Purdue University | Hambrusch
CS 381 | Spring 2015

Assignment 3

Due: Tuesday, February 17, 2015 (hand in before class)

1) (20 pts.) Review the divide and conquer algorithms discussed in class (see slides posted on 1/29/15). For problems 1-3 consider the $O(n)$ time divide-and-conquer algorithm finding the maximum sum subarray. For problem 4 consider the $O(n \log n)$ time divide-and-conquer algorithm counting inversions.

1. Give an example of an array A_{16} of size 16 for which the entries returned from the two calls to the subarrays of size 8, $A_{8,L}$ and $A_{8,R}$, use the maximum sum entries anchored at the right end of $A_{8,L}$ and the left end of $A_{8,R}$ to compute the maximum subarray in A_{16}. For your example, give the values of the four entries returned by the call on A_{16}. Explain your answers.

2. Give an example of an array A_{16} of size 16 for which the entries returned from the two calls to the subarrays of size 8, $A_{8,L}$ and $A_{8,R}$, use the sum entry (sum of all entries) of subarray $A_{8,L}$ for the answers returned by the call on A_{16}. For your example, explain your answers.

3. Consider the *maximum product subarray problem*. The given array can contain positive and negative entries and the problem is to determine the two indices such that the product of the entries in the induced subarray is a maximum. Modify the $O(n)$ time divide-and-conquer algorithm finding the maximum sum subarray to solve the maximum product subarray problem. Clearly state and explain the entries a recursive call returns and how they are used.

4. Consider the permutation 1 2 4 6 14 3 9 12 7 8 11 10 5 13 16 15 stored in array A_{16}. Let $A_{8,L}$ and $A_{8,R}$ be its left and right subarray. State the entries returned by the recursive call on $A_{8,L}$ and $A_{8,R}$ and describe how they are used to determine the number of inversions for A_{16}.

2) (15 pts.) Tired of Mergesort and Heapsort as popular $O(n \log n)$ time sorting algorithms, Donald decides it is time for a new fast sorting algorithm. He comes up with the following recursive algorithm which he calls D-Sort. He believes it is fast and correct, but can't convince anyone. You are asked to help. D-sort works as follows when sorting array A of size n:

if $n = 2$ **then**
    sort the two elements by making one comparison
else
    $m = \lceil 2n/3 \rceil$
    D-Sort(A, 0, m-1)
    D-Sort(A, n-m, n-1)
    D-Sort(A, 0, m-1)
end if

1. Does D-sort sort an array of size n correctly? If it does not, give a counterexample. If it does, argue its correctness as to why D-sort correctly sorts any input sequence.

2. State the recurrence relations capturing the running time and give the associated performance in terms of n. Either use the Master Theorem (if it can be used) or "guess" the time bound using the recursion tree and make a proof by induction.

3) (15 pts.) You are supposed to sort n elements in a computational environment that does not allow you to make comparisons. Instead, you are supposed to use a magical sorting box which takes $\sqrt{n}$ elements as input and outputs them in sorted order.

Design and describe an efficient sorting algorithm using the sorting box. Express the performance of your algorithm in terms of the number of sorting box calls. Your algorithm can move and copy data and maintain indices, but it cannot compare two elements. The only way to make comparisons is through the use of the sorting box. Explain your algorithm, argue its correctness, and analyze the number of sorting box uses in terms of n.