Assignment **HW 2**

Solutions by **Lucas Chen**   `lucasch(at)uchicago.edu`

This document contains my solutions to the Gradescope assignment named on the top of this page. Specifically, my solutions to the following problems are included:

I did not forget

- to REFRESH my browser for the latest information about each problem
- to link problems to pages.
  This page is linked to the problems I did not solve.
- to update the items marked *** in the template (my name, email, the Gradescope title of the assignment, the list of problems solved, the `\lhead` statements (left page headers: list of (sub)problems solved on each page)
- to make sure no subproblem solution spills over to the next page (except when this is unavoidable, i.e., when the solution to a subproblem does not fit on a page)
- if a problem takes more than one page, I linked each of those pages to the problem
- I took care not to defeat the mechanisms provided by this template.

With each problem, **I stated my sources and collaborations**.
By submitting this solution *I certify* that *my statement of sources and collaborations is accurate and complete.* I understand that without this certification, my solutions will not be accepted.
In case I am giving a link to a source, *I am also sending this link to the instructor by email.*

5.15 Question.
Let $S(n) > 0$ for every $n \in \mathbb{N}$ and assume $S(n) \leq 3S(n/2) + O(n)$ for those $n \geq 2$ that are powers of 2. From this information we derived that $S(n) = o(n^\alpha)$ where $\alpha = \log_2 3 \approx 1.58$.

Prove that $S(n) = o(n^\alpha)$ does NOT follow from the same information (namely, from the recurrent inequality given and the positivity of $S(n)$.

Sources and collaborations.

Answer.
In order to show the desired statement does not follow we may provide a counterexample of some $S(n)$ which satisfies $S(n) \leq 3S(n/2) + O(n)$ but does not satisfy $S(n) = o(n^\alpha)$. The desired counterexample is provided by $S(n) = n^\alpha$ itself: then $\lim_{n\to\infty} \frac{S(n)}{n^\alpha} = \lim_{n\to\infty} 1 = 1 \neq 0$ and $3\left(\frac{n}{2}\right)^\alpha = n^\alpha \geq n^\alpha$, satisfying the condition for $S(n)$ but not for $o(n^\alpha)$.

5.18 Question.
Let $T(n)$ denote the cost of a Divide-and-Conquer algorithm on inputs of size n, so $T(n) > 0$ for all $n \in \mathbb{N}$. Assume that the function $T(n)$ is monotone non-decreasing, i.e., $(\forall n \in \mathbb{N})(T(n+1) \geq T(n))$. Assume further that for all $n \geq 2$ we have $T(n) \leq 7 \cdot T(\lceil n/2 \rceil) + O(n^2)$. Prove that $T(n) = O(n^\beta)$ for some constant β. Find the smallest value of β for which this conclusion follows from the assumptions.

Sources and collaborations.

Answer.
We will assume first the stronger condition that $T(n) \leq 7 \cdot T(\lceil n/2 \rceil)$, find the smallest possible value of β, and prove it holds for the more general condition. Take $T_1(n) = n^\beta$ for n a power of 2 (we will fill in the rest of $T_1(n)$ later. We find β that satisfies the condition. For some even n we want

$$n^\beta \leq 7 \left(\frac{n}{2}\right)^\beta$$

and we may set $\beta = \log_2(7)$, yielding $n^\beta \leq n^\beta$. For n not a power of 2, we set $T_1(n) = T_1(2^m)$ where 2^m is the least power of 2 greater than n. Then if $2^m < n \leq 2^{m+1}$ we know $2^{m-1} < \frac{\lceil n \rceil}{2} \leq 2^m$ and

$$T_1(2^{m+1}) = 7T_1(2^m) \implies T_1(n) = 7T_1(\lceil n/2 \rceil)$$

We note that since T_1 satisfies our stronger condition it satisfies the original condition, and if we take a $\beta < \log_2(7)$, then

$$\lim_{m\to\infty} \frac{T_1(2^m)}{2^{m\beta}} = \lim_{m\to\infty} 2^{m(\beta - \log_2(7))}$$

which diverges.

