Project 1: implementing algorithms

CPSC 335 - Algorithm Engineering

Spring 2020

Instructors: Doina Bein ([dbein@fullerton.edu](mailto:dbein@fullerton.edu))

**Pseudo Code:**

**sorted\_disks sort\_left\_to\_right**

1.Start

2.Create first loop for number of transverse in the list(count of transverse depend on number of values in list)

3. Use second loop inside the first loop for swap of value which are not in ordered.(adjuecent swapping only) .transverse only in one direction

4.Repeat the step 2 until it complete transverse count.

5. Return the ordered list.

6. Stop

**sort\_lawnmower**

1.Start

2.Create first loop for number of transverse in the list(count of transverse depend on number of values in list)

3. Use second loop inside the first loop for swap of value which are not in ordered.(adjuecent swapping only)

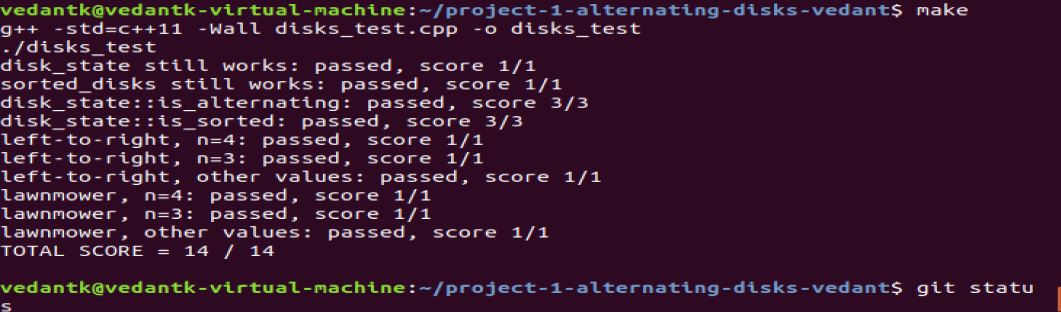
4. Use third loop inside the first loop, same as second loop but transverse in reverse direction as of second loop.(adjuecent swapping only)

5.Repeat the step 2 until it complete transverse count.

6. Return the ordered list.

7. Stop

**Screenshot of the result:**



**Mathematical analysis**

Get Function:

disk\_color get(size\_t index) const {

assert(is\_index(index)); ----O(1)

return \_colors[index]; ----O(1)

}

**Total: ----O(2)**

Swap Function:

void swap(size\_t left\_index) {

assert(is\_index(left\_index)); ----O(1)

auto right\_index = left\_index + 1; ----O(2)

assert(is\_index(right\_index)); ----O(1)

std::swap(\_colors[left\_index], \_colors[right\_index]); ----O(1)

}

**Total: ----O(5)**

Check Alternating:

bool is\_alternating() const {

for(std::vector<disk\_color>::size\_type i = 0; i < \_colors.size(); i+=2) { ----O(n/2)

if(\_colors[i]!=0 || \_colors[i+1]!=1) -----O(5)

{

return false; -----O(1)

}

}

return true; -----O(1)

}

**Total: -----5n/2 +1**

**Sorting**

**Sort left to right**

sorted\_disks sort\_left\_to\_right(const disk\_state& before) {

assert(before.is\_alternating()); ----O(5n/2 +1)

int swapCount=0; ----O(1)

disk\_state disks(before); ----O(1)

for(std::vector<disk\_color>::size\_type j = 0; j < disks.total\_count()-1; j++) ---O(n)==(12n^2)

{

for(std::vector<disk\_color>::size\_type i = j; i < disks.total\_count()-1; i++) { ----O(n)

if(disks.get(i)>disks.get(i+1)) ----O(6)

{

disks.swap(i); ----O(5)

swapCount++; ----O(1)

}

}

}

return sorted\_disks(disks, swapCount); ----O(1)

5n/2+1+2+12n^2

}

**Total: O(3+12n^2+2.5n))**

**Sort lawnmower**

sorted\_disks sort\_lawnmower(const disk\_state& before) {

assert(before.is\_alternating()); ----O(5n/2 +1)

int swapCount=0; --O(1)

disk\_state disks(before); --O(1)

for(std::vector<disk\_color>::size\_type j = 0; j < disks.total\_count()-1; j+=2) -O(n/2)

{

for(std::vector<disk\_color>::size\_type i = j; i < disks.total\_count()-1; i++) { -O(n)

if(disks.get(i)>disks.get(i+1)) --O(7)

{

disks.swap(i); --O(5)

swapCount++; --O(1)

}

}

for(std::vector<disk\_color>::size\_type i = disks.total\_count()-2; i>j; i--) --O(n)

{

if(disks.get(i)>disks.get(i+1)) --O(7)

{

disks.swap(i); ---O(5)

swapCount++; ---O(1)

}

}

}

return sorted\_disks(disks, swapCount); ----O(1)

}

1+1+n/2(n\*(7+6) + n\*(7+6))+1=4+13n^2 +5n/2 **Total: O(4+13n^2+5n/2)**