Quokkas + Assignment 2 Solutions

Jess McBroom

September 24, 2017

Question 1

(a) Algorithm Description

(**Intuition:** when the points are sorted, their importance basically corresponds to their index, except in the case of ties, where their importance will be the same as the point before.)

We first sort the points in P in ascending order. We then iterate through the points in this order, computing the importance of each point, p_i, as follows:

$$\mathbf{imp}^*(p_i) = \begin{cases} 0 & i = 0 \\ \mathbf{imp}^*(p_{i-1}) & i > 0 \text{ and } p_i = p_{i-1} \\ i & i > 0 \text{ and } p_i \neq p_{i-1} \end{cases}$$

(b) Algorithm Correctness

We prove correctness by induction. Let $\mathbf{imp}^*(p_i)$ be the computed importance of p_i, and let $\mathbf{imp}(p_i)$ be the actual importance.

Base Case. Since the list of points is in ascending order, the first point, p_0, has the smallest value. Thus, it cannot be superior to any other point and its importance is 0. So, $\mathbf{imp}(p_0) = 0 = \mathbf{imp}^*(p_i)$, and the calculation is therefore correct in the base case.

Inductive Hypothesis. Assume $\mathbf{imp}^*(p_k) = \mathbf{imp}(p_k), k \geq 0$.

Inductive Step. We now show that, under the inductive hypothesis, $\mathbf{imp}^*(p_{k+1}) = \mathbf{imp}(p_{k+1})$.

Case 1: if $p_{k+1} \neq p_k$, then the first $k+1$ points must all be strictly less than p_{k+1}, since the points are in ascending order. In addition, due to the sorting, any points after p_{k+1} cannot have a lower value. This means p_{k+1} is superior to exactly $k+1$ points and its importance is $k+1$. So, in this case, $\mathbf{imp}(p_{k+1}) = k+1 = \mathbf{imp}^*(p_{k+1})$.

Case 2: if $p_{k+1} = p_k$, then:

$$\begin{aligned} \mathbf{imp}^*(p_{k+1}) &= \mathbf{imp}^*(p_k) \\ &= \mathbf{imp}(p_k) && \text{(inductive hypothesis)} \\ &= \Big|\{p_i \in P \mid p_i < p_k\}\Big| && \text{(by definition)} \\ &= \Big|\{p_i \in P \mid p_i < p_{k+1}\}\Big| && \text{(since } p_k = p_{k+1}\text{)} \\ &= \mathbf{imp}(p_{k+1}) \end{aligned}$$

In all cases [1], if $\mathbf{imp}^*(p_k) = \mathbf{imp}(p_k)$, then $\mathbf{imp}^*(p_{k+1}) = \mathbf{imp}(p_{k+1})$. Now, $\mathbf{imp}^*(p_0) = \mathbf{imp}(p_0)$, so $\mathbf{imp}^*(p_1) = \mathbf{imp}(p_1)$, $\mathbf{imp}^*(p_2) = \mathbf{imp}(p_2)$ and so on. Thus, we have correctly computed the importance of all points in P.

[1] Note that we do not need to consider the case where there is no p_k, since $k \geq 0$

Figure 1: This is the perfect assignment to make an IMPORTANT POINT about how cute quokkas are! :D

(c) Complexity

We can sort P in $O(n \log n)$ time. We need to iterate through n points and each time we have a constant amount of work to do (since we have already calculated $\mathbf{imp}^*(p_{i-1})$ upon reaching p_i). Thus, it takes $O(n) \times O(1) = O(n)$ time for this part. Thus the run time is dominated by the sorting, and so our final bound is $O(n \log n)$.

Question 2

(a i) Description of algorithm

Before we begin, let's introduce some notation: $p_{i,x}$ and $p_{i,y}$ will be the x and y components respectively of point p_i. Also, let $\mathbf{imp}_{P_1}(p_i)$ be the importance of $p_i \in P_1$ with respect to P_1, and $\mathbf{imp}_{P_2}(p_i)$ be the importance of $p_i \in P_2$ with respect to P_2.

