

$2^{n+1} = O(2^n)$, but $2^{2n} \neq O(2^n)$.

To show that $2^{n+1} = O(2^n)$, we must find constants $c, n_0 > 0$ such that $0 \leq 2^{n+1} \leq c \cdot 2^n$ for all $n \geq n_0$.

Since $2^{n+1} = 2 \cdot 2^n$ for all n, we can satisfy the definition with $c = 2$ and $n_0 = 1$.

To show that $2^{2n} \neq O(2^n)$, assume there exist constants $c, n_0 > 0$ such that $0 \leq 2^{2n} \leq c \cdot 2^n$ for all $n \geq n_0$.

Then $2^{2n} = 2^n \cdot 2^n \leq c \cdot 2^n \Rightarrow 2^n \leq c$. But no constant is greater than all 2^n, and so the assumption leads to a contradiction.

PROBLEM 3-4

Asymptotic notation properties. Let $f(n)$ and $g(n)$ be asymptotically positive functions. Prove or disprove each of the following conjectures.

a. $f(n) = O(g(n))$ implies $g(n) = O(f(n))$.

b. $f(n) + g(n) = \Theta(\min(f(n), g(n)))$.

c. $f(n) = O(g(n))$ implies $\lg(f(n)) = O(\lg(g(n)))$.
where $\lg(g(n)) \geq 1$ and $f(n) \geq 1$ for all sufficiently large n.

d. $f(n) = O(g(n))$ implies $2^{f(n)} = O\left(2^{g(n)}\right)$.

e. $f(n) = O\left((f(n))^2\right)$.

f. $f(n) = O(g(n))$ implies $g(n) = \Omega(f(n))$.

g. $f(n) = \Theta(f(n/2))$.

h. $f(n) + o(f(n)) = \Theta(f(n))$.

Problem 3-4

a. $f(n) = O(g(n))$ implies $g(n) = O(f(n))$.

False. Counterexample: $n = O\left(n^2\right)$ but $n^2 \neq O(n)$.

b. $f(n) + g(n) = \Theta(\min(f(n), g(n)))$.

False. Counterexample: $n + n^2 \neq \Theta(n)$.

c. $f(n) = O(g(n))$ implies $\lg(f(n)) = O(\lg(g(n)))$.
where $\lg(g(n)) \geq 1$ and $f(n) \geq 1$ for all sufficiently large n.

True. Since $f(n) = O(g(n))$ there exist c and n_0 such that $n \geq n_0$ implies $f(n) \leq cg(n)$ and $f(n) \geq 1$. This means that $\log(f(n)) \leq \log(cg(n)) = \log(c) + \log(g(n))$. Note that the inequality is preserved after taking logs because $f(n) \geq 1$. Now we need to find d such that $f(n) \leq d\log(g(n))$. It will suffice to make $\log(c) + \log(g(n)) \leq d\log(g(n))$, which is achieved by taking $d = \log(c) + 1$, since $\log(g(n)) \geq 1$.

Problem 3-4

d. $f(n) = O(g(n))$ implies $2^{f(n)} = O\left(2^{g(n)}\right)$.

False. Counterexample: $2n = O(n)$ but $2^{2n} \neq O(2^n)$ as shown in exercise 3.1-4.

e. $f(n) = O\left((f(n))^2\right)$.

False. Counterexample: Let $f(n) = \frac{1}{n}$. Suppose that c is such that $\frac{1}{n} \leq c\frac{1}{n^2}$ for $n \geq n_0$. Choose k such that $k \cdot c \geq n_0$ and $k > 1$. Then this implies $\frac{1}{k \cdot c} \leq \frac{c}{k^2 \cdot c^2} = \frac{1}{k^2 \cdot c}$, a contradiction.

f. $f(n) = O(g(n))$ implies $g(n) = \Omega(f(n))$.

