CSE 551: Quiz 4 Solutions

Jamison Weber

September 24, 2021

Note that some of the solutions provided in this document may apply to questions that did not appear in your particular quiz.

1 Problem 1

Solve the following recurrence relation using any method. Provide your answer in big-O notation: $T(n) = 2T(\frac{n}{2}) + log(n) \quad \text{for } n > 1, 1 \text{ otherwise}$

- [✓ $T(n) = O(n)$]
- $T(n) = O(nlogn)$
- $T(n) = O(n^2)$
- $T(n) = O(logn)$

1.1 Rationale

This recurrence relation can be tricky to solve using iterative substitution or tree-based methods. It's best to use the Master theorem here. Recall the form:

$$T(n) = aT(n/b) + f(n)$$

Since $f(n) = log(n) = n^\epsilon$ where $\epsilon < log_2(2) = 1$, the Master method gives: $T(n) = O(n^{log_2(2)}) = O(n)$

2 Problem 2

Suppose we have a modified version of Merge-Sort that at each recursive level splits the array into two parts each of size $\frac{1}{4}$ and $\frac{3}{4}$ respectively. Also, assume the size of any given input array is a power of four. Give the asymptotic time complexity of this Merge-Sort variant.

- [✓ $O(nlogn)$]
- $O(n)$
- $O(logn)$
- $O(n^2)$

2.1 Rationale

This gives the recurrence relation $T(n) = T(\frac{3}{4}) + T(\frac{1}{4}) + cn$. Using the Akra-Bazzi method: We must satisfy

$$\sum_{i=1}^{k} a_i b_i^P = 1$$

Where a_i is the coefficent in front of recursive call i and b_i is the divisor of n in recursive call i. Thus, we have:

$$1(\frac{3}{4}) + 1(\frac{1}{4})^P = 1 \Rightarrow P = 1$$

Now we use the Akra-Bazzi formula to find:

$$T(n) \in \Theta(n^P(1 + \int_1^n \frac{g(u)}{u^{P+1}} du)) = \Theta(n(1 + \int_1^n \frac{cn}{n^2} dn))$$

$$= \Theta(n(1 + c\int_1^n \frac{1}{n} dn)) = \Theta(n + ncln(n)) = \Theta(nlog(n))$$

3 Problem 3

Suppose we modify the combine step of the closest pair of points algorithm such that distance δ from dividing line L is updated immediately to δ' whenever the distance between two points on either side of L is discovered to be less than δ. In this sense, we allow the two-dimensional range about L

wherein we compare points to assume multiple areas (getting smaller as δ becomes updated) during the same combine step for each recursive call. Determine whether the following statement is true or false and explain your reasoning: The time complexity of the closest pair of points algorithm is guaranteed to be improved by a constant factor.

- **False: If δ is reduced only after the last pair of points sorted by y-coordinate within δ of L is compared, then no additional benefit will be gained.**
- False: The number of comparisons made between points within δ of L would necessarily be the same as without the modification.
- True: If δ is reduced every time a new closest pair of points is found during the combine step, then it will always be the case that a fewer number of future comparisons will be necessary after δ is adjusted toδ'.
- True: If δ is reduced every time a new closest pair of points is found during the combine step, then it will sometimes be the case that a fewer number of future comparisons will be necessary after δ is adjusted to δ'.

3.1 Rationale

It is certainly possible that δ may only be reduced during the final comparison across the line L, which would not yield any improvements in efficiency.

4 Problem 4

Suppose there is a set of n points each with the same x-coordinate. Obviously, this will cause the closest pair of points algorithm to fail. Since this is the case, what must be the smallest possible asymptotic time complexity required to find the closest pair of points in this instance?

- $O(nlogn)$
- $O(nlog^2n)$
- $O(n^2)$
- $O(n)$

4.1 Rationale

Sort each point by y-coordinate and then run a linear scan over the sorted array storing the distance between every two consecutive points in memory. Return the smallest distance. The total run time would then be dominated by sorting, which has a lower bound of $\Theta(nlogn)$.

5 Problem 5

Given a two-dimensional plane with points (0,1),(1.5,2),(1,4),(4.8,2),(5,3), (7,3.5),(8,8),(7.5,9.5), give the numerical value of δ before the combine step on the highest level of the recursive stack.

