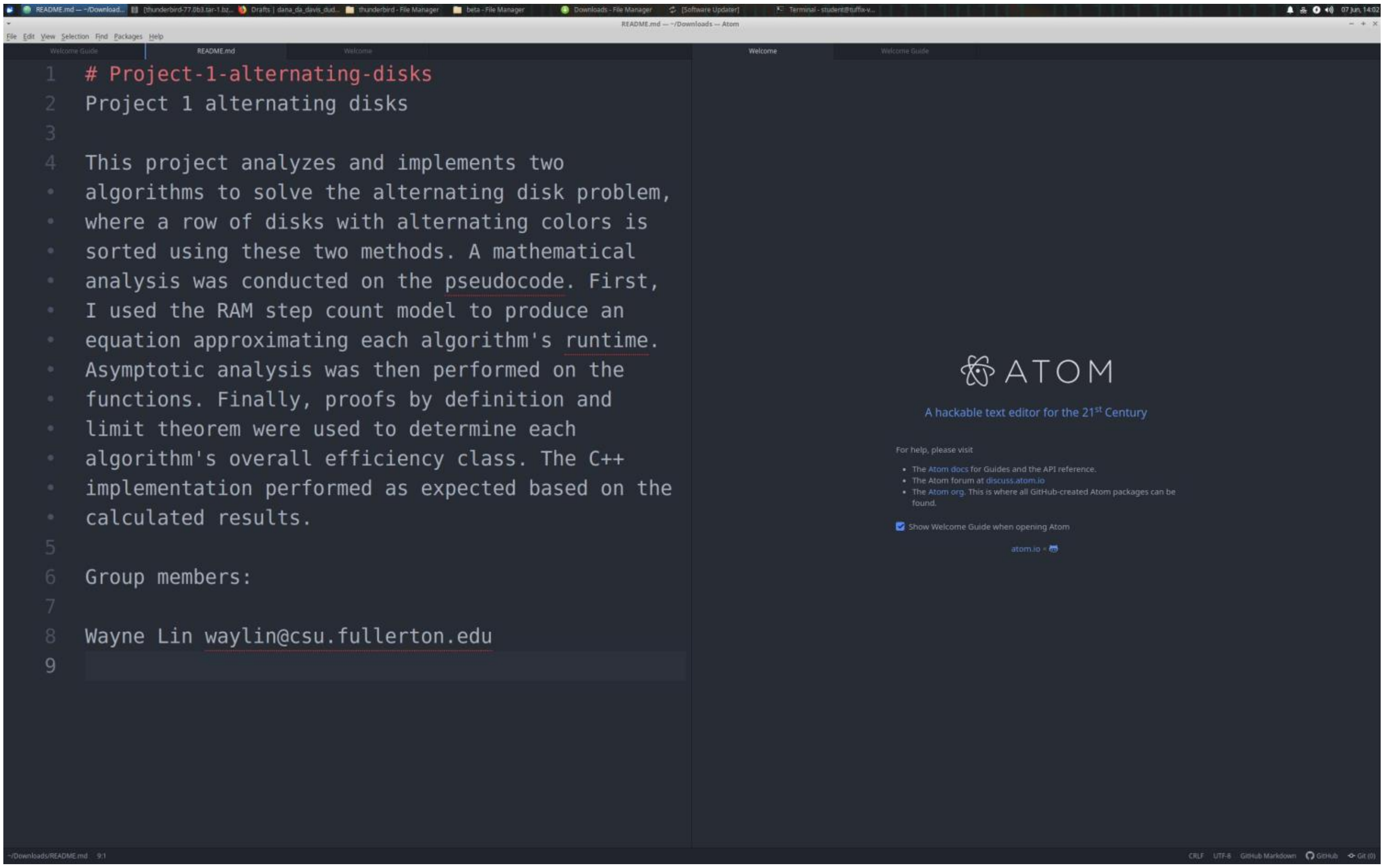
Project 1 Documentation: Alternating Disk Problem