True. Since $f(n) = O(g(n))$ there exist c and n_0 such that $n \geq n_0$ implies $f(n) \leq cg(n)$. Thus $g(n) \geq \frac{1}{c}f(n)$, so $g(n) = \Omega(f(n))$.

g. $f(n) = \Theta(f(n/2))$.

False. Counterexample: Let $f(n) = 2^{2n}$. By exercise 3.1-4, $2^{2n} \neq O(2^n)$.

h. $f(n) + o(f(n)) = \Theta(f(n))$.

True. Let g be any function such that $g(n) = o(f(n))$. Since g is asymptotically positive let n_0 be such that $n \geq n_0$ implies $g(n) \geq 0$. Then $f(n) + g(n) \geq f(n)$ so $f(n) + o(f(n)) = \Omega(f(n))$. Next, choose n_1 such that $n \geq n_1$ implies $g(n) \leq f(n)$. Then $f(n) + g(n) \leq f(n) + f(n) = 2f(n)$ so $f(n) + o(f(n)) = O(f(n))$. By Theorem 3.1, this implies $f(n) + o(f(n)) = \Theta(f(n))$.

Give the asymptotic tight bound of $\log(n!)$

Answer: $\log(n!) = \Theta(n\log(n))$.
We have that

$$\log(n!) = \log(1) + \log(2) + \ldots + \log(n-1) + \log(n)$$

So the upper bound is

$$\begin{aligned}\log(1) + \log(2) + \ldots + \log(n) &\le \log(n) + \log(n) + \ldots + \log(n)\\ &= n\log(n)\end{aligned}$$

The lower bound is:

$$\begin{aligned}
&\log(1) + \ldots + \log\left(\frac{n}{2}\right) + \ldots + \log(n) \\
&\geq \log\left(\frac{n}{2}\right) + \ldots + \log(n) \\
&= \log\left(\frac{n}{2}\right) + \log\left(\frac{n}{2} + 1\right) + \ldots + \log(n-1) + \log(n) \\
&\geq \log\left(\frac{n}{2}\right) + \ldots + \log\left(\frac{n}{2}\right) \\
&= \frac{n}{2}\log\left(\frac{n}{2}\right)
\end{aligned}$$

Actually, there is **Stirling's approximation**:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

$$\log(n!) \sim n\log n - n + \frac{1}{2}\log \pi n$$

For an given sequence $a_1, a_2, \ldots, a_n$ real numbers, find (i, j) that $\sum_{k=i}^{j} a_k (1 \leq i \leq j \leq n)$ is maximum. Present two different algorithms for the above problem as well as corresponding performance analysis.

Brute-Force Approach:

MaxSubarray (A)
$n = A.length$
$maximum = 0$
for $i = 1$ **to** n
    $current = 0$
    for $j = i$ **to** n
        $current = current + A[j]$
        if $current > maximum$
            $maximum = current$
return $maximum$

Time Complexity: $\Theta(n^2)$

4.2-3

How would you modify Strassen's algorithm to multiply $n \times n$ matrices in which n is not an exact power of 2 ? Show that the resulting algorithm runs in time $\Theta\left(n^{\lg 7}\right)$.

You could pad out the input matrices to be powers of two and then run the given algorithm. Padding out the the next largest power of two (call it m) will at most double the value of n because each power of two is off from each other by a factor of two. So, this will have runtime

$$m^{\lg 7} \leq (2n)^{\lg 7} = 7n^{\lg 7} \in O\left(n^{\lg 7}\right)$$

and

$$m^{\lg 7} \geq n^{\lg 7} \in \Omega\left(n^{\lg 7}\right)$$

Putting these together, we get the runtime is $\Theta\left(n^{\lg 7}\right)$.

4.3-4

Show that by making a different inductive hypothesis, we can overcome the difficulty with the boundary condition $T(1) = 1$ for recurrence (4.19) without adjusting the boundary conditions for the inductive proof.

