HW 5

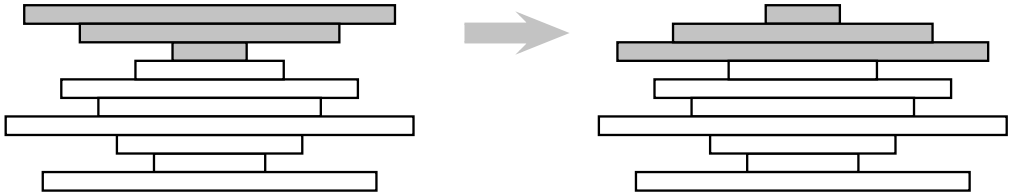
Due on **Monday, March 2, 2020 at 11pm**

CS/ECE 374 B: Algorithms & Models of Computation, Spring 2020 Version: **1.2**

Submission instructions as in previous homeworks.

13 (100 PTS.) Pancake Algorithm.

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



13.A. Describe an algorithm to sort an arbitrary stack of n pancakes and give a bound on the number of flips that the algorithm makes. Assume that the pancake information is given to you in the form of an n element array A. $A[i]$ is a number between 1 and n and $A[i] = j$ means that the j'th smallest pancake is in position i from the bottom; in other words $A[1]$ is the size of the bottom most pancake (relative to the others) and $A[n]$ is the size of the top pancake. Assume you have the operation Flip(k) which will flip the top k pancakes. Note that you are only interested in minimizing the number of flips.

13.B. Suppose one side of each pancake is burned. Describe an algorithm that sorts the pancakes with the additional condition that the burned side of each pancake is on the bottom. Again, give a bound on the number of flips. In addition to A, assume that you have an array B that gives information on which side of the pancakes are burned; $B[i] = 0$ means that the bottom side of the pancake at the i'th position is burned and $B[i] = 1$ means the top side is burned. For simplicity, assume that whenever Flip(k) is done on A, the array B is automatically updated to reflect the information on the current pancakes in A.

14 (100 PTS.) Fetching Bit by Bit.

Consider an array $A[0 \,..\, n-1]$ with n distinct elements. Each element is an ℓ bit string representing a natural number between 0 and $2^\ell - 1$ for some $\ell > 0$. The only way to access any element of A is to use the function **FetchBit**(i, j) that returns the jth bit of $A[i]$ in $O(1)$ time.

14.A. (20 PTS.) Suppose $n = 2^\ell - 1$, i.e. exactly one of the ℓ-bit strings does not appear in A. Describe an algorithm to find the missing bit string in A using $\Theta(n \log n)$ calls to **FetchBit** without converting any of the strings to natural numbers.

14.B. (40 PTS.) Suppose $n = 2^\ell - 1$ as before. Describe an algorithm to find the missing bit string in A using only $O(n)$ calls to **FetchBit**.

14.C. (40 PTS.) Suppose $n = 2^\ell - k$, i.e. exactly k of the $l-$bit strings do not appear in A. Describe an algorithm to find the k missing bit strings in A using only $O(n \log k)$ calls to **FetchBit**.

15 (100 PTS.) Solving Recurrence Relations.

Solve the following recurrence relations. For parts (a) and (b), give an exact solution. For parts (c) and (d), give an asymptotic one. In both cases, justify your solution.

15.A. (20 PTS.) $A(n) = A(n-1) + 2n + 1; A(0) = 0$

15.B. (20 PTS.) $B(n) = B(n-1) + n(n-1) - 1; B(0) = 0$

15.C. (20 PTS.) $C(n) = C(n/2) + C(n/3) + C(n/6) + n$

15.D. (20 PTS.) $D(n) = D(n/2) + D(n/3) + D(n/6) + n^2$