Group members:

Wayne Lin waylin@csu.fullerton.edu



Pseudocode

Left-to-right Algorithm

Input: disk_state L representing a list of disk colors

Output: the sorted disk_state L and the swap counter

```
def left_to_right (L):
    let n = the number of elements in L
    numswaps = 0
    for k from 1 to n-1
        swapped = false
        for i from 0 to n-k-1 do
            // compare disk color of L[i] and L[i+1]
            if (disk color of L[i] is dark and disk color of L[i+1] is light) then
                swap the disk color of L[i] and L[i+1]
                swapped = true
                increment numswaps
            endif
        endfor
        if not swapped then return L and numswaps // the disks have already been sorted
    endfor
    return L and numswaps
```

Lawnmower Algorithm

Input: disk_state L representing a list of disk colors

Output: the sorted disk_state L and the swap counter

```
def lawnmower (L):
    let n = the number of elements in L
    numswaps = 0
    for k from 1 to n/2 - 1
        swapped = false
        for i from 0 to n - 2*k do
            // compare disk color of L[i] and L[i+1]
            if (disk color of L[i] is dark and disk color of L[i+1] is light) then
                swap the disk color of L[i] and L[i+1]
                swapped = true
                increment numswaps
            endif
        endfor
        if not swapped then return L and numswaps // break out of loop if already sorted
        for i from n - 2*k to 2 do
            // compare disk color of L[i-1] and L[i]
            if (disk color of L[i-1] is dark and disk color of L[i] is light) then
                swap the disk color of L[i-1] and L[i]
                swapped = true
                increment numswaps
            endif
        endfor
        if not swapped then return L and numswaps // break out of loop if already sorted
    endfor
    return L and numswaps
```

Mathematical Analysis

Left-to-right | Step Count

```
def left_to_right (L):
    let n = the number of elements in L                          // 1 step
    numswaps = 0                                                 // 1 step
    for k from 1 to n-1                                          // (n-1) times
        swapped = false                                          // 1 step
        for i from 0 to n-k-1 do                                 // (n-k) times
            // L.get(i) = color of disk at L[i]
            if (L.get(i) == dark && L.get(i+1) == light) then    // 6 steps
                swap the disk color of L[i] and L[i+1]           // 4 steps
                swapped = true                                   // 1 step
                numswaps = numswaps + 1                          // 2 steps
            else // do nothing                                   // 0
            endif
        endfor
        if not swapped then return L and numswaps // the disks have already been sorted    // 1 step
        else // do nothing                                       // 0
        endif

    endfor
    return L and numswaps                                        // 0 (return)
```

Step Count Calculation

Inner loop block: S.C. = 6 + max (4 + 1 + 2, 0) = 6 + 7 = 13 steps

Outer loop block: S.C. = 1 + max (1, 0) = 2 steps

Nonrepeated actions: S.C. = 1 + 1 = 2 steps

Total step count = 2 + sum_{k=1}^{n-1}(2 + sum_{i=0}^{n-k-1}(13))

= 2 + sum_{k=1}^{n-1}(2) + sum_{k=1}^{n-1}(sum_{i=0}^{n-k-1}(13))

= 2 + 2(n-1) + sum_{k=1}^{n-1}(sum_{i=0}^{n-k-1}(13))

= 2 + 2(n-1) + sum_{k=1}^{n-1}(13(n-k))

= 2 + 2(n-1) + sum_{k=1}^{n-1}(13n) - sum_{k=1}^{n-1}(13k)

= 2 + 2(n-1) + (n-1)*(13n) - 13*(n)*(n-1)/2

= 2 + 2n - 2 + 13n^2 - 13n - (13/2)*(n^2-n)

= 2n - 13n + 13n^2 - (13/2)n^2 + (13/2)n

= (13/2)n^2 - (13/2)n = (13/2)(n^2-n)

Proof by definition that (13/2)n^2 - (13/2)n belongs to O(n^2):

Let f(n) = (13/2)n^2 - (13/2)n and g(n) = n^2.

