CS3230 Design and Analysis of Algorithms	September 14, 2019
Midterm 1	
Name: Divesh Aggarwal	

Problem 1-1 (Palindrome)

Recall that a palindrome is a string that is the same as its reverse. Any string can be decomposed into a sequence of palindromes. For example, the string XYXYXY can be broken into palindromes in the following ways(and some others): XYX + YXY, XYXYX + Y, X + YXYXY, X + YXY + X + Y, X + Y + X + Y + X + Y. Your task in this problem is to design an efficient algorithm to find the smallest number of palindromes that make up a given input string $A[1 \dots n]$. For example, given the input string XYYXYXY, your algorithm would return the integer 2. You do NOT need to prove correctness or analyze the running time for any of the sub-tasks.

(a) Define a boolean array $B[1 \dots n][1 \dots n]$ as follows: $B[i][j] = 1$ if and only if either $i \geq j$ (trivial case), or $i < j$ and $A[i \dots j]$ is a palindrome. Give a recurrence relation for $B[i][j]$.

Solution:

$$B[i][j] = \begin{cases} 1 & \text{, if } i \geq j; \\ B[i+1][j-1] & \text{, if } i < j \text{ and } A[i] = A[j] \\ 0 & \text{, otherwise} \end{cases}$$

□

(b) Assume that you already computed the boolean array $B[1 \dots n][1 \dots n]$ using the recursion above. Let $S[0 \dots n]$ be an array such that $S[0] = 0$ and, for $i \geq 1$, $S[i]$ is the smallest number of palindromes that make up the string $A[1 \dots i]$. Give a recurrence relation to compute $S[i]$ as a function of $S[0 \dots i-1]$ and the boolean array $B[1 \dots n][1 \dots n]$.

$$S[i] = \begin{cases} \_\_\_\_\_, & \text{if } i = 0; \\ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, & \text{if } i > 0. \end{cases}$$

Solution:

$$S[i] = \begin{cases} 0, & \text{if } i = 0; \\ 1 + \min_{j<i \,:\, B[j+1][i]=1} S[j], & \text{if } i > 0. \end{cases}$$

□

(c) Given array B as (already computed) input, use part (b) to give pseudocode for the *bottom-up* dynamic programming $O(n^2)$ algorithm to compute $S[0 \dots n]$, and output $S[n]$.

Solution: Straightforward from the recurrence relation in part (b). □

Problem 1-2 (Ski and Skier)

Consider the following problem. The input consists of n skiers with heights $p_1, \ldots, p_n$, and n skies with heights $s_1, \ldots, s_n$. The problem is to assign each skier a ski to minimize the average difference between the height of a skier and his/her assigned ski. That is, if the i-th skier is given the $\alpha(i)$-th ski, then you want to minimize:

$$\frac{1}{n}\sum_{i=1}^{n}\left|p_i - s_{\alpha(i)}\right| .$$

We suggest two greedy algorithms to solve this problem. Only one of these algorithms is correct.

Algorithm A: Find the skier and ski whose absolute height difference is smallest. Assign this skier this ski. Repeat the process until every skier is assigned a ski.

Algorithm B: Give the shortest skier the shortest ski, give the second shortest skier the second shortest ski, give the third shortest skier the third shortest ski, etc.

(a) Which of the Algorithms A or B is *incorrect*? Also, give a counter-example for this algorithm for $n = 2$ formatted as in below.

Counter-example:

$$p_1 = \_\_\_\_\_, \quad p_2 = \_\_\_\_\_, \quad s_1 = \_\_\_\_\_, \quad s_2 = \_\_\_\_\_$$

$$\text{Greedy average difference } = \frac{1}{2}\left(\_\_\_\_ + \_\_\_\_\right) = \_\_\_\_$$

$$\text{Optimal average difference } = \frac{1}{2}\left(\_\_\_\_ + \_\_\_\_\right) = \_\_\_\_$$

Solution: Algorithm A is incorrect. Any example with $p_1 \ll s_1 < p_2 \ll s_2$ works.

Counter-example:

$$p_1 = 1, \quad p_2 = 7, \quad s_1 = 5, \quad s_2 = 11$$

$$\text{Greedy average difference } = \frac{1}{2}(2 + 10) = 6$$

$$\text{Optimal average difference } = \frac{1}{2}(4 + 4) = 4$$

□

(b) Which of the Algorithms A or B is *correct*? Convince yourself that the algorithm is correct for $n = 2$ (you do not need to prove this), and use this and the *local swap* argument to prove the correctness of the algorithm for all n.

