

# Project 1: implementing algorithms

CPSC 335 - Algorithm Engineering

## Group:

Katie Tran([katiet419@csu.fullerton.edu](mailto:katiet419@csu.fullerton.edu))

Quan Duong([duongq99@csu.fullerton.edu](mailto:duongq99@csu.fullerton.edu))

## Abstract

In this project you will set up an environment for implementing algorithms in C++, and use that to implement two algorithms that solve the same problem. The first step is for you to become familiar with GitHub and makefiles. We will use GitHub to submit and grade projects all semester. The second step is for you to translate descriptions of two algorithms into pseudocode; analyze your pseudocode mathematically; implement each algorithm in C++; test your implementation; and describe your results. The third step is to do research on the Internet about the usefulness of having a .gitignore file and create the file .gitignore in your directory. Then upload everything on GitHub.

## What to Do

First, add your group members' names to README.md. Then:

1. Write your own pseudocode for the lawnmower and the alternate algorithms.
2. Compute the step count for each algorithm and prove its efficiency class using definition or limit theorem.
3. Implement all the skeleton functions in disks.hpp. Use the disks\_test.cpp program to test whether your code works.

Finally, produce a brief written project report *in PDF format*. Submit your PDF by committing it to your GitHub repository along with your code. Your report should include the following:

1. Your names, CSUF-supplied email address(es), and an indication that the submission is for project 1.
2. A full-screen screenshot, inside Tuffix, showing the Atom editor or the editor you used, with your group member names shown clearly below. One way to make your names appear in Atom is to simply open your README.md.
3. A full-screen screenshot showing your code compiling and executing.
4. Two pseudocode listings, for the two algorithms, and their step count.
5. A brief proof argument for the time complexity of your two algorithms.

# Grading Rubric

Your grade will consist of three parts: *Form*, *Function*, and *Analysis*.

*Function* refers to whether your code works properly as defined by the test program. We will use the score reported by the test program, when run inside the Tuffix environment, as your Function grade.

*Form* refers to the design, organization, and presentation of your code. A grader will read your code and evaluate these aspects of your submission.

*Analysis* refers to the correctness of your mathematical and empirical analyses, scatter plots, question answers, and the presentation of your report document.

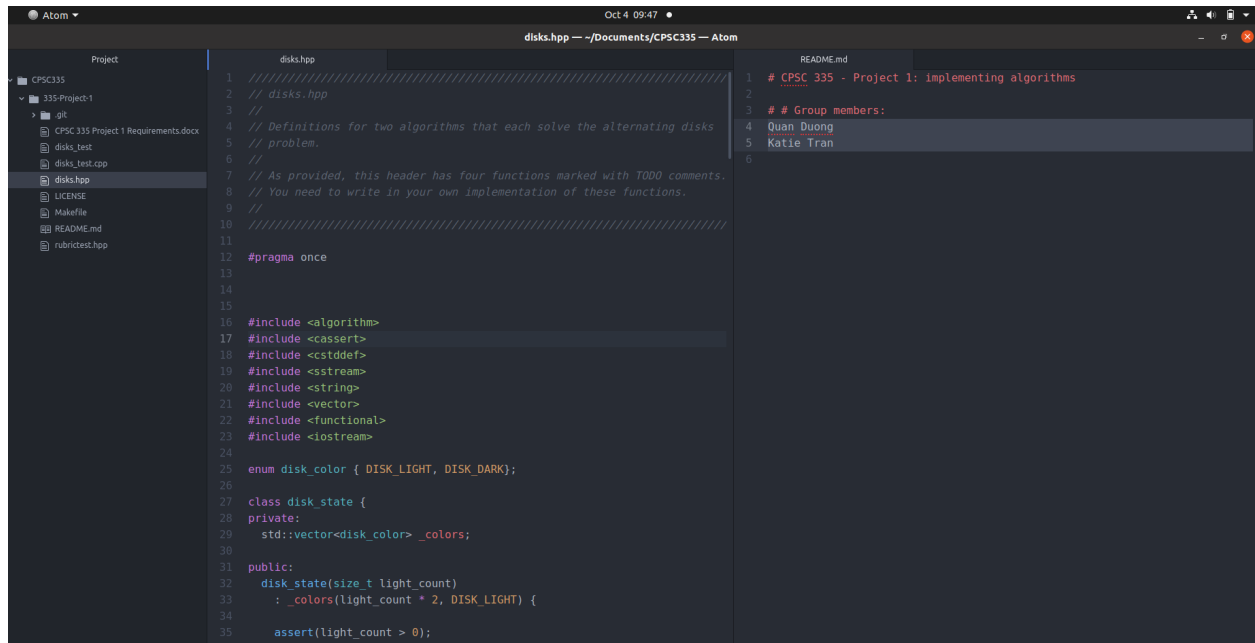
The grading rubric is below.

