

**Project 1**

**Pseudocode for Alternate Algorithm**

sort\_alternate(before)

after = before // 1tu

swap\_count = 0 //1tu

run\_count = 0 //1tu

if after.issorted() == true then // 8n + 16 tu

return sequence unchanged

else

if after.is\_initialized() == true then // tu

for i = 0 to n-1 do //outer for loop

if run\_count % 2 != 0 then //2 tu

for j = 0 to 2n-1, step 2, do //inner loop 1

if color[j] != color[j+1] then //2tu

swap() //5tu

swap\_count++ //1tu

endif

endfor

else

for j = 1 to 2n-2, step 2, do //inner loop 2

if color[j] != color[j+1] then //2tu

swap() //5tu

swap\_count++ //1tu

endif

endfor

endif

run\_count++ //1tu

endfor

else

return sequence unchanged

endif

endif

*// is\_initialized() function step count calculated to be tu*

*// is\_sorted() function step count calculated to be 8n + 16 tu*

*//swap() function determined to be 5tu*

**Proof and Time Complexity for Alternate Algorithm**

(Independent for loops)

outer for loop:

using + 1 :

+ 1 =

inner for loop 1:

+ 1 = n+

multiply with the if statement time units: 8( n + ) = 8n + 4 //2tu + 5tu + 1tu = 8tu

inner for loop 2:

+ 1 = n-

multiply with the if statement time units: 8( n- ) = 8n - 4

Combined:

outer for loop \* (2 + max(inner loop 1, inner loop 2) + 1 ) //+1 because of the run\_count increment

= n \* (2 + max(8n + 4, 8n - 4) + 1)

= \* (2 + (8n + 4) + 1)

= \* (8n + 7)

= 8n2 +

Add the time units outside the outer loop:

8n2 + 7n + 3 + + 8n + 16

= 8n2+ + 26

highest power is 2. Therefore, algorithm efficiency class is O(n2).

Proof by definition

Prove 8n2+ + 26 ∈ O(n2)

choose C > 0 and a value n­0 >= 0 such that:

8n2+ + 26 <= C \* n2 ∀ n >= n0

choose C = 8 + + 26 ≅ 52

8n2+ + 26 <= 52n2, true for n >= 1

choose n0 = 1

∴ 8n2+ + 26 ∈ O(n2)

**Pseudocode for Lawnmower Algorithm**

sort\_lawnmower(before)

after = before //1tu

swap\_count = 0 //1tu

temp\_n = total\_n - 1 //2tu

if after.issorted() == true then //*8n + 16 tu*

return sequence unchanged

else

if after.is\_initialized() == true then //  *tu*

for i = 0 to n-1 do //outer loop

for j = i to 2n-1, step 2, do //inner loop 1

if color[j] != color[j+1] then //2tu

swap() //5tu

swap\_count++ //1tu

endif

endfor

temp\_n-- //1tu

if temp\_n < i then //1tu

for k = 2n-1 down to i, step 2, do //inner loop 2

if color[k] != color[k-1] then //2tu

swap() //5tu

swap\_count++ //1tu

endif

endfor

endif

endfor

else

return sequence unchanged

endif

endif

*//swap() function determined to be 5tu*

**Proof and Time Complexity for Lawnmower Algorithm**

Proof Time Complexity

(Dependent for loops)

outer for loop:

part of this loop has 1 decrement, and 1 if statement. So multiply by 2tu

inner loop 1:

1 if statement with 2tu operations, and carries out 5tu function and 1tu increment. So, 2+5+1= 8tu

since step of 2, use + 1.

8 ( + 1) = 8 (n + )

inner loop 2:

1 if statement with 2tu operations, and carries out 5tu function and 1tu increment. So, 2+5+1= 8tu

since step of -2 use + 1.

8 ( + 1) = 8 (n + )

Combined:

(outer for loop) \* (inner loop 1 + inner loop 2)

[8 (n + ) + 8 (n + )]

= \* 8[(n + ) + (n + )]

= ( 2n + 2 – i ) = ( 2n – i + 2 )

=

= - + // = 16 \* 0 + = 16 ( )

= 16n(n-1 + 1) - 16 ( ) + 32(n-1 + 1)

= 16n2 – 8n2 – 8n + 32n

= 8n2 – 8n + 32

Add time units outside of outer loop:

8n2 – 8n + 32 + 4 + + 8n + 16

= 8n2 + + 59

highest power is 2. Therefore, algorithm efficiency class is O(n2).

Proof by definition

Prove 8n2 + + 59 ∈ O(n2)

choose C > 0 and a value n0 >= 0 such that:

8n2 + + 59 <= C \* n2 ∀ n >= n0

choose C = 8 + + 59 ≅ 70

8n2 + + 59 <= 70n2 , true for n >= 1

choose n0 = 1

∴ 8n2 + + 59 ∈ O(n2)