Modifying the Problem

In this question, the problem is to merge two multisets of points, P_1 and P_2, where $\forall p_i \in P_1$ and $p_j \in P_2$, $p_{i,x} < p_{j,x}$. This is included in the broader class of problems where $\forall p_j \in P_1$ and $p_i \in P_2$:

1. $p_{j,x} < p_{i,x}$, or
2. ($p_{j,x} = p_{i,x}$ and $p_{j,y} \geq p_{i,y}$)

We are going to solve this more general class of problems instead of the original to help in the next question. An example of this larger class of problems is shown in Figure 3. Note that, since this is a more general problem, if our algorithm can correctly solve it then it will also solve the original problem.

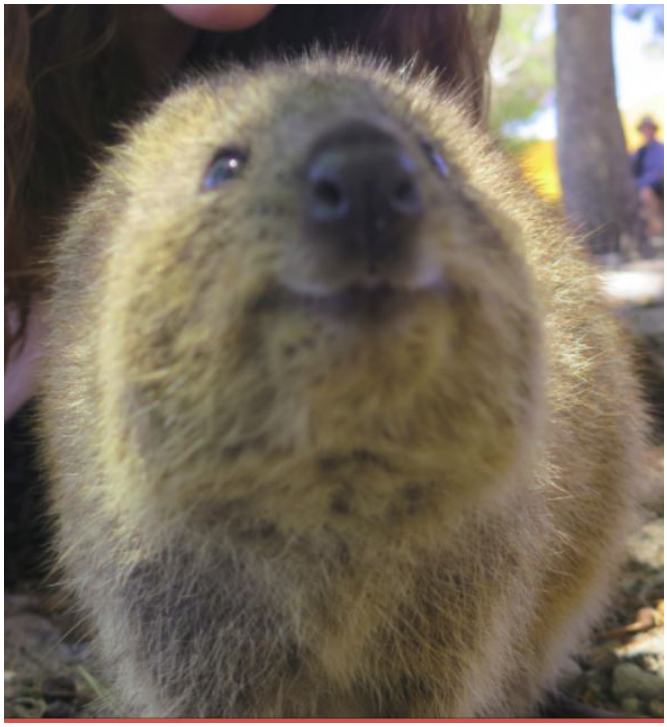

Figure 2: The quokka is SUPERIOR to all other animals!

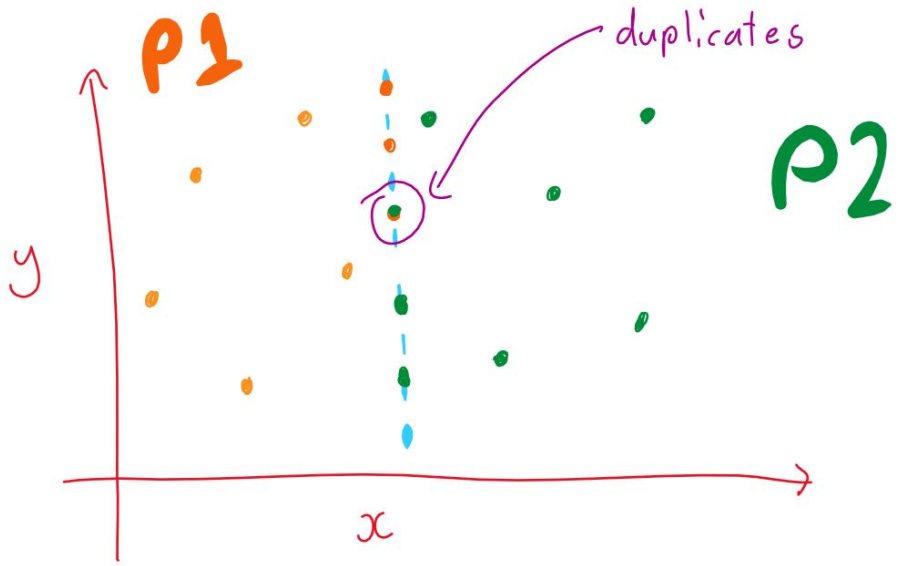

Figure 3: The broader class of problems, where points in P_1 have an x-coordinate $\leq$ to all points in P_2. However, in the case where the x-coordinates are the same, the points in P_1 must have a y-coordinate $\geq$ the P_2 points.

The Algorithm

The general idea is to not change the importance of any point in P_1, but to increase the importance of each point, p_i in P_2 by the number of points in P_1 with a strictly smaller y-coordinate and x-coordinate. To do this efficiently, we will iterate through P_1 and P_2 in parallel, considering the points in order of increasing y-coordinates. In the case of ties, we choose the point from P_2 first.