Solution 1

We'll use the induction hypothesis $T(n) \le 2n \log n + 1$. First observe that this means $T(1) = 1$, so the base case is satisfied. Then we have

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\le 2((2n/2)\log(n/2) + 1) + n \\ &= 2n\log(n) - 2n\log 2 + 2 + n \\ &= 2n\log(n) + 1 + n + 1 - 2n \\ &\le 2n\log(n) + 1 \end{aligned}$$

Solution 2

We guess $T(n) \leq n \lg n + n$

$$\begin{aligned} T(n) &\leq 2(c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor + \lfloor n/2 \rfloor) + n \\ &\leq 2c(n/2)\lg(n/2) + 2(n/2) + n \\ &= cn\lg(n/2) + 2n \\ &= cn\lg n - cn\lg 2 + 2n \\ &= cn\lg n + (2-c)n \\ &\leq cn\lg n + n \end{aligned}$$

where the last step holds for $c \geq 1$.

This time, the boundary condition is

$$T(1) = 1 \leq cn\lg n + n = 0 + 1 = 1$$

Solve the recurrent $T(n) = T(\sqrt{n}) + \theta(n)$

Observe that if we let some value $m = \log_2 n$, then $n = 2^m$ by the definition of the logarithm, then $T(2^m) = 2T((2^m)^{1/2}) + 2^m$. Create a new recurrence $S(m) = T(2^m)$. Then, $S(m) = 2S(m/2) + 2^m$. We will now try to show this falls into one of the cases of the Master Theorem, which requires recurrences of the form $S(m) = a \cdot S(m/b) + f(m)$ for the recursive case (assuming base case above is a constant).

Note that 2^m is not a polynomial, so $2^m \notin \Theta(m^{log_2 2}(\log_2 m)^k)$ for any constant $k \geq 0$, and $2^m \notin O(m^{\log_2(2)-\epsilon})$ for any $\epsilon > 0$. So we are left with one case, easily the hardest one to show, which requires me to show the regularity condition holds, that is the following:

Let there be a growth function $f(m)$. *If there exists a real constant* $c < 1$ *and a constant* $n_0 \in \mathbb{Z}^+$, *such that for all m*$\geq n_0$, *if* $a \cdot f(m/b) \leq c \cdot f(m)$, *then the regularity condition is satisfied for* $f(m)$.

Note that in $S(m)$, $a = 2$, and $b = 2$. So we need to find a $c < 1$ and n_0, such that for all $m \geq n_0$, $2 \cdot 2^{m/2} \leq c \cdot 2^m$. Applying logarithms to both sides of the inequality, $\log_2\left(2 \cdot 2^{m/2}\right) \leq \log_2\left(c \cdot 2^m\right)$, so apply the product rule of logarithms (then the power rule of logarithms),
$\log_2 2 + \log_2 2^{m/2} \leq \log_2 c + \log_2 2^m \Leftrightarrow 1 + m/2 \leq \log_2 c + m$. Choose $c = 1/2$, then $\log_2 c = -1$, and we need to find n_0 such that for all $m \geq n_0$, $1 + m/2 \leq (-1) + m \Leftrightarrow 2 \leq m/2 \Leftrightarrow m \geq 4$. So choose $n_0 = 4$. Hence, the regularity condition holds for $f(m) = 2^m$.

Now we continue with trying to apply the Master Theorem. Since the regularity condition holds, we now only need to show $2^m \in \Omega(m^{\log_2 2+\epsilon})$, where $\epsilon > 0$ is constant. Pick $\epsilon = 1$, and $2^m \in \Omega(m^2)$ (this should be straightforward to show using the definition of Big-Omega). Hence, we apply this case of the Master Theorem, and get $S(m) \in \Theta(2^m)$.

Applying our change of variable, recall that $n = 2^m$. Therefore, $T(n) \in \Theta(n)$.

