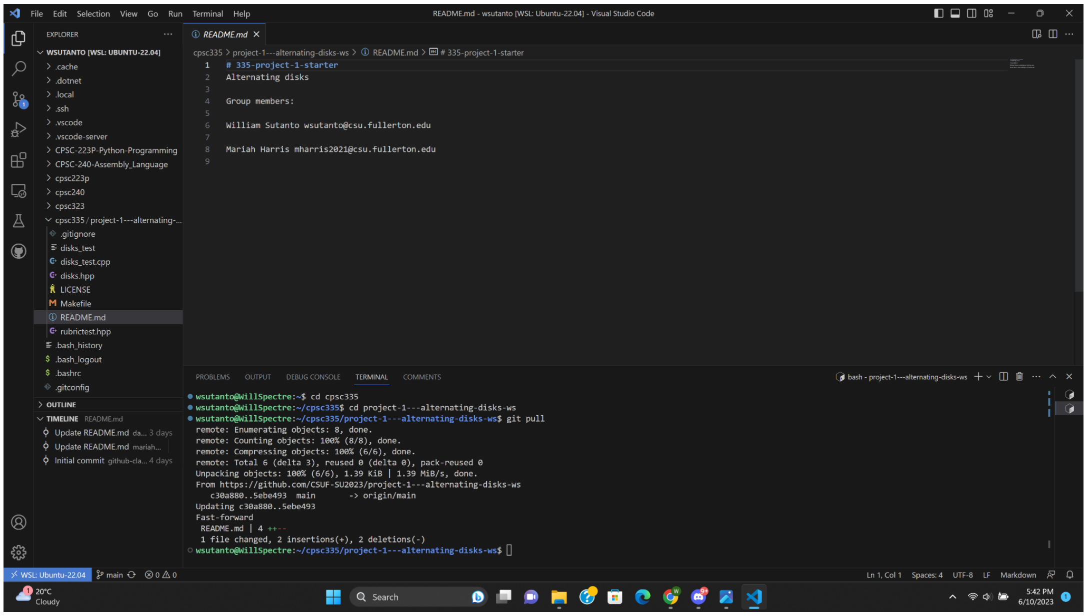
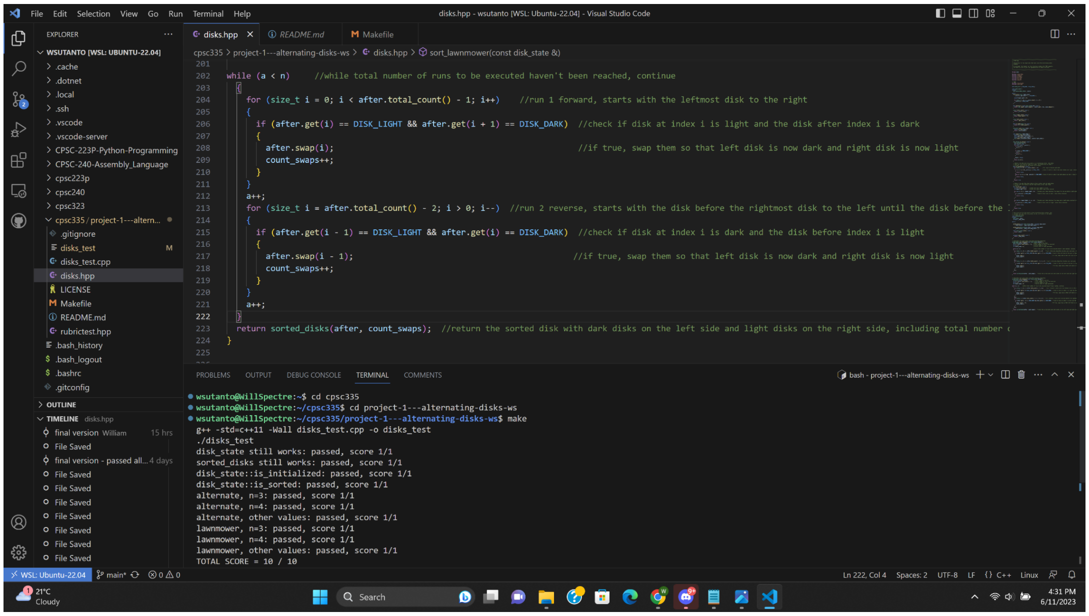
William Sutanto
wsutanto@csu.fullerton.edu
Mariah Harris
mharris2021@csu.fullerton.edu