While iterating, we will keep track of the number of points in P_1 we have already passed. We will use a counter beginning at 0 to do this, and increment it each time we reach a point in P_1. Let the value of this counter upon reaching point p_i be c_i. Then, the new importance for each point will be given by:

$$\mathbf{imp}^*(p_i) = \begin{cases} \mathbf{imp}_{P1}(p_i) & p_i \in P1 \\ \mathbf{imp}_{P2}(p_i) + c_i & p_i \in P2 \end{cases}$$

Once we have computed the importance of each point, we iterate through P_1 and P_2 one more time in parallel, again visiting points in order of lowest to highest y-coordinate. However, this time if there are ties, we visit the point with the smallest x-coordinate first. We return the points in this order.

(a ii) Proof of correctness

Let p_i be an arbitrary point in $P = P_1 \cup P_2$. We need to prove that $\mathbf{imp}^*(p_i) = \mathbf{imp}(p_i)$, where $\mathbf{imp}(p_i)$ is the actual importance of p_i and $\mathbf{imp}^*(p_i)$ is our computed importance.

Claim: $\mathbf{imp}^*(p_i) = \mathbf{imp}(p_i)\ \forall p_i \in P_1$

Proof.

$$\begin{aligned}
\mathbf{imp}^*(p_i) &= \mathbf{imp}_{P_1}(p_i) \\
&= \left|\left\{p_j \in P_1 \mid p_i \text{ superior to } p_j\right\}\right| && \text{(by definition)} \\
&= \left|\{p_j \in P_1 \mid p_{j,y} < p_{i,y} \text{ and } p_{j,x} < p_{i,x}\}\right| \\
&= \left|\{p_j \in P_1 \cup P_2 \mid P_{j,y} < P_{i,y} \text{ and } p_{j,x} < p_{i,x}\}\right| \\
&\qquad \text{(Since no points in } P_2 \text{ have an x-coordinate less than } p_{i,x}) \\
&= \left|\left\{p_j \in P \mid p_i \text{ superior to } p_j\right\}\right| \\
&= \mathbf{imp}(p_i)
\end{aligned}$$

□

Claim: $\mathbf{imp}^*(p_i) = \mathbf{imp}(p_i)\ \forall p_i \in P_2$

Proof.

$$\begin{aligned}
\mathbf{imp}^*(p_i) &= \mathbf{imp}_{P_2}(p_i) + c_i \\
&= \mathbf{imp}_{P_2}(p_i) + \left|\left\{p_j \in P_1 \mid p_{j,y} < p_{i,y}\right\}\right| \\
&\quad (c_i \text{ is the number of points in } P_1 \text{ passed, and if } p_{j,y} \geq p_{i,y},\ p_i \in P_2 \text{ will be considered first}) \\
&= \mathbf{imp}_{P_2}(p_i) + \left|\left\{p_j \in P_1 \mid p_{j,y} < p_{i,y} \text{ and } p_{j,x} < p_{i,x}\right\}\right| \\
&\quad \text{(By def., either } p_{j,x} < p_{i,x} \text{ OR } (p_{j,x} = p_{i,x} \text{ and } p_{j,y} \geq p_{i,y}), \text{ and it can't be the latter)} \\
&= \left|\left\{p_j \in P_2 \mid p_{j,y} < p_{i,y} \text{ and } p_{j,x} < p_{i,x}\right\}\right| + \left|\left\{p_j \in P_1 \mid p_{j,y} < p_{i,y} \text{ and } p_{j,x} < p_{i,x}\right\}\right| \\
&= \left|\left\{p_j \in P_1 \cup P_2 = P \mid p_{j,y} < p_{i,y} \text{ and } p_{j,x} < p_{i,x}\right\}\right| && (P_1 \text{ and } P_2 \text{ are disjoint}) \\
&= \left|\left\{p_j \in P \mid p_i \text{ superior to } p_j\right\}\right| \\
&= \mathbf{imp}(p_i)
\end{aligned}$$

□

Conclusion

We have shown that $\forall p_i \in P_1 \cup P_2$, $\mathbf{imp}^*(p_i) = \mathbf{imp}(p_i)$ for the new class of problems. That is, our calculation of the importance of each point is correct for this class of problems. Since the class of problems in the question is included in our new class of problems, we thus calculate the importance correctly for the question too. When we iterate through P_1 and P_2 in parallel the second time, we visit points from

lowest to highest, choosing the point with the lowest x-coordinate first if there are ties. Thus, we will also return the points in the correct order - from lowest to highest y-coordinate, with points with lower x-coordinates coming first.