- 1.58
- 1.80
- 1.02
- 2.06

5.1 Rationale

This problem requires a few calculations. Divide the points into two sets where every point with x-coordinate less than or equal to 4.8 is in one group and every point with x-coordinate greater than 4.8 is in the other. Use the distance formula to find the distances between the closest pair of points in group.

$$\delta_{left} = \sqrt{(1.5-0)^2 + (2-1)^2} = 1.80$$

$$\delta_{right} = \sqrt{(8-7.5)^2 + (9.5-8)^2} = 1.58$$

Return 1.58. Since we are returning δ before the combine step, we do not need to compare points across the dividing line.

6 Problem 6

Given a two-dimensional plane with points (1.5,1), (2,3.8), (2.75,1), (5,3.3), (6,3.5), (7.5,2), (8.5,1.2), (8,5), give the numerical value of δ after the combine step on the highest level of the recursive stack.

- 1.02

- 0.89
- 1.28
- 1.25

6.1 Rationale

It is the same process as before, except we will calculate the δ value for each group as well as across the line and return the minimum of the three.

$$\delta_{left} = \sqrt{(2.75 - 1.5)^2 + (1 - 1)^2} = 1.25$$

$$\delta_{middle-min} = \sqrt{(6 - 5)^2 + (3.5 - 3.3)^2} = 1.02$$

$$\delta_{right} = \sqrt{(8.5 - 7.5)^2 + (1.2 - 2)^2} = 1.28$$

Return 1.02

7 Problem 7

Identify which of the following statements are true regarding the Karatsuba multiplication algorithm.

i.) The Karatsuba algorithm is asymptotically less complex than merge-sort with respect to the number of operations it requires to terminate.

ii.) Each call to this algorithm produces three additional recursive calls on roughly half the number of bits as the previous call.

- [✓ ii. only]
- i. only
- i. and ii.
- Neither of these are true

7.1 Rationale

Statement i is false since $O(nlogn)$ is of a lower order than $O(n^{log_2(3)})$. Statement ii is true since the recurrence relation for Karatsuba's algorithm is given by a recurrence relation similar to:

$$T(n) \leq 3T(n/2) + cn$$

8 Problem 8

Suppose Anatolii Karatsuba had somehow discovered a way to produce the product of integers x and y with only two real multiplications and an asymptotically linear number of adding, subtracting and shifting operations per recursive call to his algorithm. What would have been the asymptotic time complexity of his algorithm if x and y are each represented by n bits? (Hint: Read pages 215 to 218 in the Kleinberg and Tardos textbook carefully before answering.)

- [✓ $O(nlogn)$]
- $O(n^{1.585})$
- $O(n)$
- $O(n^2)$

8.1 Rationale

According to the supplementary reading, q represents the number of subproblems created from each recursive call. In Karatsuba's algorithm, there are three subproblems created with each recursive call, so q=3. If we reduce the number of subproblems to 2 per recursive call, then we know that the run time of the algorithm will be the same as merge-sort, which is $O(nlogn)$.

9 Problem 9

Identify which of the following statements are true regarding Strassens fast matrix multiplication algorithm as applied to matrices of size n by n.

i.) Strassens algorithm is the fastest known algorithm for matrix multiplication.

ii.) If n is not a power of 2, adding rows and columns of zero-entries until n is a power of 2 will allow the algorithm to run normally and produce the correct result.

- [✓ ii. only]
- i. only

- i. and ii.
- Neither of these are true

9.1 Rationale

Statement i is false. Under certain conditions, this run time can be improved as seen with the decimal wars. Statement ii is true, and this method is precisely how matrix multiplication is computed when n is not a power of 2.

10 Problem 10

Suppose Volker Strassen had somehow discovered a way to produce the product of two n by n matrices with only six real multiplications and an asymptotically quadratic number of addition and subtraction operations per recursive call to his algorithm. What would have been the asymptotic time complexity of his algorithm? (Hint: Read pages 215 to 218 in the Kleinberg and Tardos textbook carefully before answering.)

