Name: Alexander Zavaleta Alejandro Ramos
Email: A_zavaleta@csu.fullerton.edu alejandroramosh27@csu.fullerton.edu

Github:

https://github.com/zaavaleta33/project-lawnmover

Pseudocode and Step Count:

```
sorted_disks sort_alternate(const disk_state & before) do
int numOfSwap = 0; SC -> 1tu
disk_state af = before;  SC -> 1tu
for i = 0 in range total_count SC -> n+1 tu
   for j = 0 in range total_count step 2 SC ->  n/2 + 1 tu
     if j == DISK_DARK && j + 1 == DISK_LIGHT do SC -> 4 tu
       swap(t); SC -> 1 tu
       numOfSwap++; SC -> 1 tu
     end if
   end for
   for t = 1 in range total_count step 2 do SC -> (n-1)/2 + 1 - > n/2 + 1/2 u
     if a == DISK_DARK && a + 1 == DISK_LIGHT do SC -> 4 tu
       swap(a); SC -> 1 tu
       numOfSwap++; SC -> 1 tu
     end if
   end for
end for
return sorted_disks(disk_state(af), numOfSwap);
```

SC = 2 + ((n+1) * (((n/2) + 1) * 6) + ((n/2 + 1/2) * 6))
SC = 2+ ([n+1] * [[(3n)+6] + (3n + 3))
SC = 2 + ([n + 1] * (6n + 9))
SC = 6n^2 + 15n + 11

```
sorted_disks sort_lawnmower(const disk_state & before) do
   disk_state aft = before; SC -> 1 tu
  int num_swap = 0; SC -> 1 tu
   for i = 0 in range total_count do SC -> (n - 0)/1 + 1 tu -> n + 1 tu
   for j = 0 in range total_count do SC -> (n - 0)/1 + 1 tu -> n + 1 tu
     if (j) == DISK_DARK &&j + 1 == DISK_LIGHT do SC -> 4 tu
       aft.swap(j); SC -> 1 tu
       num_swap++;  SC -> 1 tu
     end if
   end for
```

```
    for t = total_count to 0 do SC -> ((0 - n)/-1) + 1  tu -> n + 1 tu
      if t == DISK_LIGHT && t - 1 == DISK_DARK do SC -> 4 tu
        swap(t - 1); SC -> 1 tu
        num_swap++; SC -> 1 tu
      end if
    end for
  end for
return sorted_disks(disk_state(aft), num_swap);
```

SC = 2 + (n + 1) * ([(n + 1) * (4 + max(2, 0)] + [(n + 1) * (4 + max(2, 0)])
SC = 2 + (n + 1)* ([(n + 1)*(6)] + [(n + 1) * 6])
SC = 2 + (n + 1) * ([6n + 6] + [6n + 6])
SC = 2 + (n + 1) * (12n + 12)
SC = 12n^2 + 24n + 12 + 2
SC = 12n^2 + 24n + 14

Mathematical Analysis:

SC for sort_alternate:
Let f(n) = 6n^2 + 15n + 11
Let g(n) = n^2
We must prove that f(n) <= c * g(n) for all n >= n0
Let c = 10, let n = 5, then plug into the inequality
6(5)^2 + 15(5) + 11 <= (10) * (5)^2
        236   <= 250
The inequality is true, therefore we can say that yes 6n^2 + 15n + 11 does belong 0(n^2)

SC for sort_lawnmower:
Let f(n) = 12n^2 + 24n + 14
Let g(n) = n^2
We must prove that f(n) <= c * g(n) for all n >= n0
Let c = 15, let n = 10, then plug into the inequality
12(10)^2 + 24(10) + 14 <= (15) * (10)^2
                    1454   <= 1500
The inequality is true, therefore we can say that yes 2n^2 + 24n + 14 does belong 0(n^2)

Video:

[Screen Recording 2023-03-18 at 6.56.41 PM.mov]

Screenshots:

```
  // Return true when this disk_state is fully sorted, with all light disks on
  // the left (low indices) and all dark disks on the right (high indices).
  bool is_sorted() const {
    for (size_t i = 0; i < total_count() - 1; i++) { //iterate through whole array
      if (i < total_count() / 2 && get(i) != DISK_LIGHT) { //check's to see if i is less than totalcount/2 and if light is not a light disk
        return false;
      }
      if (i > total_count() / 2 && get(i) != DISK_DARK) { //check's to see if i is greate then half the total_count and sees if it's a dark disk
        return false;
      }
    }
    return true;
```