1. Function = 6 points, scored by the selected tests from the unit test program
2. Form = 11 points, divided as follows:
  - a. README.md completed clearly = 3 points
  - b. Style (whitespace, variable names, comments, helper functions, etc.) = 3 points
  - c. C++ Craftsmanship (appropriate handling of encapsulation, memory management, avoids gross inefficiency and taboo coding practices, etc.) = 3 points
  - d. File `.gitignore` created and populate it correctly = 2 points
3. Analysis = 23 points, divided as follows
  - a. Report document presentation = 3 points
  - b. Screenshots = 3 points each (total 6 points)
  - c. Pseudocode = 3 points each (total 6 points)
  - d. Mathematical analysis for each pseudocode = 4 points each (total 8 points)

*Legibility standard:* As stated on the syllabus, submissions that cannot compile in the Tuffix environment are considered unacceptable and will be assigned a 0 score for Function.

## Screenshots:

A full-screen screenshot, inside Tuffix, showing the Atom editor or the editor you used, with your group member names shown clearly below. One way to make your names appear in Atom is to simply open your README.md:

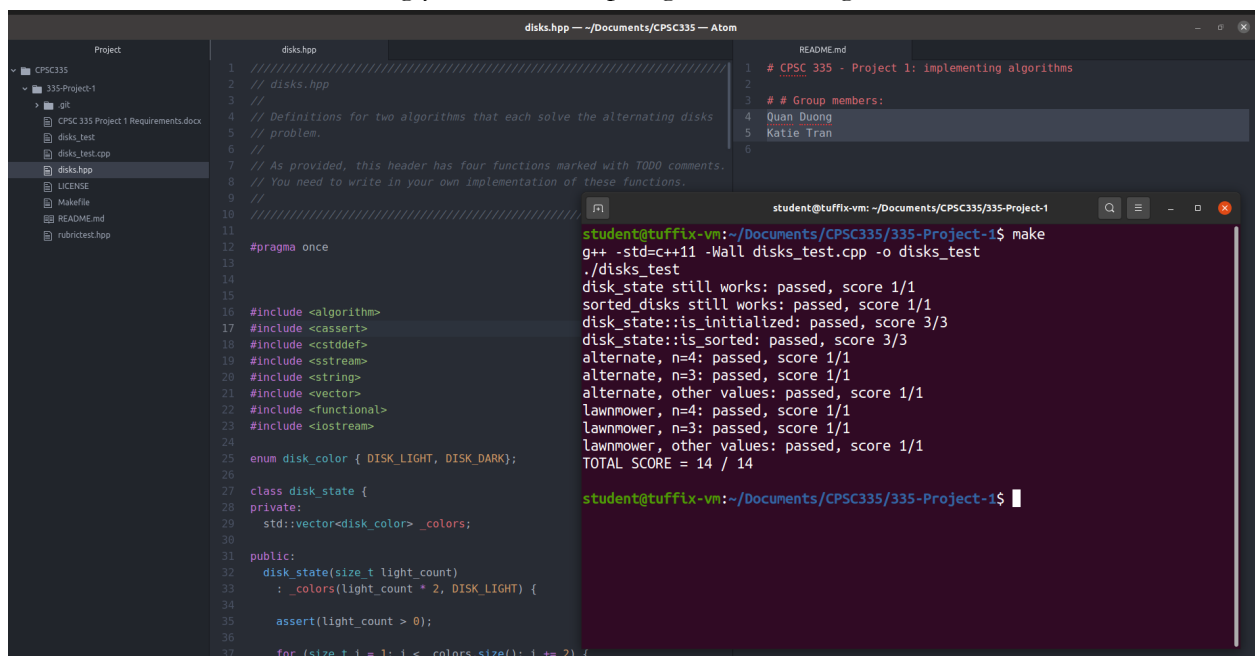


The screenshot shows the Atom editor interface. The left sidebar displays the project structure for CPSC335, including files like disks\_test.cpp, disks\_test.hpp, LICENSE, Makefile, README.md, and rubrictest.hpp. The main editor area is split into two panes. The left pane shows the contents of disks.hpp, which includes standard C++ headers, a namespace, and a class definition for disk\_state. The right pane shows the README.md file, which contains the project title and a list of group members: Quan Duong and Katie Tran.

```
1 // disks.hpp
2 //
3 //
4 // Definitions for two algorithms that each solve the alternating disks
5 // problem.
6 //
7 // As provided, this header has four functions marked with TODO comments.
8 // You need to write in your own implementation of these functions.
9 //
10 //
11 //
12 #pragma once
13
14
15
16 #include <algorithm>
17 #include <cassert>
18 #include <cstdlib>
19 #include <sstream>
20 #include <string>
21 #include <vector>
22 #include <functional>
23 #include <iostream>
24