- [✓ $O(n^{2.585})$]
- $O(n^{2.810})$
- $O(n^2 logn)$
- $O(n^{2.322})$

10.1 Rationale

The recurrence relation given by this algorithm will have the form:

$$T(n) = 6T(n/2) + cn^2$$

which does not fit the form $T(n) = qT(n/2) + cn$, so we cannot use the result from the reading directly, but we can use the ideas from the proof in the document provided to determine the solution ourselves. We make 6 recursive calls each of size $n/2$ per recursive level. In general, at the j^{th} level, we make 6^j calls of $c(\frac{n}{2^j})^2$ work each for a total of $6^j c(\frac{n}{2^j})^2 = 6^j c\frac{n^2}{2^{2j}} = (2 \times 3)^j c\frac{n^2}{2^{2j}} = (\frac{3}{2})^j cn^2$ work at level j. We sum over all $\log n$ levels of the recursive tree to obtain:

$$T(n) = cn^2 \sum_{j=0}^{\log n - 1} (\frac{3}{2})^j = cn^2 \frac{1 - (\frac{3}{2})^{\log n}}{1 - \frac{3}{2}} = cn^2 \frac{1 - n^{\log \frac{3}{2}}}{1 - \frac{3}{2}} = \mathcal{O}(n^2 n^{.585}) = \mathcal{O}(n^{2.585})$$

Problem 11

We propose the following divide-and-conquer algorithm for solving the minimum spanning tree problem for a connected, weighted graph $G = (V, E)$. Recursively, we divide the set of vertices in roughly half to obtain vertex sets V_1, V_2. We recursively compute the minimum spanning trees T_1, T_2 over V_1 and V_2, respectively. To combine the subproblems, we add the smallest edge such that T_1, T_2 now form a single component. What is the primary difficulty of achieving an efficient implementation under this framework?

- The divide step: Partitioning the vertices in two equal-sized components such that each component is connected may require considering all vertex subsets of size $n/2$.
- The combine step: searching for an edge of minimal weight to join the two trees will require exponential time
- The conquer step: there may be an exponential number of recursive calls per level.
- The base case of the recursion will require traversing a recursive depth exponential in the size of V.

Rationale

Recall that the subproblems must be small versions of the original problem, and thus in this case it really matters how you divide the vertex set. For example, if one of the sub problems contains vertices that don't form a single component on there own, then no spanning tree exists for this subset of vertices. In other words, the sub problem must preserve the properties of the larger instance, namely that the subgraph is connected. Therefore, we have to determine a way to divide the graph into exactly two components before computing the solution to the two subproblems. Thus we may have to consider $\binom{n}{n/2}$ vertex subsets to find two connected components. It is not clear whether this can be done efficiently, and thus gives the most difficulty. The remaining answer choices are false.

Problem 12

Consider the following divide-and-conquer algorithm for checking to see whether an element e exists in a pre-sorted array A. Divide A into two roughly equal sized subarrays A_1, A_2 at element e'. If $e \leq e'$ Recursively check to see if e is in A_1. Otherwise, recursively check to see if e is in A_2. Return true if so, and false otherwise. The base case considers an array of size one, where it is trivial to check whether e is the singleton element. Give the recurrence relation that describes the algorithm above in terms of the number n of elements in A. Assume $T(n) = 1$ when $n = 1$.

- $T(n) = T(n/2) + O(1)$
- $T(n) = 2T(n/2) + O(1)$
- $T(n) = 2T(n/2) + O(n)$
- $T(n) = T(n/2) + O(n)$

Rationale

At each recursive level, we find the middle point of the array, which requires $O(1)$ time. We then compare the middle element to e. If the middle element is at most e, then we only need consider the first half of the array. Otherwise, we consider only the second half. Thus we make only one recursive call at each level on a subproblem of size $n/2$. Since the divide step requires constant time, we get the highlighted recurrence relation.

Problem 13

Consider the following divide-and-conquer algorithm used to find the maximum element of an unsorted array A: Divide A into two roughly equal parts A_1 and A_2. Recursively find the maximum elements e_1, e_2 of their respective subarrays. Return maximum of e_1, e_2. Identify the asymptotic running time of the combine step in terms of the number n of elements in A.

- $\Theta(1)$
- $\Theta(n)$
- $\Theta(n \log n)$
- $\Theta(\log n)$

Rationale

The combine step is the process of computing the maximum between the maximum of the left array and the maximum of the right arrow. Of course, computing the maximum of two elements can be done in constant time.