CSc 220: Algorithms
Homework 2 Solutions

Problem 1: A different way to randomize QUICK-SORT is to use the deterministic version of QUICK-SORT over a 'randomized' array, according to the following pseudo-code

PERMUTE-QUICK-SORT(A)
  $B \longleftarrow$ RANDOM-PERMUTE(A);
  RETUR N QUICK-SORT(B)

- Give a sufficient condition on the procedure RANDOM-PERMUTE that will make PERMUTE-QUICK-SORT run in $O(n \log n)$ steps. [6pts]
- Consider the following procedure

  SHIFT-PERMUTE(A)
    $n \leftarrow |A|$;
    $s \leftarrow$ RANDOM$(1, n)$;
    FOR $i = 1$ TO n
      $j \leftarrow s + i \bmod n$;
      $B[j] \leftarrow A[i]$;
    RETURN B

  What is the expected running time of PERMUTE-QUICK-SORT if you use SHIFT-PERMUTE in place of RANDOM-PERMUTE? [4pts]

Solution:

- A sufficient condition for PERMUTE-QUICK-SORT to run in $O(n \log n)$ steps is that the output of RANDOM-PERMUTE is uniformly distributed among all possible permutations of the array A. In other words given a permutation of the elements of A, that permutation is output by RANDOM-PERMUTE with probability $\frac{1}{n!}$.
- We note first of all that SHIFT-PERMUTE does not output all possible $n!$ permutations of A (each with probability $\frac{1}{n!}$). Rather, it outputs only n of those permutations (the one resulting from a simple shift), each with probability $\frac{1}{n}$. This lack of "entropy" in the distribution of the output will not "destroy" a worst-case input causing a running time of $\Theta(n^2)$ on some inputs.

  Consider the case of an array which is already sorted. For $i = 1, \ldots, n$, with probability $1/n$ SHIFT-PERMUTE will bring the m^{th} element to the first slot of the array, i.e. the pivot position. That means that we will partition the array into two arrays of size $m - 1$ and $n - m$ respectively. Because of the way PARTITION works those arrays will also be already sorted: that's because PARTITION reads the array A in order and appends elements to the appropriate array as it reads them. Notice that when the algorithm recurses it does not randomize the array anymore: PERMUTE-QUICK-SORT makes only one random choice at the beginning and then calls the deterministic QUICKSORT algorithm. So the running time of the recursive calls will be quadratic. The expected running time is therefore

  $$\tilde{T}(n) = \frac{1}{n} \sum_{m=1}^{n} [T_{QS}(m-1) + T_{QS}(n-m) + \Theta(n)]$$

  Where $T_{QS}(m)$ is the running time of deterministic quicksort on an array of size m. As we saw in class this is the same as

  $$\tilde{T}(n) = \frac{2}{n} \sum_{m=1}^{n-1} T_{QS}(m) + \Theta(n)$$

We know that $T_{QS}(m) = \Theta(m^2)$ and therefore

$$\tilde{T}(n) = \frac{2}{n}\sum_{m=1}^{n-1}\Theta(m^2) + \Theta(n) = \Theta(m^2)$$

The last step follows from the fact that

$$\sum_{m=1}^{n} m^2 < cn^3$$

for a constant c, which can be proven by induction. Indeed this is true for $n = 1$ since $\sum_{m=1}^{1} m^2 = 1 < cn^3 = c$ for $c > 1$. Then assume it's true for $n - 1$, we have that

$$\sum_{m=1}^{n} m^2 = n^2 + \sum_{m=1}^{n-1} m^2 < n^2 + c(n-1)^3 = cn^3 - (3c-1)n^2 + 3cn - 1 < cn^3$$

provided that

$$(3c-1)n^2 - 3cn + 1 > 0$$

which is true for sufficiently large n.

Problem 2: Let a and b be two n bit numbers (assume for simplicity that n is a power of 2).

