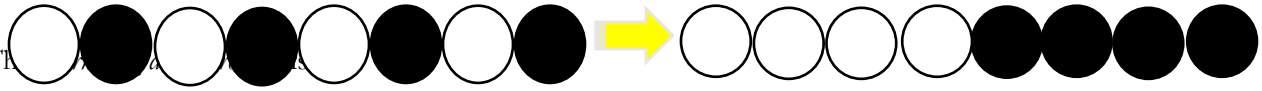
The Alternating Disk Problem

The problem below is slightly changed from the one presented in Levitin's textbook as Ex. 14 on page 103:

> You have a row of $2n$ disks of two colors, n light and n dark. They alternate: light, dark, light, dark, and so on. You want to get all the light disks to the left-hand end, and all the dark disks to the right-hand end. The only moves you are allowed to make are those that interchange the positions of two adjacent disks (i.e. they touch each other). No other moves are allowed. Design an algorithm for solving this puzzle and determine the number of moves it takes.



Input: a positive integer `n` and a list of `2n` disks of alternating colors light-dark, starting with light
Output: a list of `2n` disks, the first `n` disks are light, the next `n` disks are dark, and an integer `m` representing the number of swaps to move the dark ones after the light ones.

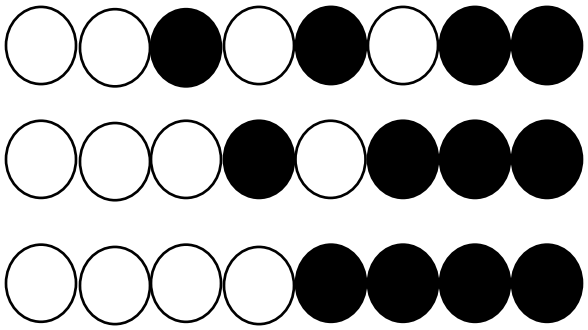
There are two algorithms, presented below, that solve this problem in $O(n^2)$ time. Some improvement can be obtained by not going all the way to the left or to the right, since some disks at the ends are already in the correct position. You need to translate the descriptions of the two algorithms into clear pseudocode. You are allowed to do the improvements as long as it does not change the description of the algorithm.

The first algorithm starts with the leftmost disk and proceeds to the right until it reaches the rightmost disk: compares every two adjacent disks and swaps them only if necessary. Now we have two light disks at the left-hand end and two dark disks at the right-hand end. Once it reaches the right-hand end, it goes back to the leftmost disk and proceeds to the right, doing the swaps as necessary. It repeats `n` times.

The second algorithm proceeds like a lawnmower: starts with the leftmost disk and proceeds to the right until it reaches the rightmost disk: compares every two adjacent disks and swaps them only if necessary. Now we have two light disks at the left-hand end and two dark disks at the right-hand end. Once it reaches the right-hand end, it starts with the rightmost disk, compares every two adjacent disks and proceeds to the left until it reaches the leftmost disk, doing the swaps only if necessary. The lawnmower movement is repeated $\lceil n/2 \rceil$ times.

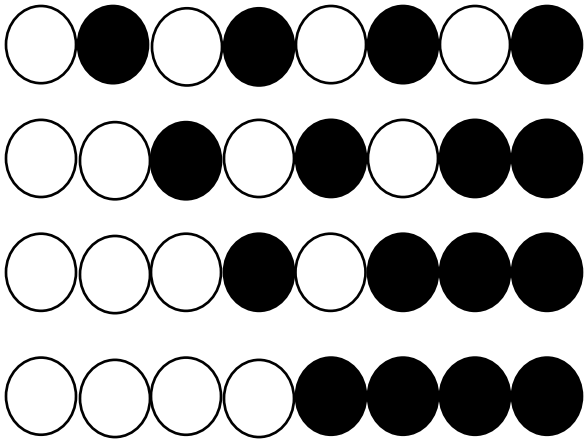
The left-to-right algorithm

It starts with the leftmost disk and proceeds to the right, doing the swaps as necessary. Now we have two lighter disks at the right-hand end and two darker disks at the left-hand end.Now we have two light disks at the left-hand end and two dark disks at the right-hand end. Once it reaches the right-hand end, it goes back to the leftmost disk and proceeds to the right, doing the swaps as necessary. It repeats `n` times.
For the example shown in the problem description, the exact list of disks changes as follows at the end of each run (we consider a run to be a check of adjacent disks from left-to-right):



The lawnmower algorithm

It starts with the leftmost disk and proceeds to the right until it reaches the rightmost disk: compares every two adjacent disks and swaps them only if necessary. Now we have two light disks at the left-hand end and two dark disks at the right-hand end. Once it reaches the right-hand end, it starts with the rightmost disk, compares every two adjacent disks and proceeds to the left until it reaches the leftmost disk, doing the swaps only if necessary. The lawnmower movement is repeated ⌈n/2⌉ times.
For the example shown in the problem description, the exact list of disks changes as follows at the end of each run (we consider a run to be a check of adjacent disks from left-to-right or right-to-left) is shown below:



Algorithm Design

Your first task is to design an algorithm for each of the two problems. Write clear pseudocode for each algorithm. This is not intended to be difficult; the algorithms I have in mind are all relatively simple, involving only familiar string operations and loops (nested loops in the case of oreos and substrings). Do not worry about making these algorithms exceptionally fast; the purpose of this experiment is to see whether observed timings correspond to big-O trends, not to design impressive algorithms.

Mathematical Analysis

Your next task is to analyze each of your two algorithms mathematically. You should prove a specific big-O efficiency class for each algorithm. These analyses should be routine, similar to the ones we have done in class and in the textbook. I expect each algorithm's efficiency class will be one of $O(n)$, $O(n^2)$, $O(n^3)$, or $O(n^4)$.