25 enum disk_color { DISK_LIGHT, DISK_DARK};
26
27 class disk_state {
28 private:
29     std::vector<disk_color> _colors;
30
31 public:
32     disk_state(size_t light_count)
33         : _colors(light_count * 2, DISK_LIGHT) {
34
35         assert(light_count > 0);
36     }
37 }
```

```
1 # CPSC 335 - Project 1: implementing algorithms
2
3 # # Group members:
4 Quan Duong
5 Katie Tran
6
```

A full-screen screenshot showing your code compiling and executing:



The screenshot shows the Atom editor interface with the same project structure and file contents as the previous screenshot. A terminal window is open in the foreground, displaying the output of the compilation and execution of the code. The terminal shows the command to compile the code using g++ and the output of the disks\_test program, which reports the results of various tests and a total score of 14 / 14.

```
student@tuffix-vm: ~/Documents/CPSC335/335-Project-1$ make
g++ -std=c++11 -Wall disks_test.cpp -o disks_test
./disks_test
disk_state still works: passed, score 1/1
sorted_disks still works: passed, score 1/1
disk_state::is_initialized: passed, score 3/3
disk_state::is_sorted: passed, score 3/3
alternate, n=4: passed, score 1/1
alternate, n=3: passed, score 1/1
alternate, other values: passed, score 1/1
lawnmower, n=4: passed, score 1/1
lawnmower, n=3: passed, score 1/1
lawnmower, other values: passed, score 1/1
TOTAL SCORE = 14 / 14

student@tuffix-vm: ~/Documents/CPSC335/335-Project-1$
```

## Alternating algorithms:

**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.

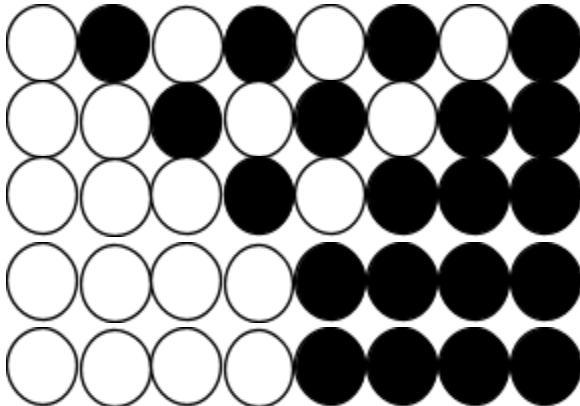
It does not iterate through each index, but iterates over each *pair* (i.e., it moves 2 steps at a time). We consider a run to be a check of adjacent disks from left-to-right. It has a total of  $n+1$  runs

**0 = white, 1 = black**

The first run compares the first[0] and second[1] disk, the third[2] and fourth[3] disk, the fifth and the sixth disk, etc.. The second run compares the second[1] and third[2] disk, the fourth[3] and the fifth[4], the sixth and the seventh disk, etc.. The third run is a repetition of the first run. The fourth run is a repetition of the second run, etc.. Again, swaps are done only if necessary.

