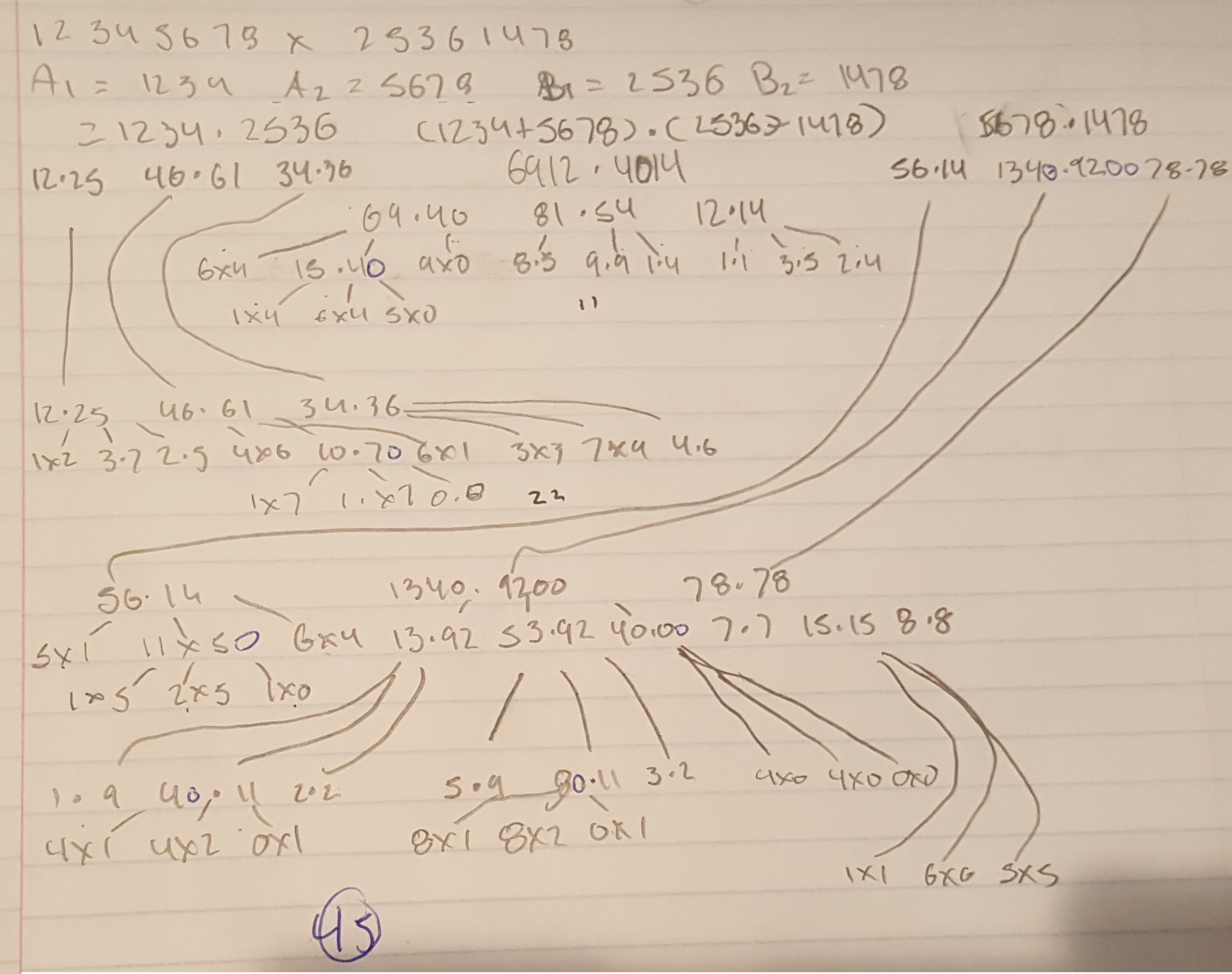
Aaron McCarthy Homework 3

1. Apply the divide-and-conquer technique you learned from class to compute 12345678 ×25361478 .

a) How many 1-digit multiplications are required?

45

b) Justify your answer.



2. The Strassen’s algorithm is only applied to multiply two n× n matrices where n is an exact power of 2, i.e., k n = 2 where k is a non-zero positive integer.

a) Modify the Strassen’s algorithm so that it can compute the multiplication of any two n× n matrices where n is not an exact power of 2? [1pt]

Increase the size of the matrices by padding the edges with zeroes until the size of both matrices is nxn where n is a power of 2

b) Show that your algorithm still runs in Θ (n^log7). [2pts]

M1 = (5 + 0) \* (2 + 0) = 10

M2 = (3 + 0) \* (2) = 6

M3 = 5 \* (6 - 0) = 35

M4 = 0 \* (0 - 2) = 0

M5 = (5 + 3) \* 6 = 48

M6 = (3 - 5) \* (2 + 6) = -24

M7 = (0 - 0) \* (0 + 0) = 0

n = 2

# of multiplications = 7

Θ

3. Binary search is a very efficient algorithm for searching in a sorted array. Here is the pseudo code for finding element K in an array A of size n. m is the middle index.

Is this a divide and conquer algorithm? Justify.

No. The algorithm starts by dividing the array into half and searching on the half where K is. This is not divide and conquer due to the fact that the algorithm is not dividing the problem into 2 or more smaller instances. The algorithm only searches on one side of the midpoint of the array, meaning that there is still only a single instance of the problem. Also the algorithm does not obtain a solution to the original problem by combining the solutions of the subproblems. Instead the solution to the original problem is the solution to a singular subproblem.

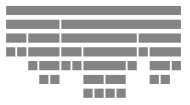
4. What are best, average and worst case time complexities of quick sort for inputs of size n? Justify each case with a figure and a brief explanation.

Best case: all pivots split their respective sub arrays in the middle



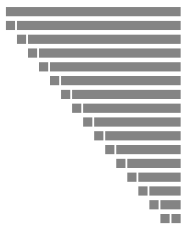
This is the best case because every branch of the recursion terminates in the same spot. The algorithm does not have to move any values after selecting the pivot. All it has to do is compare the value to the pivot and then move on to the next one. This means that the time complexity in O(nlogn). This is because it has to do one operation on every member of the array (compare it to the pivot) and it has to do this for each level of recursion. Since one operation on each member of the array is n operations and there are logn levels of recursion the time complexity is O(nlogn)

Average case: random array



This is the average case because this is the case that will be encountered most often. The time complexity of this is also O(nlogn). In this case the algorithm will still be comparing n values to the pivot about logn times. This is due to the fact that even though the pivots will be placed randomly in the array the extra work being done by the algorithm is linear and not enough to make the time significantly greater than the best case when approaching very large n’s.

Worst case: sorted array



The sorted array is the worst case because of how the pivot is chosen. Because the first element in the array is chosen as the pivot and, in the case of a sorted array, the first element is the smallest, this means that every time the array will be split into 2 arrays, one with the pivot and one with everything else. This is sub optimal due to the fact that the strength of quick sort is splitting the array into 2 pieces so it can work on those pieces separately. In the case of the sorted array the algorithm has to compare n values in the array to the pivot and this has to be done n times. Therefore, the time complexity is O(n2).