Project 1 CPSC 335





Algorithm Design

1. Alternating Algorithm Pseudocode:

Function sorted_disks sort_alternate(before: disk_state):
  LET after be a copy of before
  // Calculate the total number of runs
  LET number of runs be the total number of disks divided by 2
  LET run counter be 0
  LET swap counter be 0

  WHILE run counter is less than number of runs **DO**:
// Run 1: Swap adjacent light and dark disks in pairs, starting from the leftmost disk to the rightmost disk
    FOR each index from 0 to the total number of disks minus with a step of 2 **DO**:
      IF the disk color at index is light and the disk color at index +1 is dark **THEN**:
        SWAP the disks at index and index +1 in the after disk_state
        INCREMENT the swap counter by 1
      END IF
    END FOR

    INCREMENT the run counter by 1

// Run 2: Swap adjacent light and dark disks in pairs, starting from the second leftmost disk to the second rightmost disk
    FOR each index from 1 the total number of disks minus 1 with a step of 2 **DO**:
      IF the disk color at index is light and the disk color at index + 1 is dark **THEN**:
        SWAP the disks at index and index + 1 in the after disk_state
        INCREMENT the swap count by 1
      END IF
    END FOR

    INCREMENT the run counter by 1

  END WHILE

  RETURN the sorted disk object and the swap count
END FUNCTION

2. Lawnmower Algorithm Pseudocode:

```
Function sorted_disks sort_lawnmower(before: disk_state):
  LET after be a copy of before
  // Calculate the total number of runs
  LET number of runs be the total number of disks divided by 2
  LET run counter be 0
  LET swap counter be 0

  WHILE run counter is less than number of runs DO:
    // Run 1: Move from left to right, starting with leftmost disk to the right
    FOR each index from 0 to the total number of disks minus 1 DO:
      IF the disk color at index is light and the disk color at index + 1 is dark THEN:
        SWAP the disks at index and index + 1 in the after disk_state
        INCREMENT the swap counter by 1
      END IF
    END FOR

    INCREMENT the run counter by 1

    // Run 2: Move from right to left, starting with the disk before the rightmost disk to the left
until the disk before the leftmost end
    FOR each index from the total number of disks minus 2 to 0 with a step of -1 DO:
      IF the disk color at index - 1 is light and the disk color at index is dark THEN:
        SWAP the disks at index - 1 and index in the after disk_state
        INCREMENT the swap counter by 1
      END IF
    END FOR

    INCREMENT the run counter 1

  END WHILE

  RETURN a sorted disk object and the swap counter
END FUNCTION
```

<u>Mathematical Analysis (Time Complexity)</u>

1. The Alternating Algorithm:

- Calculating the number of runs takes O(1) time.
- The while loop executes n times (where n is half the total disk count).

- The total disk count is m.

Within the while loop there are two for loops:

- The first for loop iterates over the disks with a step size of 2, meaning every other disk is skipped. This loop runs m iterations (where m is the total count of disks). Therefore, the first for loop has a time complexity of O(m).
- The second for loop iterates over the disks beginning with the second disk (index 1) and has a step size of 2. This loop runs m iterations (where m is the total count of disks). Therefore, the first for loop has a time complexity of O(m).

The checking disk color operation and swapping disk operations within the for loops are constant time operations, O(1). Because the while loop executes for half the total number of disks (n), the while loop can be written as O(m/2). The time complexity of the two for loops within the while loop is O(m/2 * m) = O(m^2/2) -> O(m^2).

In conclusion, the overall efficiency class for the alternating algorithm can be simplified to the dominating factor, O(n^2). It may be useful to further explore implementations of this algorithm to reduce time complexity, especially for a large input disk set.