$A[1, 2, \ldots, n]$ is an increasing-ordered array with all distinct integers (possibly be negative). Find an algorithm by Divide and Conquer to find i that $A[i] = i$. The worst-case running time of your algorithm should be in $O(\lg n)$

```
FindMatch(A)
l = 1
r = n
while l ≤ r
    mid = ⌊(l + r)/2⌋
    if A[mid] == mid
        return mid
    elif A[mid] < mid
        l = mid + 1
    else
        r = mid − 1
return 0
```

```
FindMatch(l, r)
if l > r
    return 0
⌊mid = (l + r)/2⌋
if A[mid] == mid
    return mid
elif A[mid] < mid
    return FindMatch(mid + 1, r)
else
    return FindMatch(l, mid − 1)
```

Class Exercise

1. Problem 4-1

Recurrence examples Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$. Make your bounds as tight as possible, and justify your answers.

a. $T(n) = 2T(n/2) + n^3$.

b. $T(n) = T(9n/10) + n$.

c. $T(n) = 16T(n/4) + n^2$.

d. $T(n) = 7T(n/3) + n^2$.

e. $T(n) = 7T(n/2) + n^2$.

f. $T(n) = 2T(n/4) + \sqrt{n}$.

g. $T(n) = T(n-1) + n$

h. $T(n) = T(\sqrt{n}) + 1$.

a. $T(n) = 2T(n/2) + n^3 = \Theta\left(n^3\right)$.

This is a divide-and-conquer recurrence with $a = 2$, $b = 2$, $f(n) = n^3$, and $n^{\log_b a} = n^{\log_2 2} = n$. Since $n^3 = \Omega\left(n^{\log_2 2+2}\right)$ and $a/b^k = 2/2^3 = 1/3 < 1$, case 3 of the master theorem applies, and $T(n) = \Theta\left(n^3\right)$.

b. $T(n) = T(9n/10) + n = \Theta(n)$.

This is a divide-and-conquer recurrence with $a = 1$, $b = 10/9$, $f(n) = n$, and $n^{\log_b a} = n^{\log_{10/9} 1} = n^0 = 1$. Since $n = \Omega\left(n^{\log_{10/9} 1+1}\right)$ and $a/b^k = 1/(10/9)^1 = 9/10 < 1$, case 3 of the master theorem applies, and $T(n) = \Theta(n)$.

c. $T(n) = 16T(n/4) + n^2 = \Theta\left(n^2 \lg n\right)$.

This is another divide-and-conquer recurrence with $a = 16$, $b = 4$, $f(n) = n^2$, and $n^{\log_b a} = n^{\log_4 16} = n^2$. Since $n^2 = \Theta\left(n^{\log_4 16}\right)$, case 2 of the master theorem applies, and $T(n) = \Theta\left(n^2 \lg n\right)$.

d. $T(n) = 7T(n/3) + n^2 = \Theta\left(n^2\right)$.

This is a divide-and-conquer recurrence with $a = 7$, $b = 3$, $f(n) = n^2$, and $n^{\log_b a} = n^{\log_3 7}$. Since $1 < \log_3 7 < 2$, we have that $n^2 = \Omega\left(n^{\log_3 7 + \epsilon}\right)$ for some constant $\epsilon > 0$. We also have $a/b^k = 7/3^2 = 7/9 < 1$, so that case 3 of the master theorem applies, and $T(n) = \Theta\left(n^2\right)$.

e. $T(n) = 7T(n/2) + n^2 = O\left(n^{\lg 7}\right)$.

This is a divide-and-conquer recurrence with $a = 7$, $b = 2$, $f(n) = n^2$, and $n^{\log_b a} = n^{\log_2 7}$. Since $2 < \lg 7 < 3$, we have that $n^2 = O\left(n^{\log_2 7 - \epsilon}\right)$ for some constant $\epsilon > 0$. Thus, case 1 of the master theorem applies, and $T(n) = \Theta\left(n^{\lg 7}\right)$.

f. $T(n) = 2T(n/4) + \sqrt{n} = \Theta(\sqrt{n} \lg n)$.

