Project 1 (Due on 02/19)

Spring 2022 CPSC 335 - Algorithm Engineering

Instructor: Dr. Sampson Akwafuo

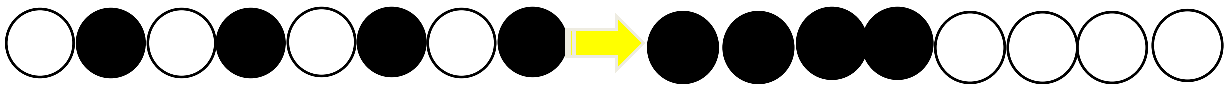
Abstract

In this project, description of a problem is given, followed by a suggested algorithm for solving the problem. You are expected to translate the description of the algorithm into a pseudocode, analyze your pseudocode mathematically; implement each algorithm in Python or C++; test your implementation; and describe your results.

The Alternating Disk Problem

The problem below is adapted from Ex. 14 on page 103 of Levitin's textbook::

> You have a row of $\boldsymbol{2n}$ disks of two colors, $\boldsymbol{n}$ light and $\boldsymbol{n}$ dark. They alternate: light, dark, light, dark, and so on. You want to get all the dark disks to the left hand side and all the light disks to the right hand end. The only moves you are allowed to make are those that interchange the positions of two neighboring disks. Design an algorithm for solving this puzzle and determine the number of moves it takes.



The alternating disks problem can be presented as:

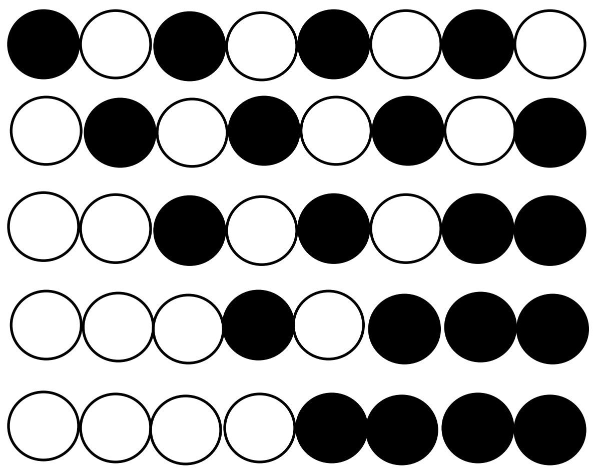
Input: a positive integer n and a list of $2n$ disks of alternating colors light-dark, starting with light
Output: a list of $2n$ disks, the first n disks are dark, the next n disks are light, and an integer m representing the number of swaps to move the light ones after the dark ones

There are more than one algorithm for solving this problem. The description of them is them is presented below. You need to translate the description of the algorithm into clear pseudocode.

The First Approach

This 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 <u>only if necessary</u>. Now we have one lighter disk at the left-hand end and the darker disk 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 second movement is repeated $\lceil n/2 \rceil$ times.

Consider the example below when n=4, and the first row is the input configuration, the second row is the end of comparison from left to right, the third row is the end of the first run (round trip that contains left to right followed by right to left), etc.. 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 the problem. Write a clear and simple pseudocode for the algorithm. The design will involve familiar string operations and loops. Do not worry about making the algorithm exceptionally fast; the purpose of this experiment is to see whether observed timings correspond to big-O trends, not to design impressive algorithms.

Do you think your suggested algorithm can be improved? If so, how?

Mathematical Analysis

Your next task is to analyze your algorithm mathematically. You should prove a specific big-O efficiency class for the algorithm, using any of the mathematical proving methods we discussed in class. Your analysis should be routine, similar to the ones we have done in class and in the textbook. The efficiency class of your algorithm is expected to be one of eight efficiency classes or a variation of them.

What to Do

1. Add your name or group member names to a README.md file. This file should also contain instructions on how run your submitted code.
2. Write your own pseudocode for the algorithm.
3. Analyze your pseudocode for the algorithm mathematically and prove its efficiency class.
4. Determine the number of moves your algorithm will make.
5. Implement your algorithm in Python or C++

Produce a brief written project report ***in PDF format***. Codes should be submitted in the executable version of the language, not in PDF.

Grading Rubric

The suggested grading rubric is given below.

a. Clear and complete Pseudocode = 20 points
b. Stating the correct Big *O* efficiency class of your algorithm = 5 points
c. Effectively proving your algorithm = 20 points
d. Successful compilation of codes= 20 points
e. Produces accurate results = 30 points
f. Statement on possible improvement = 5 points

Ensure your submissions are your own works. Be advised that your submissions may be checked for plagiarism using automated tools.

Submitting your code

Submit your files to the Project 1 Assignment on Canvas. It allows for multiple submissions. You can submit your files as a zip folder or separately. Submissions by email will not be accepted.

Deadline

The project deadline is **Saturday, February 19, 11:59 pm** on Canvas.

Penalty for late submission (within 48 hours) is as stated in the syllabus. Projects submitted more than 48 hours after the deadline will not be accepted.