CSC421 Analysis of Algorithms Assignment 2

Matt Triano

April 19, 2017

1 Problem 1) Merge Sort

1.1 Problem Statement

Illustrate the execution of Merge Sort on the array $A = \langle 6, 4, 9, 8, 5, 10, 1, 3 \rangle$.

1.2 My Illustration of Merge Sort Execution

In retrospect, I should have chosen a more easily implementable way of traversing the binary tree of splits. I went with an iteratively deepening depth-first traversal in my by-hand calculation.

Iteration	Left List	Right List	Action
1	$[6, 4, 9, 8]$	$[5, 10, 1, 3]$	Recursion level 1: Initial split
2	$\left[[6, 4], [9, 8]\right]$	$[5, 10, 1, 3]$	Recursion level 2: left half split
3	$\left[[6], [4]\right], [9, 8]$	$\left[5, 10, 1, 3\right]$	Recursion level 3: left half split 1
4	$\left[[6], [4]\right], \left[[9], [8]\right]$	$\left[5, 10, 1, 3\right]$	Recursion level 3: left half split 2
5	$\left[4, 6\right], \left[[9], [8]\right]$	$\left[5, 10, 1, 3\right]$	Recursion level 3: left half merge 1
6	$\left[4, 6\right], \left[8, 9\right]$	$\left[5, 10, 1, 3\right]$	Recursion level 3: left half merge 2
7	$\left[4, 6, 8, 9\right]$	$\left[5, 10, 1, 3\right]$	Recursion level 2: left half merge
8	$\left[4, 6, 8, 9\right]$	$\left[[5, 10], [1, 3]\right]$	Recursion level 2: right half split
9	$\left[4, 6, 8, 9\right]$	$\left[[5], [10]\right], \left[1, 3\right]$	Recursion level 3: right half split 1
10	$\left[4, 6, 8, 9\right]$	$\left[[5], [10]\right], \left[[1], [3]\right]$	Recursion level 3: right half split 2
11	$\left[4, 6, 8, 9\right]$	$\left[5, 10\right], \left[[1], [3]\right]$	Recursion level 3: right half merge 1
12	$\left[4, 6, 8, 9\right]$	$\left[[5, 10], [1, 3]\right]$	Recursion level 3: right half merge 2
13	$[4, 6, 8, 9]$	$[1, 3, 5, 10]$	Recursion level 2: right half merge
14	$[1, 3, 4, 5, 6, 8, 9, 10]$	$[]$	Recursion level 1: Return result

2 Problem 2) Quick Sort

2.1 Problem Statement

Illustrate the execution of Quick Sort on the array $A = \langle 6, 4, 9, 8, 5, 10, 1, 3 \rangle$

2.2 My Illustration of Quick Sort Execution

Recursion Level	Operation	Array after partition or Quick Sort (qs)
0	qs([6, 4, 9, 8, 5, 10, 1, 3])	left:[1], pivot:[3], right:[6, 4, 9, 8, 5, 10]
1	qs([1])	returns [1]
1	qs([6, 4, 9, 8, 5, 10])	left:[6, 4, 9, 8, 5], pivot:[10], right:[]
2	qs([6, 4, 9, 8, 5])	left:[4], pivot:[5], right:[6, 9, 8]
3	qs([4])	returns [4]
3	qs([6, 9, 8])	left:[6], pivot:[8], right:[9]
4	qs([6])	returns [6]
4	qs([9])	returns [9]
3	qs([6, 9, 8])	returns [6, 8, 9]
2	qs([6, 4, 9, 8, 5])	returns [4, 5, 6, 8, 9]
2	qs([])	returns []
2	qs([6, 4, 9, 8, 5, 10])	returns [4, 5, 6, 8, 9, 10]
1	qs([6, 4, 9, 8, 5, 10, 1, 3])	returns [1, 3, 4, 5, 6, 8, 9, 10]
0	qs([6, 4, 9, 8, 5, 10, 1, 3])	Sorts to [1, 3, 4, 5, 6, 8, 9, 10]

3 Problem 3) 2D Coordinate Search

1. if $|S| \leq 3$ then —-return the closest pair (p_{min}, q_{min}) by brute force
2. Using x, compute a vertical line D of equation $x = \ell$ that partition S into 2 subsets S_L and S_R of roughly equal size, such that the points in S_L are on D or to the left of it and the points in S_R are on D or to the right of it
3. Recursively compute a closest pair $p_L, q_L)$ in S_L and a closest pair (p_R, q_R) in S_R.
4. $\delta_L - |p_L q_L|$; $S_R - |p_R q_R|$; $\delta = min\delta_L, \delta_R$
5. Using y, compute the set of points y_{mid} consisting of all the points in S whose x-coordinate satisfies $\ell - \delta \leq x \leq \ell + \delta$, sorted in a non-decreasing order of their y-coordinates;
6. iterate through y_{mid} for each point in y_{mid}, compute it's distance to the next y points in y_{mid} and keep track of the pair (p_{mid}, q_{mid}) of minimum distance overall
7. return the closest pair (p_{mid}, q_{mid}) among all the pairs (p_{mid}, q_{mid}), (p_L, q_L), and (p_R, q_R)