Prove that it is correct.

Solution: Algorithm B is correct. We assume the base case that the algorithm is correct for $n = 2$. Now, assume that there is an optimal solution that is not found by the greedy algorithm. Assume that

$$p_1 \leq p_2 \leq \cdots \leq p_n \ .$$

Now any solution to the problem corresponds to finding a permutation $\alpha : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ of the skis that minimizes the given sum. For any permutation α, we let $I(\pi)$ be the number of inversions, i.e., the number of pairs (i, j) such that $i < j$ and $s_{\alpha(i)} > s_{\alpha(j)}$.

Assume that the greedy solution does not give the optimal solution. Then, we consider the optimal solution α^* that minimizes the number of inversions. Consider any pair $(i, i+1)$ such that $\alpha^*(i) > \alpha^*(i+1)$. We define a new permutation β such that $\beta(i) = \alpha^*(i+1)$, $\beta(i+1) = \alpha^*(i)$, and $\beta(j) = \alpha^*(j)$, otherwise. Then, we have the following:

- The permutation β has exactly 1 inversion less than α, i.e., $I(\beta) = I(\alpha^*) - 1$.
- By the optimality of the algorithm for $n = 2$,

$$\frac{1}{n} \sum_{i=1}^{n} |p_i - s_{\beta(i)}| \leq \frac{1}{n} \sum_{i=1}^{n} |p_i - s_{\alpha^*(i)}| \ .$$

This contradicts the fact that that α^* is an optimal solution that minimizes the number of inversions. □

Problem 1-3 (Increment or Divide?)

Consider the following process. At all times you have a single positive integer x, which is initially equal to 1. In each step, you can either increment x or double x. Your goal is to produce a target value n. For example, you can produce the integer 10 in four steps as follows:

$$1 \xrightarrow{\times 2} 2 \xrightarrow{\times 2} 4 \xrightarrow{+1} 5 \xrightarrow{\times 2} 10$$

Obviously you can produce any integer n using exactly $n - 1$ increments. But for almost all values of n, this is horribly inefficient. Describe an $O(\log n)$ time algorithm to compute the minimum number of steps required to produce any given integer n. You should prove correctness of your algorithm.

(**Hint**: Let the binary representation of n (with the leading bit 1) be

$$n = 1a_1a_2 \ldots a_\ell \ .$$

You might want to express the minimum number of steps output by your algorithm in terms of ℓ and the number of 1s in $a_1a_2 \ldots a_\ell$.)

Solution: Let the binary representation of n (with the leading bit 1) be

$$n = 1a_1a_2 \ldots a_\ell \ .$$

We claim that the minimum number of steps is $\ell + k$, where k is the number of 1s in $a_1a_2 \ldots a_\ell$. We prove this by induction on n.

Clearly, for $n = 1$, we need 0 steps, for $n = 2$, we need 1 step, and for $n = 3$, we need 3 steps, and so the claim is correct for $n \leq 3$. Now, we assume that it is correct for $n = 1, 2, 3, \ldots, m-1$, where $m \geq 4$. We now show that it is correct for $n = m$.

Let $m = 1a_1a_2\ldots a_\ell$.

If $a_\ell = 1$, then the last step must be an increment, and the result follows by induction hypothesis.

If $a_\ell = 0$, then the last step could be increment or doubling. If the last step is doubling, then in the previous step, we have $m/2 = 1a_1a_2\ldots a_{\ell-1}$, and by the induction hypothesis the minimum number of steps required in this case is $\ell + k$, where k is the number of 1s in $a_1a_2\ldots a_\ell$.

If the last step is an increment, then the number of steps to obtain m is at least 1 more than the minimum number of steps to obtain $m-1$. We show in this case, that we obtain a suboptimal solution. If the binary representation of m is of the form $100\ldots0$, then $m - 1 = 111\ldots1$ (ℓ 1s), and hence the number of steps required is at least $\ell - 1 + \ell - 1 + 1 = 2\ell - 1$, which is larger than ℓ. Else, m is of the form $1a_1\ldots a_{i-1}1000\ldots0$, and in this case, the number of steps required to obtain $m-1$ is $\ell + k + \ell - i - 1$, and hence the number of steps to obtain m is $\ell + k + \ell - i$, which is suboptimal.