Ex:

$N = 4$ , Total disks = 8



## Pseudo code:

```
int numOfSwap = 0;
```

```
For i=0 in n:
```

```
    If i%2==0:
```

```
        For j=0 in n:
```

```
            If arr[2*j] == dark & arr[2*j + 1] == light:
```

```
                Swap
```

```
                numOfSwap++
```

```
            Endif
```

```
        endfor
```

```
    Else:
```

```
        For j=0 in n-1:
```

```
            If arr[2*j+1] == dark & arr[2*j + 2] == light:
```

```
                Swap
```

```
                numOfSwap++
```

```
            Endif
```

```
        Endfor
```

```
Endfor
```

```
Return arr && swapcount
```

## Step Count:

```
int numOfSwap = 0; // 1 tu
```

```
For i=0 in n: // n - 0 + 1 => n + 1 times
```

```
    If i%2==0: // 2 tu
```

```
        For j=0 in n: // n + 1 times
```

```
            If arr[2*j] == dark & arr[2*j + 1] == light: // 5 tu
```

```
                Swap
```

```
                Swapcount++ // 1 tu
```

```
            Endif
```

```
        endfor
```

```
    Else: //
```

```
        For j=0 in n-1: // (n-1)+1 => n times
```

```
            If arr[2*j+1] == dark & arr[2*j + 2] == light: // 6 tu
```

```
                Swap
```

```
                Swapcount++ // 1 tu
```

```
            Endif
```

```
        Endfor
```

```
    Endfor
```

```
Return arr && swapcount
```

$$SC = 1 + (n+1) * (2 + \max(6n+6, 7n)) = 1 + (n+1)*(2+(7n)) = 7n^2+9n+3$$

Max: if block SC =  $(n+1)(5+1) = 6n+6$ , else block SC =  $(n)(6+1)=7n$

### Proving Big O Efficiency Class:

$O(g(n)) = f(n)$ : there exists positive constants  $c$  and  $n_0$  such that  $0 \leq f(n) \leq c(g(n))$  for all  $n \geq n_0$

Here  $f(n) = 7n^2 + 9n + 3$  and  $g(n) = n^2$  and  $c = 19$

$$f(n)/g(n) = (7n^2 + 9n + 3)/n^2 \leq c \cdot g(n)$$

$$f(n)/g(n) = (7n^2 + 9n + 3)/n^2 \leq 19n^2$$

Test:

$$n = 1$$

$$7 + 9 + 3 \leq 19 \quad \text{True}$$

$$n = 2$$

$$28 + 18 + 3 \leq 19 \cdot 4 \quad \text{True}$$

Thus  $7n^2 + 9n + 3$  belongs to  $O(n^2)$  as long as  $n \geq 1$

$O(g(n)) = f(n)$ : there exists positive constants  $c$  and  $n_0$  such that  $0 \leq f(n) \leq c(g(n))$ , for all  $n \geq n_0$

$$f(n) = 7n^2 + 9n + 3$$

$$g(n) = n^2$$

$$c = 19$$

$$\frac{f(n)}{g(n)} = \frac{7n^2 + 9n + 3}{n^2} \leq c * g(n)$$

$$\frac{f(n)}{g(n)} = \frac{7n^2 + 9n + 3}{n^2} \leq 19n^2$$

Test:

$$n_0 = 1$$

$$7(1)^2 + 9(1) + 3 \leq 19(1)^2$$

$$19 \leq 19$$

$$19 \leq 19 \quad \text{TRUE}$$

Test:

