Home Work 3 : Divide and Conquer

Deadline Thursday Nov 26, 2013, 6 PM

P-1 [Marks 20]: What do you understand by the divide and conquer strategy? Does there exist a recursive algorithm (code) that does not fall in divide and conquer strategy?

P-2 [Marks 50]: We are talking about a competition among various "divide and conquer" experts here. The names of candidates are as following:

1. Dr. Log
2. Dr. Linear
3. Dr. Exponential
4. Dr. Constant
5. Dr. Polynomial

All the experts are given the same problem to solve. The winner solves the problem in the best asymptotic running time. Following are the divide and conquer strategies followed by the competitors for the problem:

1. Dr. Log divides the problem into 6 sub problems each of size n/7.
   The total combine cost is "lgn" for Dr. Log's algorithm.
2. Dr. Linear divides the problem into 3 sub problems of different sizes n/4, n/2 and n/8.
   The total combine cost is "n" for Dr. Linear's algorithm.
3. Dr. Exponential divides the problem into 2 sub problems each of size n/2.
   The total combine cost is "2^n" for Dr. Exponential's algorithm.
4. Dr. Constant divides the problem into 10 sub problems each of size n/3.
   The total combine cost is "1" for Dr. Constant's algorithm.
5. Dr. Polynomial divides the problem into 1 sub problem size n/5.
   The total combine cost is "n^3" for Dr. Polynomial's algorithm.

Assuming all the algorithms are correct, find out the winner(s).

P-3[Marks 40]: Let $A[1 \cdot\cdot n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an ***inversion*** of A.

a. List the five inversions of the array $\langle 2, 3, 8, 6, 1 \rangle$.
b. What array with elements from the set $\{1, 2, \dots, n\}$ has the most inversions? How many does it have?
c. What is the relationship between the running time of insertion sort and the number of inversions in the input array? Justify your answer.
d. Give an algorithm that determines the number of inversions in any permutation on n elements in $\Theta(n \lg n)$ worst-case time. (*Hint:* Modify merge sort.)

P-4[Marks 50]: Professors Howard, Fine, and Howard have proposed the following "elegant" sorting algorithm:

```
STOOGE-SORT(A, i, j)
if A[i] > A[j]
   then exchange A[i] ↔ A[j]
if i + 1 ≥ j
   then return
k ← ⌊(j - i + 1)/3⌋                ▹ Round down.
STOOGE-SORT(A, i, j - k)            ▹ First two-thirds.
STOOGE-SORT(A, i + k, j)            ▹ Last two-thirds.
STOOGE-SORT(A, i, j - k)            ▹ First two-thirds again.
```

NOTE (There is no return value, all the updates are incorporated in the input array. So when code terminates, the input Array A is sorted)

a. Dry-Run the above code on the following array:

2	-2	0	49	45	7	6	5	3	1	2	1	-1	0	50	89	76	54	25	44

b. Give a recurrence for the worst-case running time of STOOGE-SORT and a tight asymptotic (Θ-notation) bound on the worst-case running time.

c. Compare the worst-case running time of STOOGE-SORT with that of insertion sort and merge sort.