- Describe the "grade school" algorithm to multiply a and b and show that it requires $O(n^2)$ steps; [2pts]
- Describe a divide-and-conquer algorithm with an asymptotically faster running time. [8pts]

Solution:

- The grade school algorithm requires pair-wise multiplications of all the digits in the two numbers, which is why it requires $O(n^2)$ steps (since there are n^2 possible pairs). More specifically let $a_0 \dots a_{n-1}$ and $b_0 \dots b_{n-1}$ be bits such that $a = \sum_i a_i 2^i$ and $b = \sum_i b_i 2^i$. Let $c_{ji} = a_j b_i$. We construct n numbers, each n-bit long

  $$c^{(k)} = \sum_i c_{ki} 2^i$$

  and

  $$ab = \sum_k c^{(k)} 2^k$$

  The computation of the c_{ji} takes $O(n^2)$.

- Split a and b into the top and bottom half of the bits. Then

  $$a = \hat{a} + \bar{a}2^{n/2} \quad \text{and} \quad b = \hat{b} + \bar{b}2^{n/2}$$

  where $\hat{a}, \bar{a}, \hat{b}, \bar{b}$ are $n/2$-bit numbers. We have that

  $$ab = \alpha + 2^{n/2}\beta + 2^n\gamma$$

  where

  $$\alpha = \hat{a}\hat{b} \quad \beta = \hat{a}\bar{b} + \bar{a}\hat{b} \quad \gamma = \bar{a}\bar{b}$$

  A trivial implementation of this algorithm would recurse *four times* on inputs of size $n/2$ to compute the four cross products. Since the additions can be done in $\Theta(n)$ time we would have the recurrence

  $$T(n) = 4T(n/2) + \Theta(n) = \Theta(n^2)$$

by the Master Method. So this approach does not improve on the grade school algorithm.

However notice what happens if we compute

$$\delta = (\hat{a} + \bar{a})(\hat{b} + \bar{b}) = \alpha + \beta + \gamma$$

Which means that we can recurse only *three* times on input of size $n/2$ to compute α, β, δ and then compute $\beta = \delta - \alpha - \gamma$ to obtain ab. Since all additions can be computed in $\Theta(n)$ the running time of this algorithm satisfies the recurrence

$$T(n) = 3T(n/2) + \Theta(n) = \Theta(n^{\log 3})$$

which is asymptotically faster than $\Theta(n^2)$.

Problem 3: You are given n samples of a chemical compound. While they look identical, some of them have in fact been contaminated. You have a testing machine that given two samples can detect if they are the same or not. You also know that most of the samples (a majority of them) are identical. Find one of those identical samples making no more than n tests with your machine (a.k.a. comparisons).

Solution: Order the n samples arbitrarily as $s_1, s_2, \ldots, s_n$. Then for each odd i, compare s_i with s_{i+1}. If they are equal, then keep just one of the two samples. If they are different, drop both of them. Repeat the above process until you are left with just one element, and output that element as a member of the majority.

First, let's prove that this algorithm terminates with less than n comparisons. Note indeed that at each step the number of sample is reduced by at least half, so the number of comparisons T is

$$T \leq \frac{n}{2} + \frac{n}{4} + \ldots + 2 \leq n$$

We need to prove that the algorithm is correct. Let m be the number of "majority samples" (m-samples in the rest), so $m > n/2$. The "nonmajority samples" (nm-samples) then are $n - m < n/2$.

Note that at the end of each iteration those samples will still be the majority of the samples left. Let's prove it for the first iteration: the others will follow analogously. To see that, let

- a be the number of pairs composed of a nm-sample and an m-sample: those are both dropped from the pool;
- b the number of pairs composed by both m-samples. Of those 1 sample remains in the pool;
- $c = n/2 - a - b$ the number of pairs composed of both nm-samples. We do not know exactly what happens in this case (depends if the pair is of identical elements or not) but we know that at most $c/2$ elements remain in the pool

The total number of elements left in the pool is at most $\frac{c+b}{2}$, with $b/2$ m-samples. Note that $b > c$ since the number of m-samples is $m = a + 2b$, while the number of nm-samples is $a + 2c$. So the m-samples are still the majority after each iteration. This implies that the last element left must be an m-sample.