This is another divide-and-conquer recurrence with $a = 2$, $b = 4$, $f(n) = \sqrt{n}$, and $n^{\log_b a} = n^{\log_4 2} = \sqrt{n}$. Since $\sqrt{n} = \Theta\left(n^{\log_4 2}\right)$, case 2 of the master theorem applies, and $T(n) = \Theta(\sqrt{n} \lg n)$.

1. Problem 4-1

h. $T(n) = T(\sqrt{n}) + 1$.

Let $m = \lg n$ and $S(m) = T(2^m)$. $T(2^m) = T\left(2^{m/2}\right) + 1$, so $S(m) = S(m/2) + 1$. Using the master theorem, $n^{\log_b a} = n^{\log_2 1} = n^0 = 1$ and $f(n) = 1$. Since $1 = \Theta(1)$, case 2 applies and $S(m) = \Theta(\lg m)$. Therefore, $T(n) = \Theta(\lg \lg n)$.

Assume you have functions f and g such that $\mathbf{f(n)}$ **is** $\mathbf{O(g(n))}$. For each of the following statements, decide whether you think it **is true or false** and give **a proof** or **counterexample**.

a) $\log_2 f(n)$ is $O\left(\log_2 g(n)\right)$

b) $2^{f(n)}$ is $O\left(2^{g(n)}\right)$

c) $f(n)^2$ is $O\left(g(n)^2\right)$

a) $\log_2 f(n)$ is $O\left(\log_2 g(n)\right)$

By assumption there exist $N \in \mathbb{N}$ and $c \in \mathbb{R}_{>0}$ such that for all $n \in \mathbb{N}$ with $n \geq N$ we have

$$0 \leq f(n) \leq cg(n)$$

But then, since $\log_2$ is order-preserving:

$$\begin{aligned} \log_2 f(n) &\leq \log_2 cg(n) \\ &= \log_2 c + \log_2 g(n) \end{aligned}$$

That looks almost OK. We want to find a $d \in \mathbb{R}_{>0}$ such that

$$\log_2 f(n) \leq d \log_2 g(n)$$

Using the previous, it is sufficient to show:

$$\log_2 c + \log_2 g(n) \leq d \log_2 g(n)$$

And this is OK, if:

$$\frac{\log_2 c}{\log_2 g(n)} + 1 \leq d$$

However, $\log_2 g(n)$ might get closer and closer to 0 while n gets bigger. This leads us to the following counterexample:

$$2\left(1 + \frac{1}{n}\right) \in O\left(1 + \frac{1}{n}\right)$$

but

$$\log_2 2 + \log_2\left(1 + \frac{1}{n}\right) \notin O\left(\log_2\left(1 + \frac{1}{n}\right)\right)$$

b) $2^{f(n)}$ is $O\left(2^{g(n)}\right)$

We have $2n \in O(n)$, however $2^{2n} \notin O\left(2^n\right)$.

c) $f(n)^2$ is $O\left(g(n)^2\right)$

By assumption there exist $N \in \mathbb{N}$ and $c \in \mathbb{R}_{>0}$ such that for all $n \in \mathbb{N}$ with $n \geq N$ we have

$$0 \leq f(n) \leq cg(n)$$

But then, since squaring is order-preserving (on positive values), also:

$$\begin{aligned} 0 = 0^2 \leq f(n)^2 &\leq (cg(n))^2 \\ &= c^2 g(n)^2. \end{aligned}$$

Thus $f(n)^2 \in O\left(g(n)^2\right)$.

3. PERMUTE-BY-SORTING

Prove that in the array P in procedure PERMUTE-BY-SORTING, the probability that all elements are unique is at least $1 - 1/n$. Hint: $(1 - a)(1 - b) > 1 - a - b$, $(a, b \geq 0)$.

