Jordhy Tapia

Pseudocode

Alternate sort Algorithm :

```
Sorted_disks sort_alternate(const disk_state &before)
disk_state after = before
int swapCount = 0

for every element in after
        for every other element in after
                if the element is not the last element in after
                        if current element is light and next element is dark
                                swap elements
                                increment swapCount
                        endIf
                endIf
        end for

        for every other element starting at 1 to after.total_count-1
                if current element is light and next element is dark
                                swap elements
                                increment swapCount
                endIf
        end for

        if after is sorted
                break out of for loop
        endIf
endFor

return after,swapCount

EndFunction sort_alternate
```

Lawn Mower Sort Algorithm:

```
Sorted_Disks sort_lawnMower(const disk_state& before)

Disk_state after = before
Int swapCount = 0

For every element in after
        For every element in after
                If current element is not the last element
                        If current element is light and next element is dark
                                Swap elements
                                Increase swapCount
                        endIf
                endIf
        endFor

        for every element starting at after.total_count -1, decrementing element position, and
        ending at element number 1
                if current element is dark and previous element is light
                        swap elements
                        increase swapCount
                endIf
        endFor

        if after is sorted
                break out of for loop
        endIf
endFor

return after, swapCount

endFunction sort_LawnMower
```

```
// Algorithm that sorts disks using the lawnmower algorithm.
sorted_disks sort_lawnmower(const disk_state& before)
{
   disk_state after = before;
   int swapCount = 0;

   for(int i = 0; i < after.total_count(); ++i)
   {
      for(int i = 0; i < after.total_count(); ++i)
      {
         if(i != after.total_count()-1)
         {
            if(after.get(i) == DISK_LIGHT && after.get(i+1) == DISK_DARK)
            {
               after.swap(i);
               swapCount++;
            }
         }
      }

      for(auto j = after.total_count()-1; j>1; j--)
      {
         if(after.get(j) == DISK_DARK && after.get(j-1) == DISK_LIGHT)
         {
            after.swap(j-1);
            swapCount++;
         }
      }
      if(after.is_sorted())
      break;
   }

  return sorted_disks(after, swapCount);
}
```

```
// Algorithm that sorts disks using the alternate algorithm.
sorted_disks sort_alternate(const disk_state& before)
{
   disk_state after = before;
   int swapCount= 0;

   for(int i = 0; i < after.total_count(); ++i)
   {
      for(int i = 0; i < after.total_count(); i+=2)
      {
         if(i != after.total_count()-1)
         {
            if(after.get(i) == DISK_LIGHT && after.get(i+1) == DISK_DARK)
            {
               after.swap(i);
               swapCount++;
            }
         }
      }
      for(int i = 1; i < after.total_count()-1; i+=2)
      {
         if(after.get(i) == DISK_LIGHT && after.get(i+1) == DISK_DARK)
         {
            after.swap(i);
            swapCount++;
         }
      }

      if(after.is_sorted())
         break;
   }

  return sorted_disks(after, swapCount);
}
```

Mathematical Analysis

```
// Algorithm that sorts disks using the lawnmower algorithm.
sorted_disks sort_lawnmower(const disk_state& before)
{
   disk_state after = before;
   int swapCount = 0;

   for(int i = 0; i < after.total_count(); ++i)
   {
      for(int i = 0; i < after.total_count(); ++i)
      {
         if(i != after.total_count()-1)
         {
            if(after.get(i) == DISK_LIGHT && after.get(i+1) == DISK_DARK)
            {
               after.swap(i);
               swapCount++;
            }
         }
      }

      for(auto j = after.total_count()-1; j>1; j--)
      {
         if(after.get(j) == DISK_DARK && after.get(j-1) == DISK_LIGHT)
         {
            after.swap(j-1);
            swapCount++;
         }
      }
      if(after.is_sorted())
      break;
   }

 return sorted_disks(after, swapCount);
}
```

1 s.c
1 s.c → 2 s.c

$\sum_{i=0}^{n} \left(\sum_{i=0}^{n} (8) + \sum_{i=1}^{n-1} (6) \right) + 4$

8 − 6

2 s.c + max(4 s.c + max(2 s.c, 0))

6

4 s.c + max(2, 0)

+1 s.c

+1 s.c

$\sum_{i=0}^{n} \left(\sum_{i=0}^{n} 8 + \sum_{i=1}^{n-1} 6 \right) + 4 \rightarrow \sum_{i=0}^{n} \left(8(n+1) + 6(n-1) \right) + 4$