(13/2)n^2 - (13/2)n <= c * n, for some c > 0 and n >= n_0 > 0

Let us choose n_0 = 1 and c = 13:

(13/2)n^2 - (13/2)n <= 13 * n^2

(13/2)n^2 - (13/2)n <= (13/2)n^2 + (13/2)n^2

-(13/2)n <= (13/2)n^2

0 <= (13/2)n^2 + (13/2)n

n^2 + n >= 0

True for all n >= n_0 = 1

c * g(n) = 13 * n^2 is an upper bound of f(n) = (13/2)n^2 - (13/2)n.

Therefore, (13/2)n^2 - (13/2)n = O(n^2). The left-to-right algorithm is on the order of O(n^2).

Proof by limits that (13/2)n^2 - (13/2)n belongs to O(n^2):

Let T(n) (13/2)n^2 - (13/2)n and f(n) = n^2.

Then, lim_{n->inf}(T(n)/f(n)) = lim_{n->inf}([(13/2)n^2 - (13/2)n]/n^2)

= lim_{n->inf}([(13/2)n^2 - (13/2)n]'/[n^2]')

= lim_{n->inf}([13n - 13/2]/[2n])

= lim_{n->inf}([13n - 13/2]'/[2n]')

= 13/2 >= 0 and a constant

Therefore, we have proven that (13/2)n^2 - (13/2)n = O(n^2). The left-to-right algorithm is on the order of O(n^2).

Lawnmower   Step Count

```
def lawnmower (L):
    let n = the number of elements in L                                         // 1 step
    numswaps = 0                                                                // 1 step
    for k from 1 to n / 2 - 1                                                   // (n / 2 - 1) times
        swapped = false                                                         // 1 step
        for i from 0 to n - 2 * k do                                            // (n - 2 * k + 1) times
            // L.get(i) = color of disk at L[i]
            if (L.get(i) == dark && L.get(i+1) == light) then                   // 6 steps
                swap the disk color of L[i] and L[i+1]                          // 4 steps
                swapped = true                                                  // 1 step
                numswaps = numswaps + 1                                         // 2 steps
            else // do nothing                                                  // 0
            endif
        endfor
        if not swapped then return L and numswaps // break loop if already sorted  // 1 step
        else // do nothing                                                      // 0
        for i from n - 2*k to 2 step -1 do                                      // (n - 2 * k - 1) times
            // L.get(i) = color of disk at L[i]
            if (L.get(i-1) == dark && L.get(i) == light) then                   // 6 steps
                swap the disk color of L[i-1] and L[i]                          // 4 steps
                swapped = true                                                  // 1 step
                numswaps = numswaps + 1                                         // 2 steps
            else // do nothing                                                  // 0
            endif
        endfor
        if not swapped then return L and numswaps // break loop if already sorted  // 1 step
        else // do nothing                                                      // 0
    endfor
    return L and numswaps                                                       // 0
```

Step Count Calculation

Inner loop block A: S.C._A = 6 + max (4 + 1 + 2, 0) = 6 + 7 = 13 steps

Inner loop block B: S.C._B = 6 + max (4 + 1 + 2, 0) = 6 + 7 = 13 steps

Outer loop block: S.C. = 1 + max (1, 0) + max (1, 0) = 3 steps

Nonrepeated actions: S.C. = 1 + 1 = 2 steps

Total step count = 2 + sum_{k=1}^{n/2-1}(3 + sum_{i=0}^{n-2*k}(13) + sum_{i=n-2*k}^{2}(-1*13))

= 2 + sum_{k=1}^{n/2-1}(3) + sum_{k=1}^{n/2-1}(sum_{i=0}^{n-2*k}(13) + sum_{i=2}^{n-2*k}(13))

= 2 + 3(n/2 - 1) + sum_{k=1}^{n/2-1}(sum_{i=0}^{n-2*k}(13) + sum_{i=2}^{n-2*k}(13))

