Project 1 Submission

Name: Kayla Ngo

Email: kngo29@csu.fullerton.edu

CWID: 885083436

Course: CPSC 335-13

Algorithm 1: The Alternating Disk Problem

Pseudocode

std::vector generate_disk_list(int n)

```
Purpose: creates the alternating disks list given an input n
Input: positive integer n
Output: vector (2n) of type int, where 0 is light-colored disks, 1 is dark-colored
disks

Let __list__ = a vector of ints
For each element in __list__:
   push_back light-colored disk = 0
   push_back dark-colored disk = 1
Return __list__
```

void print_disk_list(const std::vector& list)

```
Input: constant reference to the alternating disks list, list
Output: prints disk list in 0's and 1's Return: nothing

For every __disk__ in __list__:
   print the current element __disk__
```

int sort_disk_list(std::vector& list)

```
Purpose: modifies disk list to be sorted from dark to light
Input: reference to the alternating disks list, list
Output: sorted disk list (2n)

first n is dark-colored (1)
remaining n is light-colored (0)

Let __m__ = number of swaps
Let __swap__ = true

While swap holds true:
   __swap__ is false, no elements left to swap
   For each element in __list__:
      If (current_element = 0) and (next_element = 1):
         Swap the two elements (0,1) -> (1, 0)
      __m__++
```

__swap__ is true (0,1) -> (1,0)

Return __m__

Mathematical Analysis: Proof by Induction

```
std::vector<int> generate_disk_list(int n) {
    std::vector<int> list;             // +1

    for (int i = 0; i < n; i++) {       // n
        list.push_back(0);             // +2 atomics
        list.push_back(1);
    }

    return list;
}                                   // Overall: 2n+1


void print_disk_list(const std::vector<int>& list) {
    for (int disk : list) {
        std::cout << disk << " ";        // Overall: n
    }

    std::cout << std::endl;
}

int sort_disk_list(std::vector<int>& list) {
    int m = 0;      // +1
    bool swap;      // +1

    do {        // n
        swap = false;       // +1
        for (size_t i = 0; i < list.size()-1; i++) {        // n
            if ((list[i] == 0) && (list[i+1] == 1)) {       // +5
                int temp = list[i];     // +2
                list[i] = list[i+1];    // +3
                list[i+1] = temp;       // +2

                m++;        // +1
```

```
            swap = true;        // +1
        }
    }
} while (swap);     // Overall: 14n^2

return m;
}
```

This gives a total complexity of:
T(n) = 2n + 1 + n + 14n^2
T(n) = 14n^2 + 3n + 1

T(n) seems to resemble the quadratic efficiency class O(n^2).
Assume T(n) = 14n^2 + 3n + 1 and f(n) = n^2,

Base Case:

$When\, n = n_0,$

$T(n) \leq c \cdot f(n)$

$$14n^2 + 3n + 1 \leq c \cdot n^2$$

$$c = 14 + \frac{3}{n} + \frac{1}{n^2},\text{ n = 1}$$

$$= 14 + 3 + 1 = 18$$

$$14n^2 + 3n + 1 \leq c \cdot n^2$$

$$14(1)^2 + 3(1) + 1 \leq c \cdot (1)^2$$

$18 \leq c$ = 18 holds true

Inductive Case:

$If\ n > n_o,\ T(n) \leq c \cdot f(n),\ and\, c = 18,\ n = 1$

$Then\, T(n+1) \leq c \cdot f(n+1)$

$$14(n+1)^2 + 3(n+1) + 1 \leq c \cdot (n+1)^2$$

$$14(1+1)^2 + 3(1+1) + 1 \leq 18 \cdot (1+1)^2$$

$63 \leq 72$ holds true

Therefore, by definition of O,

$$14n^2 + 3n + 1 \in O(n^2)$$

Algorithm 2: Matching Group Schedules

Pseudocode

main.py

main () menu:

1. Add new person schedule
   a. Input: name - str, list of [start_time, end_time] stamps
   b. Output: master scheduler stored as a dictionary:
      - nested with list attributes of unavailable time stamps, time stamps in minutes, and available time stamps
      - key: name
2. View all person's schedule (master scheduler)
3. Find matching availability for all members
4. Exit scheduler

function.py

Master scheduler stored as a dictionary:

- nested with list attributes of unavailable time stamps, time stamps in minutes, and available time stamps
- key: name

Printing header()

Convert minutes()

Convert military()

Convert availability()

Mathematical Analysis: Proof by Induction