For i, j such that $1 \leq i < j \leq n$, let E_{ij} denote the event that elements $P[i]$ and $P[j]$ are identical. Since the elements in P are chosen independently and uniformly at random from values 1 to n^3, we have $\Pr\left(E_{ij}\right) = 1/n^3$ for all pairs i, j. The event that not all elements are unique, that is, there is at least one pair of identical elements, is $\bigcup_{i<j} E_{ij}$. Therefore, the probability that all elements are unique is

$$\begin{aligned}
\Pr\left(\left(\bigcup_{i<j} E_{ij}\right)^c\right) &= 1 - \Pr\left(\bigcup_{i<j} E_{ij}\right) \\
&\geq 1 - \sum_{i<j} \Pr\left(E_{ij}\right) \\
&= 1 - \frac{n(n-1)}{2} \cdot \frac{1}{n^3} \\
&= 1 - \frac{1}{2n} + \frac{1}{2n^2} \\
&\geq 1 - 1/n
\end{aligned}$$

The first inequality follows from direct usage of the union bound (also known as Boole's

Suppose you are given an array A holding $\boldsymbol{n}$ distinct numbers. You are told that the sequence of values $A[1], A[2], \ldots, A[n]$ is **uni-modal**: for some index $\boldsymbol{p}$ between 1 and n, the values in the array entries increase up to position $\boldsymbol{p}$ in A and then decrease the remainder all the way until position $\boldsymbol{n}$. Please give an efficient algorithm to find the "**peak entry** $\boldsymbol{p}$ ". Describe your algorithm in pseudo-code and **analyze** your algorithm.

Unimodal(A, low, $high$)

if $low = high - 1$

  return $A[low]$

$mid = \lfloor (high + low)/2 \rfloor$

if $A[mid] < A[mid+1]$

  return Unimodal(A, $mid + 1$,$high$)

else

  return Unimodal(A, low, $mid + 1$)

Time Complexity: $\Theta(\log n)$

We are given a sequence of numbers $a_1, a_2, \ldots a_n$. We call a pair a **significant inversion** if $i < j$ and $a_i > 2\, a_j$. Give an $O(n \log n)$ algorithm using divide-and conquer to **count** the number of significant inversions in the given sequence.

COUNT-INVERSIONS(A, p, r)
inversions $= 0$
if $p < r$
    $q = \lfloor (p + r)/2 \rfloor$
    inversions $=$ *inversions*+COUNT-INVERSIONS(A, p, q)
    inversions $=$ *inversions*+COUNT-INVERSIONS$(A, q + 1, r)$
    inversions $=$ *inversions*+Merge-INVERSIONS(A, p, q, r)
return *inversions*

```
Merge-INVERSIONS(A, p, q, r)
n1 = q − p + 1
n2 = r − q
let L[1 . . . n1 + 1] and
R[1 . . . n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i − 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
inversions = 0
counted = FALSE
for k = p to r
    if counted == FALSE and L[i] > 2 · R[j]
        inversions = inversions + n1 − i + 1
        counted = TRUE
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
        counted = FALSE
return inversions
```

Homework 3

Use indicator random variables to solve the following problem, which is known as the ***hat-check problem***. Each of n customers gives a hat to a hat-check person at a restaurant. The hat-check person gives the hats back to the customers in a random order. What is the expected number of customers who get back their own hat?

5.2-4

Another way to think of the hat-check problem is that we want to determine the expected number of fixed points in a random permutation. (A **fixed point** of a permutation π is a value i for which $\pi(i) = i$.) We could enumerate all $n!$ permutations, count the total number of fixed points, and divide by $n!$ to determine the average number of fixed points per permutation. This would be a painstaking process, and the answer would turn out to be 1. We can use indicator random variables, however, to arrive at the same answer much more easily.

Define a random variable X that equals the number of customers that get back their own hat, so that we want to compute $\mathrm{E}[X]$.

