

## CS 430 - Homework 3, solutions sketches

**Problem 1** Suppose we are given a sequence  $S$  of  $n$  elements, each of which is colored red or blue. Assuming  $S$  is represented as an array, give an in-place method for ordering  $S$  so that all the blue elements are listed before all the red elements. Can you extend your approach to three colors? In-place means that only swap operations are allowed.

### Solution Sketch:

We use a quicksort variant, simplified a bit. Our arrays is  $p \dots q$ . Put all elements of firstColor in front. Return the index of the first element that is not of the first color.

```

SORT-TWO-COLOR(S,p,q,firstColor)
1   $i = p; j = q;$ 
2  while  $i < j$ 
3    if  $S[i] == \text{firstColor}$ 
4       $i = i + 1$ 
5    elseif  $S[j] \neq \text{firstColor}$ 
6       $j = j - 1$ 
7    else
8      swap ( $S[i], S[j]$ )
9       $i = i + 1; j = j - 1$ 
10   endif
11  endwhile
12  return  $i$ 
```

For the running time, note that each time the inside of the **while** loop (lines 3-10) is executed,  $j - i$  decreases, so there can be at most  $p - q + 1$  such executions. The overall running time is  $O(n)$  for an  $n$ -length array.

```

SORT-THREE-COLOR(S,firstColor,secondColor)
1   $i \leftarrow \text{SORT-TWO-COLOR}(S,0,n,\text{firstColor})$ 
2   $\text{SORT-TWO-COLOR}(S,i,n,\text{secondColor})$ 
```

The "firstColor" stands for the color that is before another color in two color cases. And the "secondColor" stands for the color that will be after the first color and before the third color in three color cases.

**Problem 2** Let  $S$  be an array with  $n$  integers. An *inversion* in  $S$  is a pair of elements  $x$  and  $y$  such that  $x$  appears before  $y$  in  $S$  but  $x > y$ . Describe an algorithm running in  $O(n \log n)$  time for determining the *number* of inversions in  $S$ . Present pseudocode, and justify both correctness and running time.

**Hint:** Try to modify the merge-sort algorithm to solve the problem. Consider running merge-sort in parallel with your main algorithm

**Solution Sketch:**

NUMBER-OF-INVERSION  $\leftarrow$  0 /\*Global variable\*/

```
MERGESORT(m)
1 if length(m)  $\leq$  1
2   then return m
3 else
4   middle  $\leftarrow$  length(m)/2
5   for each x in m up to middle
6     add x to left
7   for each x in m after middle
8     add x to right
9   left = MERGESORT(left)
10  right = MERGESORT(right)
11  result = MERGE(left,right)
12  return result
```

```
MERGE(left,right)
1 i  $\leftarrow$  0
2 j  $\leftarrow$  0
3 k  $\leftarrow$  0
4 while i < left.size and j < right.size
5   if left[i] < right[j]
6     then result[k]  $\leftarrow$  left[i]
7     k  $\leftarrow$  k + 1
8     i  $\leftarrow$  i + 1
9   else result[k]  $\leftarrow$  right[j]
10    k  $\leftarrow$  k + 1
11    j  $\leftarrow$  j + 1
12    NUMBER-OF-INVERSION  $\leftarrow$  NUMBER-OF-INVERSION + left.size - i
13 while i < left.size
14   result[k]  $\leftarrow$  left[i]
15   k  $\leftarrow$  k + 1
16   i  $\leftarrow$  i + 1
17 while j < right.size
18   result[k]  $\leftarrow$  right[j]
19   k  $\leftarrow$  k + 1
20   j  $\leftarrow$  j + 1
21 return result
```

**Problem 3** Characterize each of the following recurrence equations using the master method (assuming that  $T(n) = c$  for  $n < d$ , for constants  $d \geq 1$ ).

