CPSC 335-01 Project 1 Documentation

Group members: Collin Chiu, Stephanie Becerra, Brian Joh

Design and Implementation

To generate the problem within our code, we decided to use a list data structure to implement the list of circles. Since our program doesn't have a graphical interface, we decided to use 1's to represent light circles, and 0's to represent dark circles.

The function genList() populates the list based on the user input *n*, the number of circle pairs in the list. Therefore, there are *2n* circles of alternating colors - light and dark - in the list. To generate the list, we implemented a for loop to keep track of the amount of circles in the list, and based on the modulus of the integer counter *i*, would decide to add a 0 or a 1 to the front of the list.

The function printList() utilizes a for loop to print all the elements in the list in a formatted manner.

The function sortList() creates two list iterators, which will be initialized later. An outer for loop keeps track of the amount of times looped, which should not exceed the number of items in the list, *n*. A nested for loop initializes the first iterator (it1) to the beginning of the list, and the second iterator (it2) to the next element right to the first iterator. Both iterators traverse the list, moving to the next element one at a time, checking if the value of it1 is equal to 0, and if the value of it2 is equal to 1. If so, it will swap the two elements; else, it will continue to the next two elements. This process repeats *n* times to make sure the entire list is sorted. Every time two elements are swapped, a counter swapCount keeps track of the number of swaps made.

The main function simply runs the genList() and sortList() in sequence, then terminates.

Possible Improvements

While the code itself works quite well, we believe that there is one way to make this program more simple and effective. Instead of moving the circles from left to right from neighboring circles, we deduced that by iterating and sorting the circles by going from the beginning and ending parts of the list and going towards the middle. Once the program meets in the middle it will go back to the ends and sort the program again if needed.

Another improvement that we noticed is that our algorithm could've been improved if the iterators started at the two last elements in the list, then moved left, checking each pair for the conditions to swap. This would result in less swaps and would therefore be more efficient.

Pseudocode

Function sortList():

```
let swapCount = 0

for (i in circleList)
     for (let i = first element in circleList, let j = second element in circleList; move i and j right every loop iteration; loop until j is end of list)
          if i = 0 and j = 1
               swap elements i and j
               swapCount++
          else
               continue to next iteration
```

Mathematical Analysis

Since our program uses nested loops, we deduced that this program has a similarity to the O(n^2) time complexity. Since we have 2 functions, we deduced that T(n) = n^2 + 2. The method we used was induction.

1. T(n) = n^2 + 2 seems to resemble the quadratic efficiency class O(n^2).

2. Find c

$n^2 + 2 \leq c \cdot n^2$

$c \geq \frac{n^2+2}{n^2} = 1 + \frac{2}{n^2}$

n must be ≥ 1, we assume $c = 3$ and $n_0 = 1$

3. $1^2 + 2 \leq 3 \cdot 1$

$3 \leq 3$

4. If $n > n_0$ and $T(n) \leq c \cdot f(n)$; then $T(n+1) \leq c \cdot f(n+1)$

$T(n) \leq c \cdot f(n)$

$n^2 + 2 \leq 3 \cdot n^2$

$(n+1)^2 + 2 \leq 3 \cdot (n+1)^2$

$n^2 + 2n + 1 + 2 \leq 3n^2 + 6n + 3$

$n^2 + 2n + 3 \leq 3n^2 + 6n + 3$

Therefore

$T(n+1) \leq c \cdot f(n+1)$

5. Sequel to (1)-(3) and by def. of O

$n^2 + 2 \in O(n^2)$