CS2010 Bonus PS - Pancake Sorting

Released: Saturday, 22 September 2012
Due: Saturday, 29 September 2012, 8am

Collaboration Policy. You are encouraged to work with other students or teaching staffs (inside or outside this module) on solving this problem set. However, you **must** write the Java code **by yourself**. In addition, when you write your Java code, you **must** list the names of every collaborator, that is, every other person that you talked to about the problem (even if you only discussed it briefly). This list may include certain posts from fellow students in CS2010 IVLE discussion forum. Any deviation from this policy will be considered cheating, and will be punished severely, including referral to the NUS Board of Discipline. It is not worth it to cheat just to get 15% when you will lose out in the other 85%.

R-option. *The entire bonus PS* is mainly for those who take the R-option. Beware that this is a (very) hard PS. PS: Subtask 1-5 should be doable with your current CS2010 knowledge.

The Story. This is a side quest and thus not related to babies...

You are given a stack of **N** pancakes, where $1 \leq N \leq 10$. The pancake at the bottom and at the top of the stack has **index 0** and **index N-1**, respectively. The size of a pancake is its diameter which is a positive integer. All pancakes in the stack have different sizes. A stack `A` of **N = 5** pancakes with size 3, 8, 7, 6, and 10 can be visualized as:

```
4 (top)       10
3             6
2             7
1             8
0 (bottom)    3
----------------------
index         A
```

We want to sort the stack in **descending order**, that is, the largest pancake is at the bottom and the smallest pancake is at the top. To make the problem more real-life like, sorting a stack of pancakes can only be done by a sequence of pancake 'flips', denoted by **flip(i)**. The **flip(i)** operation inserts a spatula into the stack, lifts up pancakes from index **i** to index **N - 1** and flips them. As a result, the position of the pancakes from index **i** to index **N - 1** are reversed.

For example, stack `A` can be transformed to stack `B` via **flip(0)**, i.e. inserting a spatula and flipping the pancakes between index 0 and 4. Stack `B` can be transformed to stack `C` via **flip(3)**. Stack `C` can be transformed to stack `D` via **flip(1)**. And so on. Our target is to make the stack sorted in **descending order**, i.e. we want the stack to be like stack E.

```
4 (top)       10 <--   3  <--   8  <--   6                3
3             6        8  <--   3        7        ...     6
2             7        7        7        3                7
1             8        6        6  <--   8                8
0 (bottom)    3  <--   10       10       10               10
-------------------------------------------------------------
index         A        B        C        D        ...     E
```

The Actual Problem. Given the starting configuration of **N** pancakes, your task is to compute the **minimum** number of flips required to sort them.

The skeleton program `PancakeSorting.java` is already written for you, you just need to implement one (or more) method(s)/function(s):

- `int Sort(int N, int[] pancake)`
  Report the minimum number of flips required to sort these N pancakes.
- If needed, you can write additional helper methods/functions to simplify your code.

Trivia: Bill Gates (Microsoft founder, former CEO, and current chairman) wrote only one research paper so far, and it is about this pancake sorting (Gates, W. and Papadimitriou, C. **"Bounds for Sorting by Prefix Reversal."** Discrete Mathematics, 27, 47-57, 1979).

Subtask 1 (5 points). It is guaranteed that all starting pancake configurations can be sorted with **at most 1 flip**. Moreover, pancake diameters will only range from **[1 .. N]** and **N** $\leq$ **8**.

Sample Input for Subtask 1:

```
2
4   4 3 2 1
8   8 7 6 5 4 1 2 3
```

Sample Output for Subtask 1:

```
0
1
```

Explanation for the Sample Output for Subtask 1:

The first input stack is already sorted in descending order, so 0 flip is needed.
The second input stack can be sorted in descending order by calling **flip(5)**: 1 flip.

Subtask 2 (5 points). Same as Subtask 1 criteria above, but now it is guaranteed that all starting pancake configurations can be sorted with **at most 2 flips**.

Sample Input for Subtask 2:

```
3
4   4 3 2 1
8   8 7 6 5 4 1 2 3
5   5 1 2 4 3
```

Sample Output for Subtask 2:

```
0
1
2
```

Explanation for the Sample Output for Subtask 2:

The first and the second stacks are exactly the same as Subtask 1 above.
The third stack can be sorted by calling **flip(3)** then **flip(1)**: 2 flips.

Subtask 3 (5 points). Same as Subtask 2 criteria above, but now the largest pancake diameter, can be as big as 1000000 (one million).

Sample Input for Subtask 3:

```
2
5   555555 111111 222222 444444 333333
8   1000000 999999 999998 999997 999996 999995 999994 999993
```

Sample Output for Subtask 3:

```
2
0
```

Explanation for the Sample Output for Subtask 3:

The first stack can be sorted by calling **flip(3)** then **flip(1)**: 2 flips.
The second stack is already sorted in descending order, so 0 flip is needed.

Subtask 4 (5 points). Same as Subtask 3 criteria above, but this time the minimum number of flips in this input instance is **not constrained**. You have to report **minimum** number of flips needed to sort the pancakes in descending order. $\mathbf{1} \leq \mathbf{TC} \leq \mathbf{200}$ and $\mathbf{N} \leq \mathbf{8}$.

Subtask 5 (5 points). Same as Subtask 4 criteria above, but $\mathbf{1} \leq \mathbf{TC} \leq \mathbf{6000}$ and $\mathbf{N} \leq \mathbf{8}$. Therefore, your program must be **efficient**.

Subtask 6 (15 points). Same as Subtask 5 criteria above, but $\mathbf{1} \leq \mathbf{TC} \leq \mathbf{6000}$ and $\mathbf{N} \leq \underline{\mathbf{10}}$. Therefore, your program must be **super efficient**.

Sample Input for Subtask 4, 5, and 6:

```
4
5   555555 111111 222222 444444 333333
8   1000000 999999 999998 999997 999996 999995 999994 999993
5   3 8 7 6 10
10   8 1 9 2 0 5 7 3 6 4
```

Sample Output for Subtask 4, 5, and 6:

```
2
0
4
11
```

Explanation for the Sample Output for Subtask 4, 5, and 6:

The first and the second stacks are exactly the same as instance 3 above.
The third stack (the sample stack shown in the first page of this problem description) can be sorted using at minimum 4 flips, i.e.
Solution 1: **flip(0)**, **flip(1)**, **flip(2)**, **flip(1)**: 4 flips.
Solution 2: **flip(1)**, **flip(2)**, **flip(1)**, **flip(0)**: also 4 flips.
The fourth stack with $\mathbf{N = 10}$ is given for you to test the runtime speed of your solution.

Notes: As this is a bonus PS, no test data is given other than the sample input/output.
You are allowed to check your program's output with your friend's.
You are encouraged to generate and post additional test data in IVLE discussion forum.
Student's code will be killed after 2 times Steven's reference speed on TA's machine.