3.1 Problem Statement

Let A[1..n] be an array of points in the plane, where $A[i]$ contains the coordinates (x_i, y_i) of a point p_i, for $i = 1, ..., n$. Give an $O(n \lg(n))$ time algorithm that determines whether any two points in A are identical (that is, have the same x and y coordinates).

For this, we start by sorting all of these pairs by the x coordinate. Merge Sort takes $O(n \lg(n))$ time, so that's still viable. Then, recursively divide the sorted subarrays around the median value of x until the regions only contain 3 points, at which point, brute force is used to compare the coordinates for these points and return the closest pair of points. Search this list of closest-pairs-of-points for any where the distance between these points is 0. This will either return success or failure.

3.1.1 Analysis

Merge Sort takes $O(n \lg(n))$ time, the recursive computation of the left side will take $T(\frac{n}{2})$, as will the right. $T(n)$ is modeled by the recurrence relation below, which goes asymptotically to $O(n \lg(n))$. The brute force search that occurs when sets have been narrowed down to 3 points will operate in constant $O(c)$ time. The division of the sets into the left and the right seems like it could be a constant time

operation if the language has implemented arrays to automatically track length, but if not, then this split would operate in $O(n)$ time. So the operation operates in $O(n \lg(n)) + O(n \lg(n)) + O(n)$ time, which is $O(n \lg(n))$.

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + O(n) & \text{if } n \geq 4 \\ O(1) & \text{if } n \leq 3 \end{cases} \tag{1}$$

4 Problem 4)

4.1 Problem Statement

Textbook, page 1020, exercise 33.1-4. (problem statement copied from Introduction to Algorithms, (Cormen, Leiserson, Rivest, Stein), Third Edition)

Show how to determine in $O(n^2 \lg(n))$ time whether any three points in a set of n points are collinear.

4.2 Solution

For every point, p_o, in the set, use $(O(n \lg(n)))$ merge sort to sort the remaining $n-1$ points by their polar angle relative p_o. Then compare $(O(n))$ the polar angle for p_i and p_{i+1} $\forall i \in$ this set of points. If their polar angles are equal, it will match the polar angle with respect to p_o, which means all 3 points will be colinear. If a pair of p_i and p_{i+1} with equal polar angles isn't found, pop p_o off of the list and then repeat this process, treating another point as the origin point. For every iteration, this process would take time $O(n + n \lg(n))$, and if we had to check all n of the points, we would get $O(n^2 + n^2 \lg(n))$, which is asymptotically equal to $O(n^2 \lg(n))$.

5 Problem 5)

5.1 Problem Statement

Give a recursive version of the algorithm Insertion-Sort (refer to page 18 in the textbook) that works as follows: To sort $A[1..n]$, we sort $A[1..n1]$ recursively and then insert $A[n]$ in its appropriate position. Write a pseudocode for this recursive version of InsertionSort and analyze its running time by giving a recurrence relation and solving it.

5.2 Solution

6 Problem 6)

6.1 Problem Statement

Textbook, pages 39-40, problem 2-1, parts a, b, and c. (problem statement copied from Introduction to Algorithms, (Cormen, Leiserson, Rivest, Stein), Third Edition)

Although merge sort runs in $\Theta(n \lg(n))$ worst-case time and insertion sort runs in $\Theta(n^2)$ worst-case time, the constant factors in insertion sort can make it faster in practice for small problem sizes on many machines. Thus, it makes sense to coarsen the leaves of the recursion by using insertion sort within merge sort when subproblems become sufficiently small. Consider a modification to merge sort in which $\frac{n}{k}$ sublists of length k are sorted using insertion sort and then merged using the standard merging mechanism, where k is a value to be determined.

1. A) Show that insertion sort can sort the $\frac{n}{k}$ sublists, each of length k in $\Theta(nk)$ worst-case time.
2. B) Show how to merge the sublists in $\Theta(n \lg(\frac{n}{k}))$ worst-case time.
3. C) Given that the modified algorithm runs in $\Theta(nk + n \lg(\frac{n}{k}))$ worst-case time, what is the largest value of k as a function of n for which the modified algorithm has the same running time as a standard merge sort, in terms of Θ-notation?