Thus no value of β less than $\log_2(7)$ may follow directly from the assumptions (as we cannot rule out $T_1(n)$).

Theorem 1: If $T(n)$ satisfies $T(n) \leq 7 \cdot T(\lceil n/2 \rceil) + O(n^2)$ then $T(n) = O(n^\beta)$ where $\beta = \log_2(7)$.

Proof: Rewrite the inequality as $T(n) \leq 7 \cdot T(\lceil n/2 \rceil) + Cn^2$, which we are able to do since $O(n^2)$ is bounded by some positive constant multiple of n^2. We use the method of reverse inequalities. Take $g(n) = An^\beta - Dn^2$ for even n and $g(n) = A(n+1)^\beta - D(n+1)^2$ for odd n. We aim to find A and D such that

$$g(n-1) = g(n) \geq 7 \cdot g(n/2) + Cn^2 \geq 7 \cdot g(n/2) + C(n-1)^2$$

for even n. We solve for

$$An^\beta - Dn^2 \geq 7(A(n/2)^\beta - D(n/2)^2) + Cn^2$$

Then since $n^\beta = 7(n/2)^\beta$ this becomes

$$\frac{3}{4}Dn^2 \geq Cn^2$$

We pick $D = \frac{4}{3}C \geq 0$ to satisfy the inequality and $A > D$ so $g(n) > 0$ (since $\beta > 2$). We take a base case $Kg(1) \geq T(1)$ (and $K > 1$). Now assuming $Kg(n) \geq T(n)$ we apply both inductive steps:

$$Kg(2n) \geq 7 \cdot Kg(n) + 4KCn^2 \geq 7 \cdot T(n) + 4Cn^2 \geq T(2n)$$

$$Kg(2n-1) \geq 7 \cdot Kg(n) + 4KCn^2 \geq 7 \cdot T(n) + C(2n-1)^2 \geq T(2n-1)$$

and thus $Kg(n) \geq T(n)$ for all $n \geq 1$. We note that

$$f(n) = 2An^\beta \geq A(n+1)\beta - D(n+1)^2 \geq An^\beta - Dn^2$$

meaning $Kf(n) \geq T(n)$ and since $Kf(n), T(n) > 0$ this yields $T(n) = O(2An^\beta) = O(n^\beta)$ and we are done.

5.22 Question.
Let N be a k-bit positive integer. Prove: $k = \lceil \log_2(N+1) \rceil$.

Sources and collaborations.

Answer. If N is k-bit we must have $2^{k-1} \leq N < 2^k$. Since $\log_2$ is an increasing function this yields:

$$2^{k-1} < N + 1 \leq 2^k$$

$$k - 1 < \log_2(N+1) \leq k$$

and thus $\lceil \log_2(N+1) \rceil = k$ as desired.

5.28 Question.
Find two sequences, (a_n) and (b_n), of positive numbers such that $a_n = \Theta(b_n)$ but the limit $\lim_{n\to\infty} \frac{a_n}{b_n}$ does not exist.

Sources and collaborations.

Answer. We take $a_n = (-1)^n + 2$ and $b_n = 1$. Then $\frac{a_n}{b_n} = a_n$ which diverges as n approaches infinity.

5.41 Question.
(Communication Complexity) Alice has access to an n-bit integer $X = \overline{x_0 x_1 \dots x_{n-1}}$, Bob has access to an n-bit integer $Y = \overline{y_0 y_1 \dots y_{n-1}}$. (Initial zeros are permitted; $x_i, y_i \in \{0,1\}$.) Alice and Bob share a k-bit positive integer q such that $X \not\equiv Y \pmod q$. The numbers k and q are known to both of them. The task before Alice and Bob is to find i such that $x_i \neq y_i$. Show that they can accomplish this with no more than$\lceil log_2(n) \rceil \cdot (k+1)$ bits of communication. Describe the protocol in **pseudocode**.

Sources and collaborations.

Answer.