= 2 + 3(n/2 - 1) + sum_{k=1}^{n/2-1}(sum_{i=0}^{n-2*k}(13) + sum_{i=0}^{n-2*k}(13) - sum_{i=0}^{1}(13))

= 2 + 3n/2 - 3 + sum_{k=1}^{n/2-1}(sum_{i=0}^{n-2*k}(13+13) - sum_{i=0}^{1}(13))

= 2 + 3n/2 - 3 + sum_{k=1}^{n/2-1}(sum_{i=0}^{n-2*k}(26) - 26)

= 3n/2 - 1 + sum_{k=1}^{n/2-1}((1+n-2*k) * 26 - 26)

= 3n/2 - 1 + sum_{k=1}^{n/2-1}(26n - 52k)

= 3n/2 - 1 + 26[sum_{k=1}^{n/2-1}(n - 2k)]

= 3n/2 - 1 + 26[(n/2 - 1)(n/2) / 2 - 2(n/2 - 1)]

= 3n/2 - 1 + 26[((n^2)/8 - n/2 - n + 2)]

= 3n/2 - 1 + 13(n^2)/4 - 39n + 52

= 13(n^2)/4 - 75n/2 + 51

Proof by definition that 13(n^2)/4 - 75n/2 + 51 belongs to O(n^2):

Have f(n) = 13(n^2)/4 - 75n/2 + 51 and g(n) = n^2.

13(n^2)/4 - 75n/2 + 51n <= c * n, for some c > 0 and n >= n_0 > 0

Let us choose n_0 = 1 and c = ceil(13/4 +75/2 + 51) = ceil(367/4) = 368/4 = 82

Then we need to show that 13(n^2)/4 - 75n/2 + 51 <= 82 * n^2

13(n^2)/4 - 75n/2 + 51 <= 13(n^2)/4 + 355(n^2)/4

355(n^2)/4 + 75n/2 - 51 >= 0

355n^2 + 150n - 102 >= 0   True for all n >= 1

Thus, the function 13(n^2)/4 - 75n/2 + 51 is bounded at the top by 82n^2.

Therefore, 13(n^2)/4 - 75n/2 + 51 = O(n^2). The lawnmower algorithm is also on the order of O(n^2).

Proof by limits that 13(n^2)/4 - 75n/2 + 51 belongs to O(n^2):

Let T(n) 13(n^2)/4 - 75n/2 + 51 and f(n) = n^2. Then,

lim_{n->inf}(T(n)/f(n)) = lim_{n->inf}([13(n^2)/4 - 75n/2 + 51]/n^2)

= lim_{n->inf}([13(n^2)/4 - 75n/2 + 51]'/[n^2]')

= lim_{n->inf}([13n/2 - 75/2 + 0]/[2n])

= lim_{n->inf}([13n/2 - 75/2]'/[2n]')

= lim_{n->inf}([13/2]/[2])

= 13/4 which is >= 0 and a constant

Thus, 13(n^2)/4 - 75n/2 + 51 = O(n^2). The lawnmower algorithm is on the order of O(n^2).

Results

The two algorithms should behave similarly in runtime with respect to input size, both being on the order of O(n^2). This similarity increases as the sample size gets larger and the dominated values have a smaller impact on the runtime. The left-to-right algorithm has a larger (n^2) coefficient and less negative (n) coefficient, but a smaller constant factor. This should render it faster than the lawnmower algorithm at small n-values, but only within the same efficiency class. The lawnmower algorithm performs more efficiently at larger input sizes, due to the smaller (n^2) coefficient and highly negative (n) coefficient, but its constant factor makes it weaker at small input sizes.

In my test implementation, every sort was done in under 0.001 second. The program finished in an average of 0.97 seconds. Both algorithms are clearly sufficiently fast at the tested sample sizes despite growing at O(n^2). Their runtimes should remain viable in practice so long as n is not an inordinately large number.