A2

1. Array-based Data Structures

1.1 Implement two Stacks in one Array

- *From 10.1-3 of CLRS*

Explain how to implement two stacks in one array $A[1:n]$ in such a way that neither stack overflows unless the total number of elements in both stacks together is n .
The `PUSH` and `POP` operations should run in $O(1)$ time.

1.1 Solution code/TwoStackinOneArray.cpp

2. Linked List

2.1 Reverse a Linked List

- *From 10.2-5 of CLRS*

Give a $\Theta(n)$-time non-recursive procedure that reverses a singly linked list of elements.
The procedure should use no more than constant storage beyond that needed for the list itself.

2.1 Solution code/Reverse_Linked_List.cpp

3. Divide-and-Conquer & Recursion

3.1 Multiply Matrices with Squaring Algorithm

- *From 4.2-6 of CLRS*

Suppose that you have a $\Theta(n^\alpha)$-time algorithm for squaring $n \times n$ matrices, where is $\alpha \geq 2$ a constant. Show how to use that algorithm to multiply two different $n \times n$ matrices in $\Theta(n^\alpha)$ time.

3.1 Solution

假设 A,B 为两个$n \times n$ 的矩阵,我们需要得到 AB

$$\text{假设 } C = \begin{bmatrix} O & A \\ B & O \end{bmatrix} \tag{1}$$

则 C 为一个 $2n \times 2n$ 的矩阵, 且:

$$C^2 = \begin{bmatrix} AB & O \\ O & BA \end{bmatrix} \tag{2}$$

由于 我们对于$n \times n$矩阵有 $\Theta(n^\alpha)$ 时间复杂度的平方算法($\alpha \geq 2$ 为一个常数),

那么 得到 C^2 的时间复杂度为 $\Theta((2n)^\alpha) = \Theta(2^\alpha \times n^\alpha) = \Theta(n^\alpha)$

又因为 AB 在 C^2 的左上角,可以由C^2直接得到,故而 $A \times B$ 的时间复杂度为 $\Theta(n^\alpha)$.

3.2 Sort the Pancakes

- *From 1.9 of Algorithms by Jeff Erickson*

Suppose you are given a stack of n pancakes of different sizes. You want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation you can perform is a *flip*——insert a spatula under the top k pancakes, for some integer k between 1 and n, and flip them all over.

1. Describe an algorithm to sort an arbitrary stack of n pancakes using $O(n)$ flips. Exactly how many flips does
   your algorithm perform in the worst case?[1]
2. Now suppose one side of each pancake is burned. Describe an algorithm to sort an arbitrary stack of n pancakes, so that the burned side of every pancake is facing down, using $O(n)$ flips. Exactly how many flips does your algorithm perform in the worst case?

3.2.1. Solution code/Sort_Pancakes.cpp

```
// 核心代码:
    int SortPancakes() {
        for (int _size = pancakes.size(); _size > 0; _size--) {
            if (pancakes[_size - 1] == _size) continue; // 已经在所在位置了
            flip(search(_size)); // 翻转到头
            flip(_size - 1); // 翻转到尾
        }
        return cnt;
    }
```

最坏的情况是每一次都要翻转到头 再翻转到尾巴，也就是要翻转 $2 \times n = 2n$次
而每一次翻转 复杂度为 $\_size$, 故而总时间复杂度为 $O(n^2)$

3.2.2. Solution code/Sort_Pancakes2.cpp

```
// 核心代码:
    int SortPancakes() {
        for (int _size = pancakes.size(); _size > 1; _size--) {
            if (pancakes[_size - 1].size == _size && pancakes[_size - 1].is_facedown == true) {
                continue; // 已经在所在位置了 且 头朝下
            }
            flip(search(_size)); // 翻转到头

            // 翻转到尾 check一下是否是 burned朝下,如果朝下 翻转到上
            if (pancakes[0].is_facedown) {
                cnt++;
                pancakes[0].flip();
            }
            flip(_size - 1);
        }
        if (!pancakes[0].is_facedown) {
            cnt++;
            pancakes[0].flip();
        }
        return cnt;
    }
```

最坏的情况是每一个煎饼都要翻转到头,然后翻面, 再翻转到尾巴，也就是要翻转 $3 \times n = 3n$次
而每一次翻转 复杂度为 $\_size$, 故而总时间复杂度为 $O(n^2)$

3.3 Calculate the Inversions

- *From 1.13 of Algorithms by Jeff Erickson*

An **inversion** in an array $A[1...n]$ is a pair of indices (i, j) such that $i < j$ and $A[i] > A[j]$.
The number of inversions in an n-element array is between 0 (if the array is sorted) and $\binom{n}{2}$ (if the array is sorted backward).
Describe and analyze an algorithm to count the number of inversions in an -element array in $O(nlogn)$ time.[Hint: Modify mergesort.]

Solution code/count_ReversedPairs.cpp

by Mojo 毛九弢 221900175 2023/09/30

1. The exact worst-case optimal number of flips required to sort pancakes (either burned or unburned) is an long-standing open problem; just do the best you can. ↩