Algorithm:

Binary Search String Comparison

INPUT: X, Y n-bit binary integers, q an integer. We assume $X \not\equiv Y$ (mod q).

procedure StringCompare(X, Y, q, n)
  if $n = 1$ **then return** 0
  end if
  $p := \lfloor n/2 \rfloor$ ▷ Pivot index
  $X_2 := X \mod 2^p$ ▷ Second half of X, p bits
  $Y_2 := Y \mod 2^p$ ▷ Same for Y
  if $X_2 \equiv Y_2 \mod q$ **then** ▷ Communication, max $k+1$ bits.
    return StringCompare($\lfloor X/2^p \rfloor, \lfloor Y/2^p \rfloor, q, n-p$)
  else‘
    return $n - p +$ StringCompare(X_2, Y_2, q, p)
  ▷ (First index of X_2 is $n-p$.)
  end if
end procedure

Analysis: Correctness: procedure ends when X, Y are both one bit long. We verify that StringCompare is never called on X, Y where $X \equiv Y \mod q$: the else statement ensures this trivially. If X_2 and Y_2 are equivalent then

$$X \not\equiv Y \mod q \implies \frac{X - X_2}{2^p} \not\equiv \frac{Y - Y_2}{2^p} \mod q$$

and thus the procedure may only end on 1-bit non-equivalent X and Y, when it will recursively return the indices of these bits.

Complexity: The only line of communication is the if statement, which requires the mod value of X_2 to be communicated (max k bits) and a 1 or 0 communicated back for comparison with Y_2 (1 bit). This line is run a maximum of $\lceil \log_2(n) \rceil$ times, as described in the binary search handout, as the pivot is calculated the same way — thus the algorithm's communication complexity is $\lceil \log_2(n) \rceil \cdot (k+1)$ bits.

6.15 Question.
Given an $n \times n$ array A of zeros and ones, find the maximum size of a contiguous square of all ones. (You do not need to locate such a largest all-ones square, just determine its size.) Solve this problem in *linear time*. "Linear time" means the number of steps must be O(size of the input). In the present problem, the size of the input is $O(n^2)$. Manipulating integers between 0 and n counts as one step; such manipulation includes copying, incrementing, addition ad subtraction, looking up an entry in an $n \times n$ array. Describe your solution in **pseudocode**.

Sources and collaborations. Initially iterated back over the array M to find the maximum size. Isaac Chang suggested recording this during each step using a variable, which uses almost the same number of integer manipulations (still equivalent to $O(n^2)$) but removes the need to write out two entire nested loops.

Answer.

Algorithm:

Maximum Square Size

INPUT: A: list[list[int]] of size $n \times n$, n positive integer

procedure MaxSquare(A, n)
    $M :=$ ARRAY[n] of ARRAY[n] of int
    ▷ Entry is largest square in A with bottom left corner at indices.
    $h := 0$ ▷ Variable holding max size
    for $0 \leq i < n$ **do**
        for $0 \leq j < n$ **do**
            if $i, j > 0$ **then**
                $M[i,j] = A[i,j] \cdot$ $(\min(M[i-1,j-1], M[i,j-1], M[i-1,j])$
            else
                $M[i,j] = A[i,j]$
            end if
            $h = \max(h, M[i,j]$
        end for
    end for
    return h
end procedure

Analysis

Correctness: At each index $[i,j]$ the value of h is the maximum square size in the rectangle with top left at $[0,0]$ and bottom right at $[i,j]$. The value of $M[i,j]$ is the size of the largest square exactly with its bottom right at $[i,j]$. For a square of size a to have a bottom right at i,j, there must be squares of at least size $a-1$ with bottom rights in the indices directly above, to the left, and to the upper right diagonal of $[i,j]$, so we check these along with the value of $M[i,j]$. The maximum size square's bottom right corner will inevitably be iterated over so we will inevitably achieve the maximum size in h.

Complexity: We count the number of integer manipulations:

- Line 3: 1
- Line 4: $n-1$
- Line 5, run n times: $n-1$
- Line 6, run n^2 times: 2
- Lines 7-9, run n^2 times: a maximum of 5
- Line 11, run n^2 times: 3

This yields $11n^2 = O(n^2)$ total integer manipulations and we are done.