$$n_0 = 2$$

$$7(2)^2 + 9(2) + 3 \leq 19(2)^2$$

$$49 \leq 19(4)$$

$$49 \leq 76 \quad \text{TRUE}$$

$$7n^2 + 9n + 3 \leq 19n^2 \text{ for every } n \geq 1$$

$$7n^2 + 9n + 3 \text{ belongs to } O(n^2) \text{ as long as } n \geq 1$$



# The lawnmower algorithm

**Input:** a positive integer  $n$  and a list of  $2n$  disks of alternating colors light-dark, starting with light

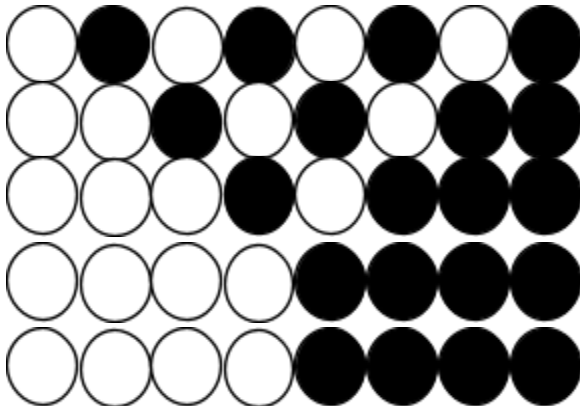
**Output:** a list of  $2n$  disks, the first  $n$  disks are light, the next  $n$  disks are dark, and an integer  $m$  representing the number of swaps to move the dark ones after the light ones.

solve this problem in  $O(n^2)$

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  $(n+1) / 2$  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:



**Pseudocode:**

Int leftAndRight = 0;

Int numOfSwap = 0;

Bool notCompleted = true;

While(notCompleted):

    notCompleted = false

    if leftAndRight%2 == 0:

        Int current = 0

        for j = 1 in array.size:

            if arr[current] > arr[j]:

                swap

                notCompleted= true

                numOfSwap++

                current= j

    else

        current= array.size - 1

        for j = array.size -2, j >= 0, j--:

            if arr[current] < arr[j]:

                swap

                notCompleted = true

                numOfSwap++

                current= j

    if(notCompleted == true):

        leftAndRight++

## Step Count:

Int leftAndRight = 0; // 1 tu

Int numOfSwap = 0; // 1 tu

Bool notCompleted = true; // 1 tu

While(notCompleted): n times

    notCompleted = false // 1 tu

    if leftAndRight%2 == 0: // 2 tu

        Int current = 0 // 1 tu

        for j = 1 in array.size: // n + 1 times

            if arr[current] > arr[j]: // 1 tu

                swap

                notCompleted = true: // 1 tu

                numOfSwap++ // 1 tu

                current = j // 1 tu

    else

        current = array.size - 1 // 2 tu

        for j = array.size to 0: // n + 1 times

            if arr[current] < arr[j]: // 1 tu

                swap

                notCompleted = true // 1 tu

                numOfSwap++ // 1 tu

                current = j // 1 tu

    if(notCompleted == true): // 1 tu

        leftAndRight++ // 1 tu

$$SC = 3 + n \cdot (1 + (2 + \max) + 2)$$

$$\text{Maxif} = 1 + (n+1) \cdot (1 + 1 + 1 + 1) = 4n + 5$$

$$\text{Maxelse} = 2 + (n+1) \cdot (1 + 1 + 1 + 1) = 4n + 6$$

So for max, max = 4n + 6

$$SC = 3 + n \cdot (1 + (2 + (4n + 6))) + 2 = 4n^2 + 11n + 3$$

### Proving Big O Efficiency Class:

$O(g(n)) = f(n)$ : there exists positive constants  $c$  and  $n_0$  such that  $0 \leq f(n) \leq c(g(n))$ , for all  $n \geq n_0$

$$f(n) = 4n^2 + 11n + 3$$

$$g(n) = n^2$$

$$c = 18$$

$$\frac{f(n)}{g(n)} = \frac{4n^2 + 11n + 3}{n^2} \leq c * g(n)$$

$$\frac{f(n)}{g(n)} = \frac{4n^2 + 11n + 3}{n^2} \leq 18n^2$$

Test:

$$n = 1$$

$$4(1)^2 + 11(1) + 3 \leq 18(1)^2$$

$$4 + 11 + 3 \leq 18$$

$$18 \leq 18 \quad \text{TRUE}$$

Test:

$$n = 2$$

$$4(2)^2 + 11(2) + 3 \leq 18(2)^2$$

$$16 + 22 + 3 \leq 18(4)$$

$$41 \leq 72 \quad \text{TRUE}$$

$4n^2 + 11n + 3$  belongs to  $O(n^2)$  as long as  $n \geq 1$