Divide & Conquer Group Exercises

Adapted from material by Cole Frederick

Please work the following problems in groups of 2 or 3. Use additional paper as needed, and staple the sheets together before turning them in. ONLY TURN IN 1 WORKSHEET/ANSWERS PER TEAM.

1. The stages of a divide and conquer strategy are:

____________________________________________________________________________________
____________________________________________________________________________________
____________________________________________________________________________________

2. Recursion is:

____________________________________________________________________________________
____________________________________________________________________________________

3. What are the recurrences for:
   (a) Merge sort

   (b) Binary search

   (c) Build heap
Maximum subarray problem

Ex: maximize profit for buying a stock on one day and selling it another. We know about the future, and the data we have is the date and the price of the stock on that date.

4. Brute force method – look at how much we could make between every pair of dates and pick the pair that gives the maximum profit:
   
   Complexity: ____________________________

5. Transform the list of daily prices to a list of daily changes in price. Then we can look for (choose one):
   
   - any subarray and take its sum
   - the subarray with the maximum sum
   - the subarray with maximum length

Divide and Conquer Strategy for this problem:

6. Divide the array into ______________________ chunk. (How many chunks?)

7. A maximum subarray is either contained completely in __________________________, completely in __________________________, or __________________________

8. We can use __________________________ to find it in the first 2 cases. (a greedy strategy, D&C, recursion, intervals, backtracking, stable matching)

9. For the third case, we can compute the maximum subarray in linear time by finding the starting and ending positions, of such a maximum subarray. What two elements are (by definition) included in this subarray?
   __________________________
Analysis:

Answer the questions about the pseudo-code that follows.

10. What is the complexity of the FIND-MAX-CROSSING-SUBARRAY procedure?

11. What is the recurrence that describes the FIND-MAXIMUM-SUBARRAY procedure? What is its solution? You can assume that the problem size $n$ is a power of 2 so that all sub-problem sizes are integers.

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
1 if high == low
2    return (low, high, A[low])
3 else mid = ⌊(low + high) / 2⌋
4    (left-low, left-high, left-sum) =
5        FIND-MAXIMUM-SUBARRAY(A, low, mid)
6    (right-low, right-high, right-sum) =
7        FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
8    (cross-low, cross-high, cross-sum) =
9        FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
10   if left-sum ≥ right-sum and left-sum ≥ cross-sum
11      return (left-low, left-high, left-sum)
12   elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
13      return (right-low, right-high, right-sum)
14   else return (cross-low, cross-high, cross-sum)
```

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
1 left-sum = -∞
2 sum = 0
3 for i = mid downto low
4    sum = sum + A[i]
5    if sum > left-sum
6       left-sum = sum
7       max-left = i
8 right-sum = -∞
9 sum = 0
10 for j = mid + 1 to high
11    sum = sum + A[j]
12    if sum > right-sum
13       right-sum = sum
14       max-right = j
15 return (max-left, max-right, left-sum + right-sum)
```
**Due at end of class:** Friday, Sep 15, 2017

**CS320 – Worksheet 4**

**Observations on sub-problem sizes:**

We’ve mostly dealt with sub-problems that are around half of the size of the bigger problems. Let’s see what happens if we divide them very differently.

Assume we’re working on an algorithm to find the biggest integer in a Python list. We decide to create sub-problems by taking the first element as one sub-problem, and all the rest as the second sub-problem:

MaximumNum is the maximum of L[0] and MaximumNum(L[1:]) (We use Python slicing here for the rest of the array)

12. What is the recurrence if we create sub-problems this way? Hint: the first split gives us $T(n-1) + O(1)$

13. What is the solution of this recurrence?

Now assume we decide to try taking chunks of 3 numbers. So MaximumNum is the maximum of L[0:3] and MaximumNum(L[3:])

14. What is the recurrence if we create sub-problems this way? Hint: you have to figure out the maximum of each group of 3 numbers, assume using 2 comparisons, so the first split gives us $T(n-3) + O(2)$

15. What is the solution of this recurrence?

16. What is the recurrence if we decide to create sub-problems from $c$ numbers? (The first sub-problems are L[0:c] and L[c:])
17. Now that we have explored the effect of changing the $c$ parameter as part of splitting problems into $[c, n-c]$, let's see why merge sort (and binary search, arraylist growth, etc) does not split problems by subtracting constants. Write the recurrence equation for merge sort if it splits problems into $[c, n-c]$ at each level of recursion. Find the complexity bound (using the master theorem). Does changing the constant $c$ affect the Big-O complexity?