14n + 2

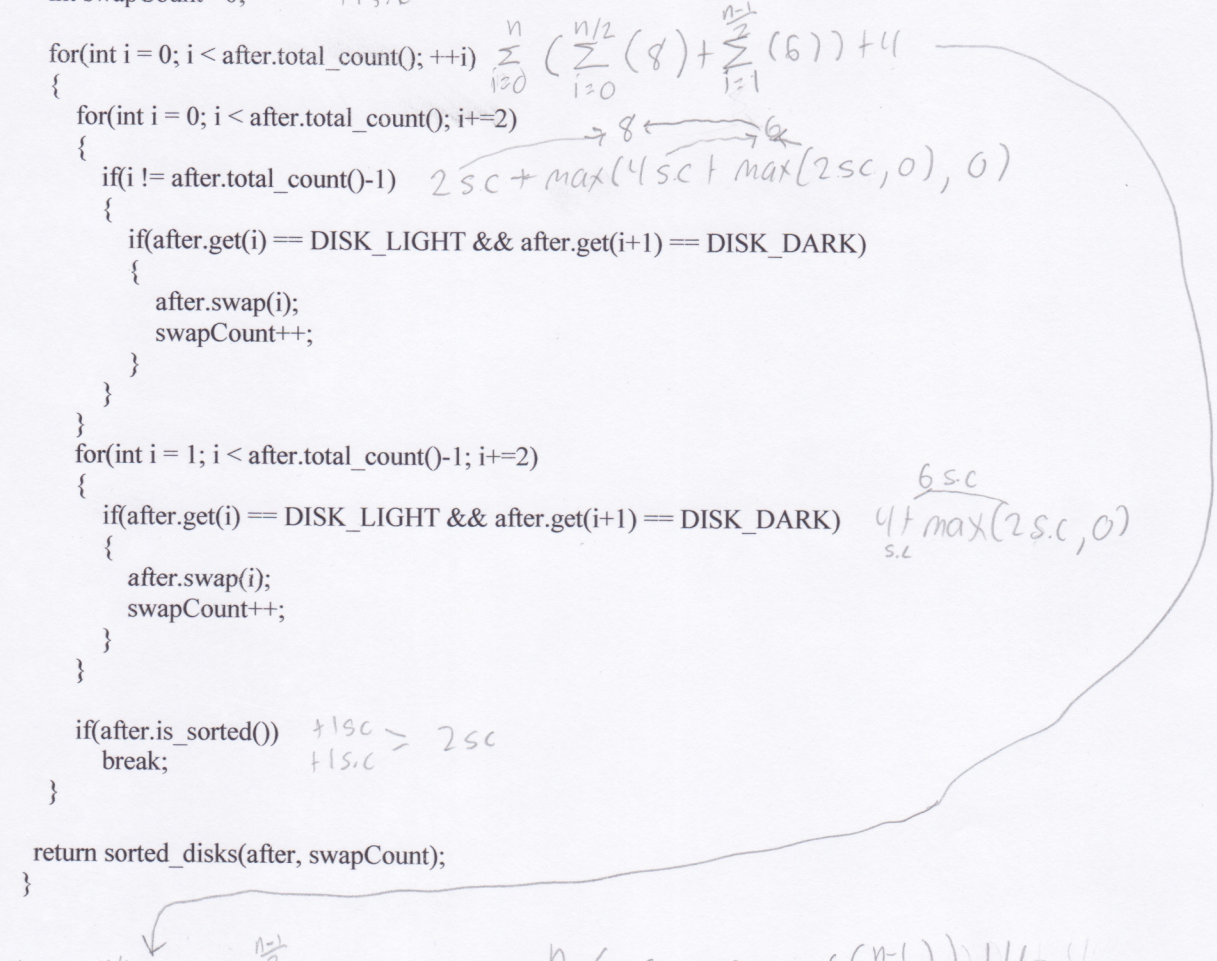
$\sum_{i=0}^{n} (14n+2) + 4 \rightarrow (14n+2)(n+1) + 4 \rightarrow \boxed{14n^2 + 16n + 6} \rightarrow \boxed{O(n^2)}$

```
// Algorithm that sorts disks using the alternate algorithm.
sorted_disks sort_alternate(const disk_state& before)
{
   disk_state after = before;
   int swapCount= 0;

   for(int i = 0; i < after.total_count(); ++i)
   {
      for(int i = 0; i < after.total_count(); i+=2)
      {
         if(i != after.total_count()-1)
         {
            if(after.get(i) == DISK_LIGHT && after.get(i+1) == DISK_DARK)
            {
               after.swap(i);
               swapCount++;
            }
         }
      }
      for(int i = 1; i < after.total_count()-1; i+=2)
      {
         if(after.get(i) == DISK_LIGHT && after.get(i+1) == DISK_DARK)
         {
            after.swap(i);
            swapCount++;
         }
      }

      if(after.is_sorted())
         break;
   }

 return sorted_disks(after, swapCount);
}
```



$$\sum_{i=0}^{n} \left(\sum_{i=0}^{n/2} (8) + \sum_{i=1}^{\frac{n-1}{2}} (6) \right) + 4 \rightarrow \sum_{i=0}^{n} \left(8\left(\frac{n}{2}+1\right) + 6\left(\frac{n-1}{2}\right) \right) + 4 + 4$$

$$\sum_{i=0}^{n} (4n+8+3n-3)+4 \rightarrow \sum_{i=0}^{n} (7n+5)+4 \rightarrow (7n+5)(n+1)+4 = 7n^2+12n+9$$

$$O(n^2)$$