Alternative argument: If the last step is an increment when $a_\ell = 0$, i.e., n is even, then look for the last doubling step, which occurs from $m/2 \to m$ (where m is even), i.e.,

$$m/2 \xrightarrow{\times 2} m \xrightarrow{+1} m+1 \xrightarrow{+1} \cdots \xrightarrow{+1} n\ ,$$

Then the total number of steps after $m/2$ are $n - m + 1$. An alternative path from $m/2$ is

$$m/2 \xrightarrow{+1} m/2+1 \xrightarrow{+1} \cdots \xrightarrow{+1} n/2 \xrightarrow{\times 2} n\ ,$$

which requires a total of $\frac{n-m}{2} + 1$ steps, contradicting the optimality of the solution. □

Problem 1-4 (Recursive Squaring)

Describe a recursive algorithm that squares any n-digit number in $O(n^{\log_3 5})$ time, by reducing to squaring only *five* $(n/3 + O(1))$-digit numbers.

(**Hint**: What is $(a+b+c)^2 + (a-b+c)^2$? Consider using a^2, c^2, $(a+b+c)^2$, $(a-b+c)^2$ and $(\cdots)^2$.)

For partial credit, describe a recursive algorithm that squares any n-digit number in $O(n^{\log_3 6})$ time, by reducing to squaring only *six* $(n/3 + O(1))$-digit numbers.

Solution: Let x be an n-digit number. We write $x = a \cdot 10^{2m} + b \cdot 10^m + c$, where $m = \lceil n/3 \rceil$, and $a, b, c < 10^m$.

Then

$$x^2 = a^2 \cdot 10^{4m} + 2ab \cdot 10^{3m} + (b^2 + 2ac) \cdot 10^{2m} + 2bc \cdot 10^m + c^2\ .$$

Now, it is easy to see that if we have a^2, b^2, c^2, $(a+b)^2$, $(b+c)^2$, $(c+a)^2$, then we get each of $2ab$, $2bc$, and $2ca$, and so we can easily reduce the problem to squaring *six* $(n/3 + O(1))$-digit numbers.

For reducing to squaring five numbers, we need to work a little more. Using the hint, consider four of the squares to be a^2, c^2, $(a+b+c)^2$, $(a-b+c)^2$. Then,

$$(a+b+c)^2 + (a-b+c)^2 - 2a^2 - 2c^2 = 2(b^2 + 2ac)\ ,$$

and

$$(a+b+c)^2 - a^2 - c^2 - (b^2+2ac) = 2ab + 2bc \ ,$$

which means that using the given four squares, we can compute a^2, c^2, b^2+2ac, and $ab+bc$. We want another equation in ab and bc in order to get their individual values.

Let the fifth square be $(\alpha a + \beta b + \gamma c)^2$. We have

$$(\alpha a + \beta b + \gamma c)^2 - \alpha^2 a^2 - \gamma^2 c^2 - \beta^2(b^2 - 2ac) = 2(\alpha\gamma - \beta^2)ac + 2\alpha\beta ab + 2\beta\gamma bc \ .$$

So, to get another different equation in ab and bc, we need to get rid of the ac term, which happens if $\alpha\gamma = \beta^2$, and we want $\alpha\beta \neq \gamma\beta$, which implies $\alpha \neq \gamma$.

So any α, β, γ satisfying the above will work. In particular, we can choose the fifth square to be $(a+2b+4c)^2$, and then

$$(a+2b+4c)^2 - a^2 - 16c^2 - 4(b^2+2ac) = 4ab + 16bc \ ,$$

and thus we can compute the value of ab and bc. The number of digits in $a+2b+4c$ is at most $\lceil n/3 \rceil + 1 \leq n/3 + 2$, and we obtain the following recurrence.

$$T(n) \leq 5T(n/3+2) + O(n) \ ,$$

where $O(n)$ is the time taken for the required additions and subtractions. Let $S(n) = T(n+3)$. Then, we have that

$$S(n) = T(n+3) \leq 5T((n+3)/3+2) + O(n) = 5T(n/3+3) + O(n) = 5S(n/3) + O(n) \ .$$

Using the recursion tree method, or the masters method gives the desired running time for the algorithm. $\square$