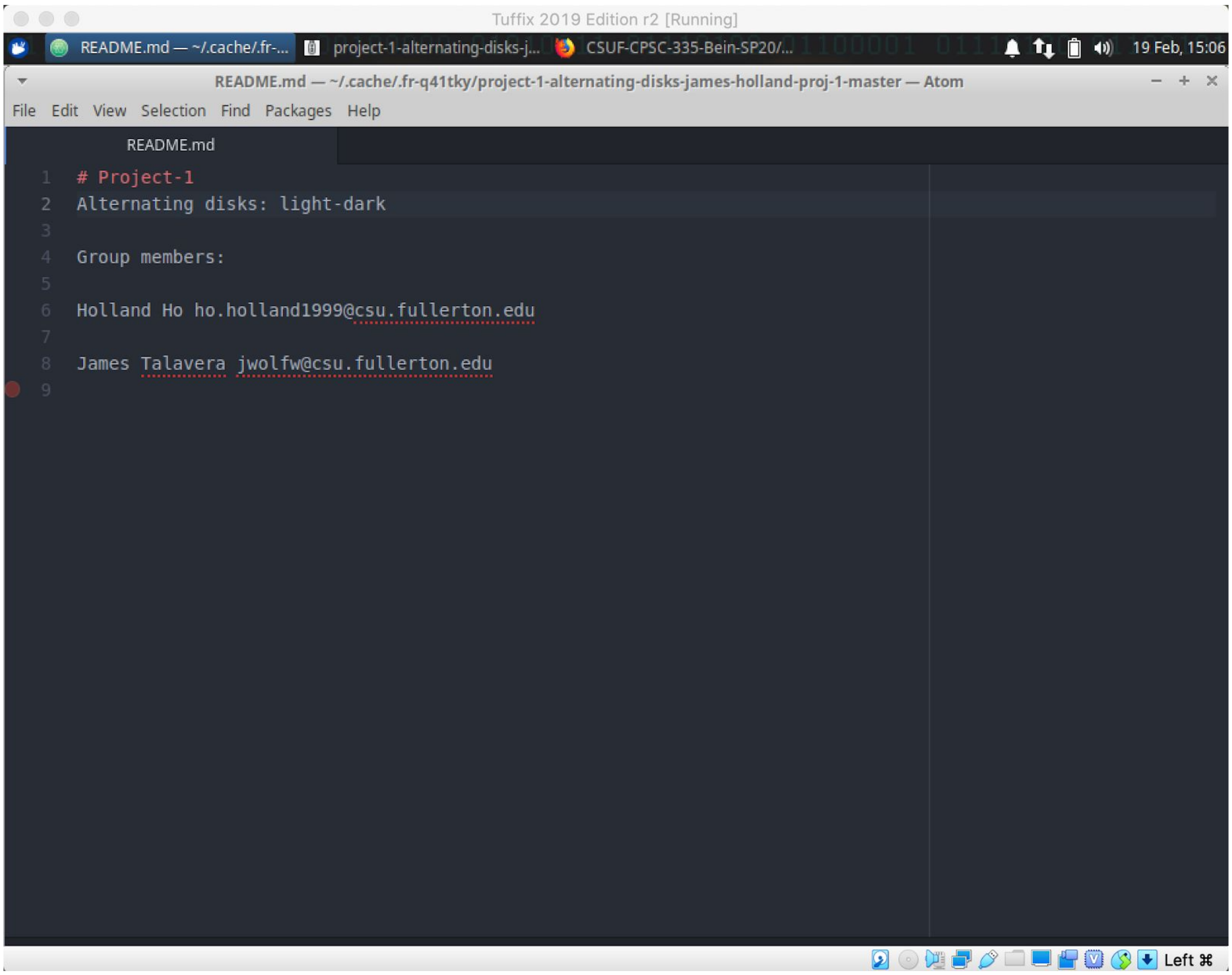
Project 1 Brief Written Report

Group Member Names:

Holland Ho ho.holland1999@csu.fullerton.edu

James Talavera jwolfw@csu.fullerton.edu

Screenshot:



Pseudocode listings:

1. **sort_left_to_right algorithm:**

```
# sort_left_to_right algorithm pseudocode

def sort_left_to_right(disk_state before):
   assert(before.is_alternating())
```

```
    disk_state temp = before
    swap_count = 0

    for i in range(temp.light_count()):
        for j in range(temp.total_count - 1):
            if temp.get(j) is DISK_DARK and temp.get(j+1) is DISK_LIGHT:
                temp.swap(j)
                swap_count += 1

    return sorted_disks(temp, swap_count)
```