(a iii) Complexity

The points are already sorted, so we don't need to do any sorting. The first time we go through P_1 and P_2 in parallel, we consider each point once, giving n iterations, and each iteration requires constant work, so this whole step takes $O(n) \times O(1) = O(n)$ operations. Similarly, the second time we iterate through P_1 and P_2 in parallel, we have n iterations and constant work in each, again giving $O(n)$ operations. Thus, the final upper bound is $O(n)$.

(b i) Algorithm Description

Note that this algorithm does not assume the points are distinct. We will first sort the points in two different ways:

1. By x-coordinate, from lowest to highest. If there are ties, the points with the largest y-coordinate should come first. We'll call the resulting sorted list A.

2. By y-coordinate, from lowest to highest. If there are ties, the points with the lowest x-coordinate should come first. We'll call the resulting sorted list B.

We then call the function, *calculate_importance* (let's call this CI), which we are about to describe below, with input A and B and return the result.

function: calculate_importance (CI)

This function takes as input two lists of the same points, X and Y, ordered by x and y coordinates respectively in the same way as described above. If these lists only contain one point, then it sets the importance of that point to 0 and returns that point.

Otherwise, it first splits X into two halves of size $\frac{n}{2}$: X_1 and X_2. All points in X_1 are marked as belonging to P_1, and all points in X_2 are marked as belonging to P_2. Next, it creates two new lists, Y_1 and Y_2 of size $\frac{n}{2}$, then iterates through all points in Y, placing the points marked P_1 in Y_1 and the points marked P_2 in Y_2 in the order they are reached. It then calls itself twice, first with input X_1 and Y_1, and second with input X_2 and Y_2. The results of the two calls are merged using the merge algorithm defined in part (a i) and the resulting list of points is returned.

(b ii) Correctness

We will first prove the correctness of the function, CI, by induction. We are proving:

If the input is correct, CI computes the importance of all points correctly

Base case: prove CI is correct for $|X| = |Y| = 1$ when the input is correct

If $|X| = 1$, then there is exactly one point. This means the point is superior to no other points and thus its importance must be 0. If there is exactly one point, the its importance is set to 0, so the function is correct in the base case.

Inductive Hypothesis: CI is correct when $|X| \leq k$ where $k, |X| \in \mathbb{N}$ and input valid

Inductive Step: prove CI correct for $|X| \leq k+1$ if induction hypothesis is true and input is valid

We will prove each part of the function is correct when $|X| = k+1$. Since $|X| = k+1 > 1$, the function will perform two recursive calls to itself and merge the results. We need to prove this is done correctly.

Figure 4: All quokkas are wonderful, so any two quokkas are INCOMPARABLE

Recursive calls. We first show that the input to these calls is valid (ie. ordered correctly) and then that the size of the input for these calls is $\leq k$.

1. **Valid input**. Clearly, X_1 and X_2 will be ordered correctly, since X is ordered correctly and they are simply the first and second half of X. Y_1 and Y_2 will also be ordered correctly, because when we iterate through Y (already ordered), we add the elements that appear earlier in Y to Y_1 and Y_2 first, and so they will still be ordered. Clearly, they will also contain the same points. So, we can be assure the input will be valid.

2. **Input Size** $\leq k$. The size of the two smaller lists is $\frac{k+1}{2}$, which is $\leq k$ since $k > 1$.

Thus, we are making two calls to CI when the input size is $\leq k$ and the input is valid and so, by the induction hypothesis, the output of these functions is correct.

Merge step. We have already proven that the merge algorithm is correct in part a(ii), but only provided that the input is valid, which we therefore need to show:

1. **Points sorted correctly.** The points need to be sorted by y-coordinate (lowest to highest), then by x-coordinate (lowest to highest) if there are ties. CI outputs points in this order and, since we have determined the output of the two recursive calls to be correct, the input is clearly valid in this respect.