For $i = 1, 2, \ldots, n$, define the indicator random variable

$X_i = \mathrm{I}\,\{$ customer i gets back his own hat$\}$

Then $X = X_1 + X_2 + \cdots + X_n$

5.2-4

Since the ordering of hats is random, each customer has a probability of $1/n$ of getting back his or her own hat. In other words, $\Pr\{X_i = 1\} = 1/n$, which, by Lemma 5.1, implies that $\mathrm{E}[X_i] = 1/n$.
Thus,

$$\begin{aligned}
\mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n} X_i\right] \\
&= \sum_{i=1}^{n} \mathrm{E}[X_i] \qquad \text{(linearity of expectation)} \\
&= \sum_{i=1}^{n} 1/n \\
&= 1\ .
\end{aligned}$$

and so we expect that exactly 1 customer gets back his own hat.

5.2-4

Note that this is a situation in which the indicator random variables are *not* independent. For example, if $n = 2$ and $X_1 = 1$, then X_2 must also equal 1. Conversely, if $n = 2$ and $X_1 = 0$, then X_2 must also equal 0. Despite the dependence, $\Pr\{X_i = 1\} = 1/n$ for all i, and linearity of expectation holds. Thus, we can use the technique of indicator random variables even in the presence of dependence.

5.3-4

```
PERMUTE-BY-CYCLIC (A)
n = A.length
let B[1 . . . n] be a new array
offset = RANDOM (A)
for i = 1 to n
    dest = i + offset
    if dest > n
        dest = dest − n
    B[dest] = A[i]
return B
```

Show that each element $A[i]$ has a $1/n$ probability of winding up in any particular position in B. Then show that Professor Armstrong is mistaken by showing that the resulting permutation is not uniformly random.

5.3-4

PERMUTE-BY-CYCLIC chooses offset as a random integer in the range $1 \leq \text{offset} \leq n$, and then it performs a cyclic rotation of the array. That is, $B[((i+\mathit{offset}-1) \bmod n)+1] = A[i]$ for $i = 1, 2, \ldots, n$. (The subtraction and addition of 1 in the index calculation is due to the 1-origin indexing. If we had used 0-origin indexing instead, the index calculation would have simplied to $B[(i+\mathit{offset}) \bmod n] = A[i]$ for $i = 0, 1, \ldots, n-1$).

Thus, once *offset* is determined, so is the entire permutation. Since each value of offset occurs with probability $1/n$, each element $A[i]$ has a probability of ending up in position $B[j]$ with probability $1/n$.

This procedure does not produce a uniform random permutation, however, since it can produce only n different permutations. Thus, n permutations occur with probability $1/n$, and the remaining $n! - n$ permutations occur with probability 0.

Problem 5-1

With a b-bit counter, we can ordinarily only count up to $2^b - 1$. With R. Morris's ***probabilistic counting***, we can count up to a much larger value at the expense of some loss of precision.

a. Show that the expected value represented by the counter after n INCREMENT operations have been performed is exactly n.

b. The analysis of the variance of the count represented by the counter depends on the sequence of the n_i. Let us consider a simple case: $n_i = 100i$ for all $i \geq 0$. Estimate the variance in the value represented by the register after n INCREMENT operations have been performed.

a. To determine the expected value represented by the counter after n INCREMENT operations, we define some random variables:

- For $j = 1, 2, \ldots, n$, let X_j denote the increase in the value represented by the counter due to the jth INCREMENT operation.
- Let V_n be the value represented by the counter after n INCREMENT operations.

Then $V_n = X_1 + X_2 + \cdots + X_n$. We want to compute $\mathrm{E}[V_n]$. By linearity of expectation,

$$\mathrm{E}[V_n] = \mathrm{E}[X_1 + X_2 + \cdots + X_n] = \mathrm{E}[X_1] + \mathrm{E}[X_2] + \cdots + \mathrm{E}[X_n] \; .$$

We shall show that $\mathrm{E}[X_j] = 1$ for $j = 1, 2, \ldots, n$, which will prove that $\mathrm{E}[V_n] = n$.