2. **sort_lawnmower algorithm:**

```
# sort_lawnmower pseudocode

def sort_lawnmower(disk_state before):
    assert(before.is_alternating())
    disk_state after = before
    swap_count = 0

    for i in range(after.light_count()/2):
        for j in range(after.total_count()-1):
            if after.get(j) is DISK_DARK and after.get(j+1) is DISK_LIGHT:
                after.swap(j)
                swap_count +=1

        for k in range(after.total_count()-2,0,-1):
            if after.get(k) is DISK_DARK and after.get(k+1) is DISK_LIGHT:
                after.swap(k)
                swap_count += 1

    return sorted_disks(after, swap_count)
```

<u>Brief Proof Argument for time complexity:</u>

Proof of Step Count for is_alternating():

```
def is_alternating():
        if _colors[0] is DISK_DARK:  ← 2 tu    ⎤ 2+max(1,0)
                return false  ← 1 tu           ⎦ =2+1=3
        for i in range(_colors.size()-1): ← 2n        2n(7+7)
                                                      2n(14)
                                                      =28n
                if _colors[i] is DISK_LIGHT and _colors[i+1] is DISK_LIGHT: ← 6 tu ⎤ 6+max(1,0)
                        return false ← 1tu                                          ⎦ =6+1=7
                else if _colors[i] is DISK_DARK and _colors[i+1] is DISK_DARK ← 6 tu ⎤ 6+max(1,0)
                        return false ← 1tu                                           ⎦ 6+1=7
        return true ← 1tu

  total SC:
     3+1+28n = 28n+4
```

Left to Right Algorithm Time Complexity Proof:

```
def sort_left_to_right(disk_state before):
    assert(before.is_alternating()) ← 28n+4
    disk_state temp = before ← 1 tu
    swap_count = 0 ← 1 tu
    for i in range(temp.light_count()): ← n-1
        for j in range(temp.total_count - 1): ← 2n-1
            if temp.get(j) is DISK_DARK and temp.get(j+1) is DISK_LIGHT: ← 6tu
                temp.swap(j) ← 1 tu
                swap_count += 1 ← 1 tu

    return sorted_disks(temp, swap_count) ← 1 tu
```

$28n + 4 + 2 = 28n + 6$

$6 + \max(2, 0) = 8$

$2n-1(8) = 16n - 8$

$n-1(16n-8)$
$= 16n^2 - 8n - 16n + 8$
$= 16n^2 - 24n + 8$

total sc:

$28n + 6 + 16n^2 - 24n + 8 + 1$

$= 16n^2 + 4n + 15$

Proof:

$$\lim_{n \to \infty} \frac{16n^2}{n^2} + \frac{4n}{n^2} + \frac{15}{n^2} \in O(n^2)$$

$16 \geq 0$

therefore the algorithm is $O(n^2)$

Lawnmower Algorithm Time Complexity Proof:

```
def sort_lawnmower(disk_state before):
    assert(before.is_alternating()) ← 28n+4
    disk_state after = before ← 1tu
    swap_count = 0 ← 1tu
    for i in range(after.light_count()/2): ← n/2
        for j in range(after.total_count()-1): ← 2n-1
            if after.get(j) is DISK_DARK and after.get(j+1) is DISK_LIGHT: ← 6 tu
                after.swap(j) ← 1tu
                swap_count += 1 ← 1tu

        for k in range(after.total_count()-2, 0, -1): ← 2n-2
            if after.get(k) is DISK_DARK and after.get(k+1) is DISK_LIGHT: ← 6 tu
                after.swap(k) ← 1tu
                swap_count += 1 ← tu
    return sorted_disks(after, swap_count) ← 1tu
```

$28n+6$

$\frac{n}{2}(32n-24)$

$16n^2-12n$

$6+\max(2,0) = 8$

$2n-1(8) = 16n-8$

$6+\max(2,0) = 8$

$2n-2(8) = 16n-16$

$16n-8+16n-16$

$= \underline{32n-24}$

total sC:

$28n+6+16n^2-12n+1$

$= 16n^2+16n+7$

Proof:

$$\lim_{n\to\infty} \frac{16n^2}{n^2}+\frac{16n}{n^2}+\frac{7}{n^2} \in O(n^2)$$

$$16 \geq 0$$

therefore the algorithm is $O(n^2)$