2. $\forall p_j \in P_1$ **and** $p_i \in P_2$**: either 1:** $p_{j,x} < p_{i,x}$ **or 2:** $(p_{j,x} = p_{i,x}$ **and** $p_{j,y} \geq p_{i,y})$. The points in P_1 and P_2 come from X_1 and X_2 respectively, which are the first and second half of X. X is sorted, so all points in X_1 either have a smaller x-coordinate than any point in X_2, or they have the same x-coordinate, in which case their y coordinate is $\geq$ to the X_2 point.

Thus, the input is valid and so merge algorithm will correctly compute the importance of each point and, since this algorithm returns the points in the same order CI should, we know the output overall is correct.

Thus, if we can assume the inductive hypothesis is correct, the function is correct when $|X| = k + 1$ and is therefore correct for $|X| \leq k + 1$

Conclusion

From the base case, we know CI is correct when $|X| = 1$. That is, it is correct when $|X| \leq 1$, assuming $|X|$ is a natural number (not including 0). Now, we know that if CI is correct for $k \leq 1$, then it is also

correct for $k \leq 2$, meaning it is also correct for $k \leq 3$, $k \leq 4$, $k \leq 5$ and so on, assuming the input is valid. Thus is it correct for all possible input sizes and so CI is correct.

One final note

Technically, we only proved that CI was correct, not the entire algorithm. The final step is to argue that the original input to CI is correct, which follows immediately from the way we sort A and B.

Part c (iii) Complexity

The initial sorting by x and y coordinates can be done in $O(n \log n)$ time.

Now we must compute the time taken by CI on input size n. Splitting X into X_1 and X_2 and marking the points can be done in $O(n)$ time. Similarly, splitting Y into Y_1 and Y_2 takes $O(n)$ time. The merge step at the end can be done in $O(n)$ time, as we showed in Part (b iii). Thus, the overhead is $O(n)$. In the base case, $T(1) = O(1)$, otherwise we need to solve two sub-problems of half the original size, so the time taken is given by:

$$\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + O(n) \\
&= 2T\left(\frac{n}{2}\right) + O(n^{\log_2 2} \log^0 n) \\
&= 2T\left(\frac{n}{2}\right) + O(n^{\log_\beta \alpha} \log^k n) && \text{(k=0)} \\
&= O(n^{\log_\beta \alpha} \log^{k+1} n) && \text{(Master Thm, Case 2)} \\
&= O(n \log n)
\end{aligned}$$

Therefore, an upper bound on the running time is $O(n \log n)$, since both the sorting and call to CI take this time.

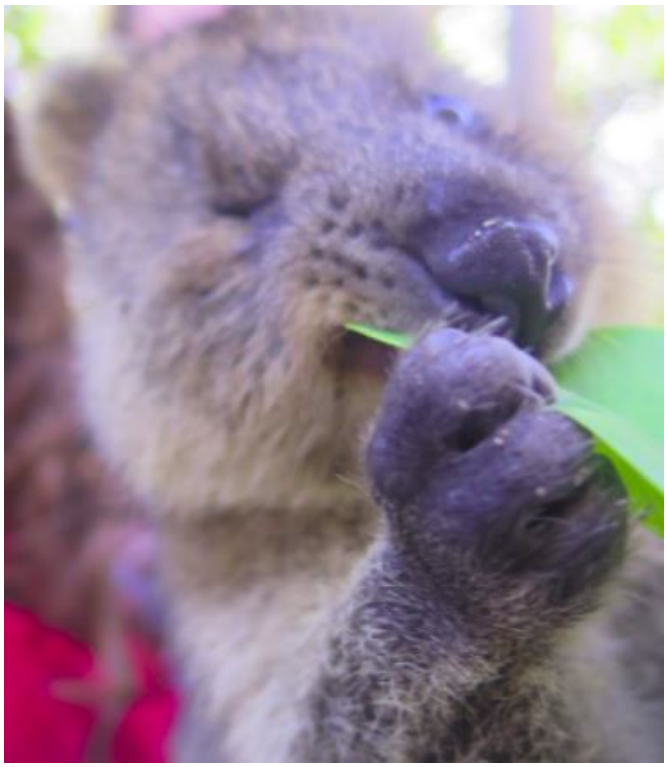

Figure 5: All done!