EECS 477. HOMEWORK 2 SOLUTIONS

IGOR GUSKOV

1. Search in 2D array (55 points)

Let $a_{i,j}, i = 1 \ldots m, j = 1 \ldots n$ be a two-dimensional array that is ordered in every row and every column so that

- $a_{i,j} \leq a_{i+1,j}$ for $1 \leq i \leq m-1$ and $1 \leq j \leq n$,
- $a_{i,j} \leq a_{i,j+1}$ for $1 \leq i \leq m$ and $1 \leq j \leq n-1$.

You are presented with two algorithms $A_1$ and $A_2$ that search for an element $x$ within the array $a_{ij}$ (see the next page). Assume that $m \leq n$ for convenience.

(a: 20pts) Prove that both algorithms return the location of $x$ within the array or return `not_found` if $a$ does not contain $x$.

(b: 15pts) Let $\phi_k^{[a,x]}(m,n)$ denote the number of $(a[i,j]<x)$ comparisons performed in the algorithm $A_k, k = 1, 2$ for input array $a$ (of the size $m \times n$) and element $x$. Find $\Phi_k(m,n) = \max_{a,x} \phi_k^{[a,x]}(m,n)$ that is the number of comparisons in the worst case for $k = 1, 2$.

(c: 10pts) Taking $\Phi_k(m,n)$ as the measure of performance, which algorithm is better to use when $m = n$ for large values of $n$?

(d: 10pts) Taking $\Phi_k(m,n)$ as the measure of performance, which algorithm is better to use when $m = 5$ for large values of $n$?

$A_1$:

procedure search_A1(array a[1..m,1..n], element x) {
    i = 1;
    j = n;
    while(a[i,j]!=x) {
        if(a[i,j]<x) {
            ++i;
            if(i>m)
                return not_found;
        } else {
            --j;
            if(j<1)
                return not_found;
        }
    }
    return (i,j);
}
A2:

procedure search_A2(array a[1..m,1..n], element x) {
    for i=1..m {
        jmin = 1;
        jmax = n;
        do {
            j = (jmin+jmax)/2;
            if ( a[i,j] < x ) {
                jmin = j+1;
            } else if ( a[i,j] > x ) {
                jmax = j-1;
            } else {
                // a[i,j]==x
                return (i,j);
            }
        } while(jmin<=jmax);
    }
    return not_found;
}

Solution:
(a) Correctness of A1: We first prove that the algorithm always returns within
m+n iterations of the while loop, indeed on every iteration the expression
δ(i,j) = i−j is increased by one, so that starting with δ(1,n) = 1−n and
having δ(m,1) = m−1 as its overall maximum value in the valid index
range, we can only have m−1−(1−n) = m+n valid iterations. As soon
as (i,j) is invalid the loop stops.

We now introduce the “discarded region” D(i,j) = {(k,l) : 1 ≥ k <
i or l < j ≤ n}. The first thing we prove is that the discarded region
never contains x. Indeed, the algorithm starts with empty D(1,n). Every
time i or j is changed, it makes sure that the row or column that is thus
added to the discarded region does not contain x (the orderedness of the
array implies that). It is also easy to see that the algorithm always either
decreases j or increases i hence at some point it arrives at one of the two
degenerate situations i > m or j < 1 for which D(i,j) contains the whole
array. Only in these two cases the algorithm correctly returns not_found.
Thus if not_found is returned, we have proven that the array does not
contain x. On the other hand, if the array does not contain x the while
loop’s condition a[i,j]!=x will always be satisfied, and hence the algorithm
can only return with not_found value. (That it will return within finite
number of steps is shown above.) Therefore, not_found is returned if and
only if x is not in the array. The only other case not considered so far is
that the algorithm finds x on its way and returns. The above considerations
prove that it does it correctly.

Correctness of A2: The second algorithm consists of running binary
search on every row of the matrix, and thus its correctness is trivial.
(b) A1: it was proven above that Φ1(m,n) ≤ m+n and it is easy to see that
the worst case achieves this bound (construct a path that leads from upper
right to lower left corner and fill it with values close to x while everything
above it is much less than \( x \), and everything below that path is much greater than \( x \). Thus \( \Phi_1(m, n) = m + n \).

A2: performs binary search \( m \) times and every binary search takes at most \( \log n \) comparisons. Then \( \Phi_2(m, n) = m \log n. \)

(c) Taking \( m = n \) we get \( \Phi_1(n, n) = 2n \) and \( \Phi_2(n, n) = n \log n. \) For large values of \( n \) we have \( \log n > 2 \) hence the first algorithm is better.

(d) Taking \( m = 5 \) we get \( \Phi_1(5, n) = n + 5 \) and \( \Phi_2(n, n) = 5 \log n. \) For large values of \( n \) we have \( 5 \log n < n + 5 \) hence the second algorithm is better.

2. LIMITS (45 POINTS)

Find the following limits:

(a:15pts)
\[
\lim_{n \to \infty} \frac{2^{n+1} + \log n}{n^3} = \lim_{n \to \infty} \frac{2^{n+1}}{n^3} + \lim_{n \to \infty} \frac{\log n}{n^3} = +\infty + 0 = +\infty.
\]

The two limits above can be evaluated using L'Hopital rule:
\[
\lim_{n \to \infty} \frac{2^{n+1}}{n^3} = \lim_{x \to \infty} \frac{2^{x+1}}{x^3} = \lim_{x \to \infty} \frac{2\ln 2}{3} \frac{2^x}{x^3} = +\infty.
\]

The second limit appeared in the lecture.

(b:15pts) Differentiating four times we get:
\[
\lim_{n \to \infty} \frac{3^{n+1}}{3^n + n^3} = \lim_{x \to \infty} \frac{3^{x+1}}{3^x + x^3} = \lim_{x \to \infty} \frac{3(\ln 3)^3}{3^x (\ln 3)^3} = 3.
\]

(c:15pts) The below expression had a typo – I will compute both cases, either one will count towards the grade.

How it was typed
\[
\lim_{n \to \infty} \sum_{i=0}^{n} 2^{-n+4} = 2^{n+1} * (n + 1) = 0.
\]

How it was supposed to be
\[
\lim_{n \to \infty} \sum_{i=0}^{n} 2^{-n+4} = 2^4 * (1 + 1/2 + 1/4 + \ldots) = 16 * 2 = 32.
\]