PROBLEM 5-1

We actually show that $\mathrm{E}[X_j] = 1$ in two ways, the second more rigorous than the first:

1. Suppose that at the start of the jth INCREMENT operation, the counter holds the value i, which represents n_i. If the counter increases due to this INCREMENT operation, then the value it represents increases by $n_{i+1} - n_i$. The counter increases with probability $1/(n_{i+1} - n_i)$, and so

$$\begin{aligned}
\mathrm{E}[X_j] &= (0 \cdot \Pr\{\text{counter does not increase}\}) \\
&\qquad + ((n_{i+1} - n_i) \cdot \Pr\{\text{counter increases}\}) \\
&= \left(0 \cdot \left(1 - \frac{1}{n_{i+1} - n_i}\right)\right) + \left((n_{i+1} - n_i) \cdot \frac{1}{n_{i+1} - n_i}\right) \\
&= 1 ,
\end{aligned}$$

and so $\mathrm{E}[X_j] = 1$ regardless of the value held by the counter.

2. Let C_j be the random variable denoting the value held in the counter at the start of the jth INCREMENT operation. Since we can ignore values of C_j greater than $2^b - 1$, we use a formula for conditional expectation:

$$\begin{aligned} \mathrm{E}[X_j] &= \mathrm{E}[\mathrm{E}[X_j \mid C_j]] \\ &= \sum_{i=0}^{2^b-1} \mathrm{E}[X_j \mid C_j = i] \cdot \Pr\{C_j = i\} . \end{aligned}$$

To compute $\mathrm{E}[X_j \mid C_j = i]$, we note that

- $\Pr\{X_j = 0 \mid C_j = i\} = 1 - 1/(n_{i+1} - n_i)$,
- $\Pr\{X_j = n_{i+1} - n_i \mid C_j = i\} = 1/(n_{i+1} - n_i)$, and
- $\Pr\{X_j = k \mid C_j = i\} = 0$ for all other k.

Thus,

$$\begin{aligned} \mathrm{E}[X_j \mid C_j = i] &= \sum_k k \cdot \Pr\{X_j = k \mid C_j = i\} \\ &= \left(0 \cdot \left(1 - \frac{1}{n_{i+1} - n_i}\right)\right) + \left((n_{i+1} - n_i) \cdot \frac{1}{n_{i+1} - n_i}\right) \\ &= 1 . \end{aligned}$$

PROBLEM 5-1

Therefore, noting that

$$\sum_{i=0}^{2^b-1} \Pr\{C_j = i\} = 1 ,$$

we have

$$\begin{aligned} \mathrm{E}[X_j] &= \sum_{i=0}^{2^b-1} 1 \cdot \Pr\{C_j = i\} \\ &= 1 . \end{aligned}$$

Why is the second way more rigorous than the first? Both ways condition on the value held in the counter, but only the second way incorporates the conditioning into the expression for $\mathrm{E}[X_j]$.

b. Defining V_n and X_j as in part (a), we want to compute $\mathrm{Var}[V_n]$, where $n_i = 100i$. The X_j are pairwise independent, and so by equation (C.28), $\mathrm{Var}[V_n] = \mathrm{Var}[X_1] + \mathrm{Var}[X_2] + \cdots + \mathrm{Var}[X_n]$.

Since $n_i = 100i$, we see that $n_{i+1} - n_i = 100(i+1) - 100i = 100$. Therefore, with probability $99/100$, the increase in the value represented by the counter due to the jth INCREMENT operation is 0, and with probability $1/100$, the value represented increases by 100. Thus, by equation (C.26),

$$
\begin{aligned}
\mathrm{Var}[X_j] &= \mathrm{E}\left[X_j^2\right] - \mathrm{E}^2[X_j] \\
&= \left(\left(0^2 \cdot \frac{99}{100}\right) + \left(100^2 \cdot \frac{1}{100}\right)\right) - 1^2 \\
&= 100 - 1 \\
&= 99\,.
\end{aligned}
$$

Summing up the variances of the X_j gives $\mathrm{Var}[V_n] = 99n$.

Thank you for your attention!

Introduction to Algorithms

TA: Tianlu Fei

2021-03-26