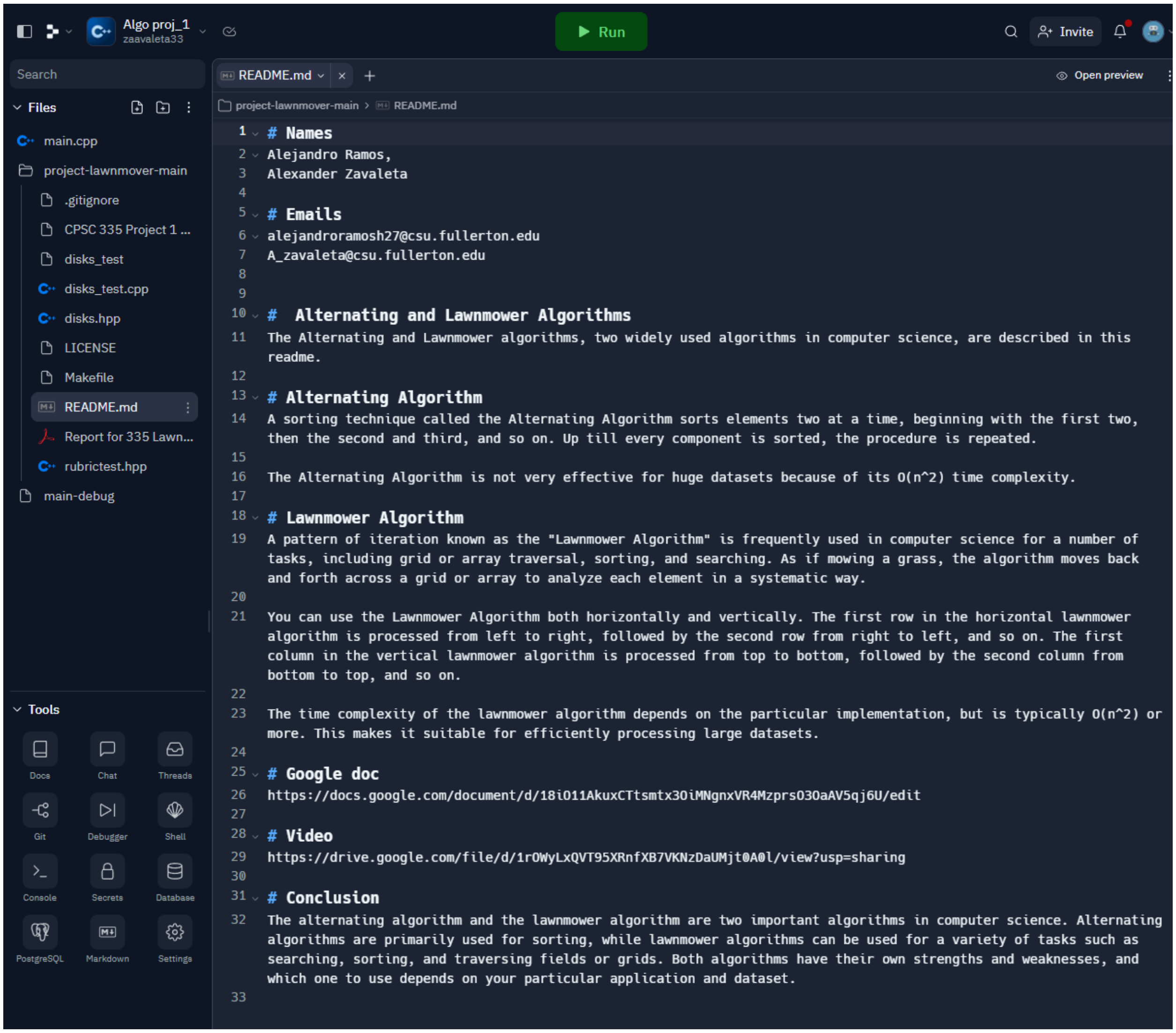
```cpp
// Algorithm that sorts disks using the alternate algorithm.
sorted_disks sort_alternate(const disk_state & before) {
  int numOfSwap = 0; //record # of step swap
  disk_state af = before; //copies everything from before into af
  for (size_t i = 0; i < af.total_count() - 1; i++) { //iterate thorugh each index
    for (size_t j = 0; j < af.total_count() - 1; j += 2) { //iterate through all the odd
      //!this first iteration goes through each and every iteration
      if (af.get(j) == DISK_DARK && af.get(j + 1) == DISK_LIGHT) {
        af.swap(j); ////swaps a and the next index
        numOfSwap++; ////add's to numswap counter
      }
    }
    for (size_t a = 1; a < af.total_count() - 1; a += 2) { //iterats through all the even
      if (af.get(a) == DISK_DARK && af.get(a + 1) == DISK_LIGHT) {
        af.swap(a); //swaps a and the next index
        numOfSwap++; //add's to numswap counter
      }
    }
  }
  return sorted_disks(disk_state(af /*state*/ ), numOfSwap);
}
```

```cpp

// Algorithm that sorts disks using the lawnmower algorithm.
sorted_disks sort_lawnmower(const disk_state & before) {
  //the only move we can make is when one disk is next to another one
  //second algorightm for this seems to be easir
  disk_state aft = before;
  int num_swap = 0;
  //Todo:start from the left and go to each iteration
  for (size_t i = 0; i < aft.total_count() - 1; i++) {
    for (size_t j = 0; j < aft.total_count() - 1; j++) { //iterates from left to right
      if (aft.get(j) == DISK_DARK && aft.get(j + 1) == DISK_LIGHT) {
        aft.swap(j); //swaps from left to right
        num_swap++; //increases the numswap
      }
    }
    for (size_t t = aft.total_count() - 1; t > 0; t--) { //iterates from right to left
      if (aft.get(t) == DISK_LIGHT && aft.get(t - 1) == DISK_DARK) {
        aft.swap(t - 1); //swaps from right to left
        num_swap++; //increases the numswap
      }
    }
  }
  //Todo: Once you reach the end you go from right to left
  return sorted_disks(disk_state(aft), num_swap);
}
```

```
 main         Makefile
 main-debug  'project-lawnmower-main REAL'
 main.o       replit.nix
> cd 'project-lawnmower-main REAL'
> ls
'CPSC 335 Project 1 Requirements.docx'
 disks.hpp
 disks_test
 disks_test.cpp
 LICENSE
 Makefile
 rubrictest.hpp
> g++ 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

>
```