Step Count:

```
disk_state after = before;          //SC = 1

int n = (after.total_count() / 2);  //SC = 3

int a = 0;                          //SC = 1

int count_swaps = 0;                //SC = 1


while (a < n)                   //SC = (n/2) times

{

  for (size_t i = 0; i < after.total_count(); i += 2)    //SC = ((n/2)+1) times

  {

    if (after.get(i) == DISK_LIGHT && after.get(i + 1) == DISK_DARK) // SC = 6

    {

      after.swap(i);      //SC = 1

      count_swaps++;  //SC = 1

    }// SC IF= 6 + max (2,0) = 6 + 1 = 8
```

```
   }// SC FOR = (n/2 + 1) * 8
   a++;            //SC = 1
   for (size_t i = 1; i < after.total_count() - 1; i += 2)  //SC= ((n-2)/2 + 1) times
   {
     if (after.get(i) == DISK_LIGHT && after.get(i + 1) == DISK_DARK)  //SC = 6
     {
       after.swap(i);  //SC= 1
       count_swaps++;  //SC = 1
     }// SC If = 6 + max(2,0) = 8
   }// SC For = ((n-2)/2 + 1) * 8
  a++;  // SC = 1
 }
 return sorted_disks(after, count_swaps); // SC = 0
}
```

SC = 6 + SC While * [(SC First For Foop) + 1 + (SC Second For Loop) + 1]
SC = 6 + (n/2) * [((n/2 + 1) * 8) + 1 + (((n-2)/2 + 1) * 8) + 1]
SC = 6 + (n/2)*[4n+8+1+4(n-2)+8+1]
SC = 6 + (n/2)*[4n+9+4n-8+8+1]
SC = 6 + (n/2)*[8n+10]
SC = 6 + 4n^2 + 5n

2. The Lawnmower Algorithm:
Calculating the number of runs takes O(1) time.
The while loop executes n times (n is half the total disk count). Therefore, the while loop is O(n).
The total disk count is m.
Within the while loop there are two for loops:

- The first for loop iterates from 0 to after.total_count() - 1. This loop has a time complexity of O(m) since it iterates over all the disks (m).
- The second for loop iterates from after.total_count() - 2 to 0. This loop has a time complexity of O(m) since it iterates over all the disks (m).

The total number of iterations performed by the for loops is O(2m). So, the time complexity for the for loops is O(m). Because the while loop executes for half the total number of disks (m), the while loop can be written as O(m/2). The time complexity of the two for loops within the while loop is O(m/2 * 2m) = O(m^2) .

In conclusion, the overall efficiency class for the lawnmower algorithm is O(n^2). It may be useful to further explore implementations of this algorithm to reduce time complexity, especially for a large input disk set.

Step count:

```
 disk_state after = before;           // SC = 1

 int n = (after.total_count() / 2);    //SC = 3

 int a = 0;                            //SC = 1

 int count_swaps = 0;                  //SC = 1


while (a < n)      //SC = (n/2) times

 {

  for (size_t i = 0; i < after.total_count() - 1; i++)    //SC = n times

  {

   if (after.get(i) == DISK_LIGHT && after.get(i + 1) == DISK_DARK)  //SC = 6

   {

    after.swap(i);                                //SC = 1

    count_swaps++;                                //SC = 1

   }SC If = 6 + max(2,0) = 8

  }// SC for = 8*n = 8n

  a++; //SC = 1

  for (size_t i = after.total_count() - 2; i > 0; i--)  //SC = (n-1) times
```

```
  {
    if (after.get(i - 1) == DISK_LIGHT && after.get(i) == DISK_DARK)  //SC = 6
    {
      after.swap(i - 1);   // SC = 2
      Count_swaps++; //SC = 1
    }//SC If = 6 + max(3,0) = 9
  }//SC For = (n-1)* 9 = 9n-9
  a++;            //SC = 1
 }
 return sorted_disks(after, count_swaps); // SC = 0
}
```

SC = 6 + SC While * [(SC First For Foop) + 1 + (SC Second For Loop) + 1]
SC = 6 + n/2 * [8n + 1 + 9n - 9 +1)]
SC = 6 + n/2 * [17n - 7]
SC = 6 + (17/2)n^2 - (7/2)n