Assignment 4: Divide and Conquer

SCIS 313: Data Structures and Algorithm Analysis

Prof. Antonio Khalil Moretti
Spelman College

Due: 3/16

Instructions

This assignment covers divide and conquer algorithms, recurrence relations, and the Master Method. For full credit, **show all work**.

Conventions. Throughout, $T(1) = c$ for some constant $c > 0$. When applying the Master Method, explicitly identify a, b, d, compute b^d, state which case applies, and give the final bound.

Problem 1 (8 points)

Recognizing the recurrence.

For each algorithm sketch below, write the recurrence relation $T(n)$. Do *not* solve it yet — just set it up correctly.

(a) An algorithm that splits an array of size n into two equal halves, recurses on *both* halves, and then spends $O(n)$ time merging the results.

(b) An algorithm that splits an array of size n into two equal halves, recurses on *one* half only (discarding the other), and does $O(1)$ work outside the recursive call.

(c) An algorithm that splits its input into *three* equal thirds, recurses on all three, and combines results in $O(n^2)$ time.

(d) The following code:

```
int count(int n) {
    if (n <= 1) return 1;
    return count(n/4) + count(n/4) + n;
}
```

Problem 2 (10 points)

Solving by unrolling.

Solve the following recurrences by unrolling (repeated substitution). Show at least three steps of substitution, identify the pattern, and derive the closed form. Do *not* use the Master Method for this problem.

(a) $T(n) = T(n-1) + n, \quad T(1) = 1$

(b) $T(n) = T(n/4) + 1, \quad T(1) = c$

(c) $T(n) = 3T(n/3) + c, \quad T(1) = c$

Hint: How many levels does the tree have? How many nodes at level k?

Problem 3 (12 points)

Applying the Master Method.

For each recurrence, apply the Master Method. Identify a, b, d; compute b^d; state the case; and give the Θ bound. If the Master Method does not apply, say so and explain why.

(a) $T(n) = 5T(n/2) + O(n^2)$

(b) $T(n) = 8T(n/2) + O(n^3)$

(c) $T(n) = T(n/2) + O(n)$

(d) $T(n) = 4T(n/3) + O(n)$

(e) $T(n) = 2T(n/2) + O(n \log n)$

Note: Is $O(n \log n)$ of the form $O(n^d)$ for some constant d? What does that mean for the Master Method?

(f) $T(n) = T(n-1) + O(n^2)$

Problem 4 (10 points)

From code to recurrence to solution.

Analyze the following recursive function completely: identify the recurrence, apply the Master Method, and give the time complexity.

```
1  int mystery(int[] arr, int lo, int hi) {
2      if (lo >= hi) return arr[lo];          // base case: O(1)
3
4      int third = (hi - lo) / 3;
5      int a = mystery(arr, lo,           lo + third);    // recurse left
           third
6      int b = mystery(arr, lo + third,  lo + 2*third);  // recurse middle
            third
7      int c = mystery(arr, lo + 2*third, hi);           // recurse right
           third
8
9      // linear scan to combine
10     int result = 0;
11     for (int i = lo; i <= hi; i++) result += arr[i];  // O(n)
12     return result + a + b + c;
13 }
```

(a) What are a, b, and d?

(b) Write the recurrence.

(c) Apply the Master Method. What is the time complexity?

(d) Does this algorithm do anything useful? Could you compute the same result more efficiently? If so, how, and what would the complexity be?

Problem 5 (10 points)

Recursion tree.

Draw the full recursion tree for $T(n) = 2T(n/3) + cn$, assuming $n = 27$.

(a) Draw the tree, labelling each node with the work done at that node (not the subproblem size — the actual work cn, $c(n/3)$, etc.).

(b) Fill in the table below.

Level k	Subproblem size	Nodes at level k	Work at level k
0	n		
1			
2			
k			

(c) How many levels does the tree have? (Express in terms of n.)

(d) Sum the work across all levels. What is $T(n)$? Verify your answer using the Master Method.

Problem 6 (10 points)

Binary search correctness and complexity.

Consider the following implementation of binary search.

```
1 int binarySearch(int arr[], int lo, int hi, int target) {
2     if (lo > hi) return -1;
3     int mid = lo + (hi - lo) / 2;       // (A)
4     if (arr[mid] == target) return mid;
5     if (arr[mid] > target)
6         return binarySearch(arr, lo, mid - 1, target);
7     else
8         return binarySearch(arr, mid + 1, hi, target);
9 }
```

(a) Line `(A)` computes `mid` as `lo + (hi - lo) / 2` rather than the simpler `(lo + hi) / 2`. Both compute the midpoint. Why might a programmer prefer the version shown? *Hint: think about the range of* ***`int`****.*

(b) Write the recurrence for the worst-case number of comparisons. Solve it by unrolling.

(c) Trace the execution of `binarySearch` on the array

`[3, 7, 11, 15, 22, 31, 40, 55, 68, 72]`

with `target = 31`. List the value of `mid` and the comparison made at each recursive call.

(d) What is the maximum number of comparisons needed to search this 10-element array? Give the exact value and justify it.

Problem 7 (10 points)

Karatsuba multiplication.

(a) Use the Karatsuba algorithm to compute 4321×8765. Set $a = 43$, $b = 21$, $c = 87$, $d = 65$. Compute $P_1 = ac$, $P_2 = bd$, $P_3 = (a + b)(c + d)$, and the cross term $P_3 - P_1 - P_2$. Then combine with appropriate powers of 10. Show every step.

(b) The naive recursive algorithm for n-digit integer multiplication uses 4 recursive calls on $n/2$-digit numbers and does $O(n)$ work outside. Write its recurrence and apply the Master Method.

(c) Write the Karatsuba recurrence and apply the Master Method. What is the exponent, and why is it strictly less than 2?

(d) Suppose someone claims a variant of Karatsuba that uses only *two* recursive calls on $n/2$-digit numbers (with $O(n)$ extra work). What would the running time be? Is this plausible? *(No proof required — reason informally.)*

Problem 8 (10 points)

Design your own divide and conquer algorithm.

You are given a sorted array $A[0 \ldots n-1]$ of distinct integers and a target value t. You want to determine whether there exist indices $i < j$ such that $A[i] + A[j] = t$ (a "two-sum" query on a sorted array).

(a) Describe a *brute-force* algorithm for this problem. What is its time complexity?

(b) Now describe a divide and conquer algorithm.

- How do you divide the problem?
- What do you recurse on?
- How do you combine?

Write pseudocode or a clear description.

(c) Write the recurrence for your algorithm and solve it.

(d) There is actually an $O(n)$ algorithm for this problem on a sorted array that uses *no* recursion. Can you find it? *Hint: use two pointers, one starting at each end of the array.* Describe it briefly and argue why it is correct.

Submission Guidelines

- Submit a single PDF to Canvas by the due date.
- Handwritten work is acceptable if it is legible; typed work is preferred.
- For each Master Method application, the five-step structure (a, b, $d \to$ compute $b^d \to$ compare $\to$ case $\to$ result) must be explicit.
- Partial credit is available for correct reasoning with an arithmetic error.