Names
Alejandro Ramos,
Alexander Zavaleta

Emails
alejandroramosh27@csu.fullerton.edu
A_zavaleta@csu.fullerton.edu

Alternating and Lawnmower Algorithms
The Alternating and Lawnmower algorithms, two widely used algorithms in computer science, are described in this readme.

Alternating Algorithm
A sorting technique called the Alternating Algorithm sorts elements two at a time, beginning with the first two, then the second and third, and so on. Up till every component is sorted, the procedure is repeated.

The Alternating Algorithm is not very effective for huge datasets because of its O(n^2) time complexity.

Lawnmower Algorithm
A pattern of iteration known as the "Lawnmower Algorithm" is frequently used in computer science for a number of tasks, including grid or array traversal, sorting, and searching. As if mowing a grass, the algorithm moves back and forth across a grid or array to analyze each element in a systematic way.

You can use the Lawnmower Algorithm both horizontally and vertically. The first row in the horizontal lawnmower algorithm is processed from left to right, followed by the second row from right to left, and so on. The first column in the vertical lawnmower algorithm is processed from top to bottom, followed by the second column from bottom to top, and so on.

The time complexity of the lawnmower algorithm depends on the particular implementation, but is typically O(n^2) or more. This makes it suitable for efficiently processing large datasets.

Google doc
https://docs.google.com/document/d/18iO11AkuxCTtsmtx3OiMNgnxVR4MzprsO3OaAV5qj6U/edit

Video
https://drive.google.com/file/d/1rOWyLxQVT95XRnfXB7VKNzDaUMjt0A0l/view?usp=sharing

Conclusion
The alternating algorithm and the lawnmower algorithm are two important algorithms in computer science. Alternating algorithms are primarily used for sorting, while lawnmower algorithms can be used for a variety of tasks such as searching, sorting, and traversing fields or grids. Both algorithms have their own strengths and weaknesses, and which one to use depends on your particular application and dataset.