Sample Solution for Problem Set 2

Data Structures and Algorithms, Fall 2022

October 8, 2022

Contents

1 Problem 1

Check https://pastebin.com/95pkSH2n for the implementation of model solution.

Analysis

(a)

For the time complexity, $T(n) = 2T(n/2) + cn = \Theta(n \lg n)$.
For the space complexity, we use $O(n)$ space, $temp[1...n]$ to store elements during the mergence, and $O(logn)$ space used by recursive stack, So we use $O(n)$ space beside input.

(b)

For the time complexity, $T(n) = 2T(n/2) + cn = \Theta(n \lg n)$.
For the space complexity, We use $O(1)$ space in auxiliary pointer variables, and $O(logn)$ space used by recursive stack, So we use $O(logn)$ space beside input.

2 Problem 2

Check https://pastebin.com/aff4Tx3r for the model solution.

Correctness

Without loss of generality, we assume that there exists no parentheses in the expression.

We prove it by induction. Induction hypothesis: the algorithm will turn infix expression to sufix expression correctly when the length of input is at most n.

When $n = 1$, the input is a single digit, the output is correct.

When $n > 1$, denote the last operator with the smallest priority as $A[x]$. (When there are operators $x+$, that means $A[x]$ is the last $+$ in the expression). Then $A[x]$ should be in the last place in postfix expression. That is true in our algorithm, since when we add $A[x]$ to the stack, it will pop all elements in stack out (because it has the smallest priority), so it will fall into the bottom of the stack. And nothing will pop it out until the fifth line of our algorithm, which will add $A[x]$ to the last place in the postfix expression.

Now consider $A[0...x-1]$ and $A[x+1...n]$ (might be empty string). They both have length smaller than n. Before $A[x]$ is push into the stack, the procedure can be seen as running our algorithm in $A[0...x-1]$, since $A[x]$ will pop all elements out, which is just like what we do in line 5. Thus, according to induction hypothesis, $A[0...x-1]$ will be turned to postfix expression correctly. $A[x+1...n]$ can also be seen as running our algorithm independently, since all elements in $A[0....x-1]$ is not in the stack, and $A[x]$ is in the bottom of the stack while no one popping it out. Thus, the final string is $postfix(A[0...x-1]) + postfix(A[x+1...n]) + A[x]$, which is correct.

Complexity

Each operator will be popped and pushed at most twice, and each digit will be looped at most once. The complexity is $O(n)$.

3 Problem 3

(a)

The missing detail: $PWR2BIN(n)$. Suppose the running time of the algorithm is $T(n)$. Then we have

$$T(n) = \begin{cases} O(1) & , \quad n = 1 \\ T(n/2) + O(n^{\log_2 3}) & , \quad o.w. \end{cases}$$

By Master theorem, $a = 1$, $b = 2$, $f(n) = \Omega(n^{\log_2 3})$, we have $T(n) = O(n^{\log_2 3})$.

(b)

The missing detail: $FASTMULTIPLY(DEC2BIN(x_L), POW2BIN(n/2)) + DEC2BIN(x_R)$.

Suppose the running time of the algorithm is $T(n)$. Then we have

$$T(n) = \begin{cases} O(1) & , \quad n = 1 \\ 2 \cdot T(n/2) + O(n^{\log_2 3}) & , \quad o.w. \end{cases}$$

By Master theorem, $a = 2$, $b = 2$, $f(n) = \Omega(n^{\log_2 3})$, we have $T(n) = O(n^{\log_2 3})$.

4 Problem 4

(a)

Let x be an n-digits number. Take $m = \lceil \frac{n}{2} \rceil$, $x = a \times 10^m + b$ ($b < 10^m$), then

$$\begin{aligned} x^2 &= a^2 \times 10^{2m} + b^2 + 2ab \times 10^m \\ &= a^2 \times 10^{2m} + b^2 + \left(a^2 + b^2 - (a-b)^2\right) \times 10^m \end{aligned}$$

We only need to square three $\frac{n}{2}$-bit squares $a^2, b^2, (a-b)^2$, with additional addition, multiplication in $O(n)$. So time complexity

$$T(n) = 3T(\frac{n}{2}) + O(n) = O(n^{\lg 3})$$

(b)

Professor F. Lake is wrong.

- On the one hand, if we have any square algorithm, we can compute multiplication in the following way

  $$xy = \frac{(x+y)^2 - (x-y)^2}{4}$$

  notice multiplication needs $\Omega(n)$ time, so cost of addition and division here doesn't count. Therefore, we conclude that square cannot be asymptotically faster than multiplication.(or **square is harder than multiplication.**)

- On the other hand, if we have any multiplication algorithm, we can directly use is to calculate $x \times x = x^2$. Therefore, **multiplication is harder than square.**

- In summary, square is as hard as multiplication.

Remark

To say problem A is harder than problem B, you need to show that problem B can be **reduced to** problem A efficiently. Your argument **should not** focus on a certain algorithm.

5 Problem 5

(a)

Consider the following procedure:

- For each $1 \leq i \leq \lfloor \frac{n}{2} \rfloor$, test chip $2i$ and $2i+1$, and add chip $2i$ to set C if both reports good.
- If $n \equiv 1 \mod 4$, add chip n to set C.

It can be verified that

- In set C, the number of good chips is strictly larger than bad chips;
- $|C| \leq \lceil \frac{n}{2} \rceil$.

(b)

Recursively apply procedure in (a). Hence, $T(n) = T(n/2) + O(n)$, which implies $T(n) = O(n)$.

(c)

We assume that there are b bad chips $B_1, B_2, \ldots, B_b$, and g good chips $G_1, G_2, \ldots, G_g$. It is impossible to distinguish B_1 and G_1 if

- For each $1 \leq i \leq g$, chip B_i will report chip X good if and only if $X = B_j$ for some $1 \leq j \leq g$;
- For each $g+1 \leq i \leq b$, chip B_i will report chip X bad in any circumstances.