6.2 Solution

6.2.1 A) Insertion Sort Base Case

As insertion sort has an average complexity of $\Theta(k^2)$ for a sublist of k values, if you're sorting $\frac{n}{k}$ sublists, it would take $\Theta(\frac{nk^2}{k}) = \Theta(nk)$ time.

6.2.2 B) Sublist Merge Analysis

We're merging $\frac{n}{k}$ sublists of length k. We'll recursively merge those pairs until we only have 1 list remaining. There would initially be $\frac{n}{2^0 k}$ lists, then $\frac{n}{2k}$, then $\frac{n}{2^2 k}$, then $\frac{n}{2^3 k}$ (and so on) lists remaining to be paired, which indicates there will be $\lg(\frac{n}{k})$ merges, and each merge takes $\Theta(n)$ time. $\Theta(n)$ repeated $\lg(\frac{n}{k})$ times will operate in $\Theta(n \lg(\frac{n}{k}))$ time.

6.2.3 C) Modified Roughed-Up Leaf Merge Sort

We want k such that $\Theta(nk + n \lg(\frac{n}{k})) = \Theta(n \lg(n))$, so k would have to push either nk or $n \lg(\frac{n}{k})$ into the form $\Theta(n \lg(n))$, so if we take $k = \lg(n)$, we get $\Theta(n \lg(n) + n \lg(\frac{n}{\lg(n)}))$, and as $\frac{n}{\lg(n)} \leq n \; \forall \; n \geq 1$, we see the asymptotically dominant term is $n \lg(n)$, and we see that anything larger than $\lg(n)$ would make Θ grow faster than the desired $\Theta(n \lg(n))$. Therefore, $k = \lg(n)$.

7 Problem 7)

7.1 Problem Statement

Textbook, page 1045, problem 33-3 (problem statement copied from Introduction to Algorithms, (Cormen, Leiserson, Rivest, Stein), Third Edition)

A group of n Ghostbusters is battling n ghosts. Each Ghostbuster carries a proton pack, which shoots a stream at a ghost, eradicating it. A stream goes in a straight line and terminates when it hits the ghost. The Ghostbusters decide upon the following strategy. They will pair off with the ghosts, forming n Ghostbuster-ghost pairs, and then simultaneously each Ghostbuster will shoot a stream at his chosen ghost., As we all know, it is *very* dangerous to let streams cross, and so the Ghostbusters must choose pairings for which no streams will cross. Assume that the position of each Ghostbuster and each ghost is a fixed point in the plane and that no three positions are collinear (that would be very morbid).

A) Argue that there exists a line passing through one Ghostbuster and one ghost such that the number of Ghostbusters on one side of the line equals the number of ghosts on the same side. Describe how to find such a line in $O(n \lg(n))$ time.

B) Give an $O(n^2 \lg(n)$-time algorithm to pair Ghostbusters with ghosts in such a way that no streams cross.

7.2 My Solutions

7.2.1 A) Scanning for Ghosts

I would do something like Graham's scan to find this line. I would start by picking the ghost or Ghostbuster (let's call this entity p_0) with the minimum y-coordinate (and with the minimum x-coordinate in the case of multiple ghosts or Ghostbusters with that y-coordinate value). Then I would sort the remaining ghosts and Ghostbusters by their polar angle with respect to p_0. This calculation and (merge) sort will take $O(cn + n\lg(n))$ time, which is dominated by $O(n\lg(n)))$. Then, iterate through this sorted list, adding 1 to a counter for every point with the opposite 'type' (ghost or Ghostbuster) as p_0 and subtracting 1 to that counter for every point with the same type as p_0. Anytime that counter is at $+1$, you could connect that most recent point with p_0, and we're given that no 3 points are collinear, we know that there will be an even number of ghosts and Ghostbusters both sides of that line. This scan will take $O(n)$ time, which is dominated by $O(n\lg(n))$, so this algorithm operates in $O(n\lg(n))$ time.

7.2.2 B) Don't Cross the Streams!

The algorithm in part A) can be modified to both pair all ghosts with Ghostbusters, and guarantee that the line connecting them isn't crossed by another pair. Rather than just returning the dividing line, the algorithm can return the dividing line as well as the lists of points on both sides. This set can be recursively divided until there are only 2 points, at which those pairs are returned. At best, this would take $\lg(n)O(n\lg(n))$ time to find all pairs, but the worst case scenario would occur if the algorithm kept returning a single pair and the remainder of the set, and in this case, it would take $(n-1)O(n\lg(n))$ which is $O(n^2\lg(n))$ time.