a.  $T(n) = 2T(n/2) + \log n$

b.  $T(n) = 8T(n/2) + n^2$

c.  $T(n) = 16T(n/2) + (n \log n)^4$

d.  $T(n) = 7T(n/3) + n$

e.  $T(n) = 9T(n/3) + n^3 \log n$

**Solution:**

- a.  $T(n) = 2T(n/2) + \log n$ . In the Master Theorem, we have  $a = 2$ ,  $b = 2$  and  $f(n) = \log n$ . Then  $\log_b a = 1$ . As  $f(n) = O(n^{0.99})$ , the first case applies and we conclude  $T(n) = \Theta(n^{\log_2 2}) = \Theta(n)$ .
- b.  $T(n) = 8T(n/2) + n^2$ . In the Master Theorem, we have  $a = 8$ ,  $b = 2$  and  $f(n) = n^2$ . Then  $\log_b a = 3$ . As  $f(n) = O(n^{2.99})$ , the first case applies and we conclude  $T(n) = \Theta(n^3)$ .
- c.  $T(n) = 16T(n/2) + (n \log n)^4$ . In the Master Theorem, we have  $a = 16$ ,  $b = 2$ , and  $f(n) = n^4(\log n)^4$ . Then  $\log_b a = 4$ . We cannot apply Master's Theorem! The recursion tree methods (only sketched here) gives:

1.  $(n \log n)^4$  on the root level (level 0)
2.  $16(n/2 \log(n/2))^4 = (n(\log n - 1))^4$  on level 1
3.  $16^2(n/4 \log(n/4))^4 = (n(\log n - 2))^4$  on level 2
4.  $16^j(n/(2^j) \log(n/(2^j)))^4 = (n(\log n - j))^4$  on level j

with the last level being  $\log_2 n$ . Summing up all the terms gives  $T(n) = O(n^4(\log n)^5)$ . Summing up the first half of terms gives  $T(n) \geq \frac{\log n}{2}(\frac{1}{2}n \log n)^4 = \Omega(n^4(\log n)^5)$ . Thus  $T(n) = \Theta(n^4(\log n)^5)$ .

- d.  $T(n) = 7T(n/3) + n$ . In the Master Theorem, we have  $a = 7$ ,  $b = 3$ , and  $f(n) = n$ . Then  $\log_b a = \log_3 7 > 1.2$ . As  $f(n) = O(n^{1.2})$ , the first case applies and we conclude  $T(n) = \Theta(n^{\log_3 7})$ .
- e.  $T(n) = 9T(n/3) + n^3 \log n$ . In the Master Theorem, we have  $a = 9$ ,  $b = 3$ , and  $f(n) = n^3 \log n$ . Then  $\log_b a = 2$ . As  $f(n) = \Omega(n^{2.01})$ , the third case applies and we conclude  $T(n) = \Theta(n^3 \log n)$ .

**Problem 4** Describe an efficient greedy algorithm for making change for a specified value using a minimum number of coins, assuming there are four denominations of coins (called quarters, dimes, nickels, and pennies), with values 25, 10, 5, and 1, respectively. Argue why your algorithm is correct.

**Solution Sketch:**

The greedy algorithm does the following: as long as the change is non-zero, use one coin of the largest possible denomination (thus subtracting the value of the coin from the change), and then repeat. To implement this fast, use integer division to compute the number of coins of the largest possible denomination. The running time is then at most a constant the number of denominations.

If the values of the coins are 25, 10, 5, 1, the greedy algorithm produces the change with the minimum number of coins, based on the following argument:

If a solution uses more than 4 pennies, then it is not optimum, as 5 pennies should be replaced by a nickel. If a solution uses more than 1 nickel, then it is not optimum, as 2 nickels should be replaced by one dime. If a solution uses more than 2 dimes, then it is not optimum, as 3 dimes should be replaced by a quarter and a nickel. Also, if the solution uses 2 dimes and a nickel, then it is not optimum, as a quarter should be used instead.

This shows that the optimum solution will never use pennies if it could use nickels, it never uses nickels when it could use dimes, and it never uses dimes when it could use quarters. In conclusion, the optimum solution is obtained exactly by the greedy algorithm.

**Problem 5** Give an example set of denominations of coins so that a greedy change making algorithm will not use the minimum number of coins. Give an instance, show the output of the greedy algorithm on this instance, and show better output.

**Solution:**

If the denominations are 1, 6, and 10, and the change value is 18, the greedy algorithm produces the solution: 1 coin of 10c, 1 coin of 6c, and 2 coins of 1c (total: 4 coins), while a solution with only 3 coins exist: 3 coins of 6c.