Name:	Suhas Kamath	**Assignment No:**	Assignment 1
SR No:	06-18-01-10-51-25-1-25945	**Course Code:**	DS221
Email ID:	suhaskamath@iisc.ac.in	**Course Name:**	Intro. to Scalable Systems
Date:	August 29, 2025	**Term:**	AUG 2025

Question 1

Solution Approach

In this question, we had to find the minimum weight of the duplicate parcels. The list of parcels was provided in the input file. An initial approach could be to run a nested for loop, which should read the list and identify the minimum weight of the duplicate parcels. This would be an inefficient algorithm. A much better algorithm is as follows:

1. Create a hash map to store the id and the minimum weight of each parcel.
2. Create a set to store the list of package ids that have duplicates.
3. Loop through the list of parcels and update the minimum weight with respect to each id in the above hash map.
4. During this loop, if a package is found to be duplicated, it is added to the set.
5. By the end of the for loop, we have two pieces of data: the id numbers of the duplicated parcels, and the minimum weight of each parcel.
6. Using the above two pieces of data, we create a separate two-dimensional vector array to store the list of duplicated packages with their minimum weight. We then return this vector array.

Time and Space Complexity Analysis

Time Complexity

First, let us analyze the program to determine its time complexity. The code can be divided into the following major sections, namely:

1. Creation and initialization of the required data structures.
2. The for loop that calculates the list of duplicate package ids and the minimum weight of each parcel.
3. The for loop that takes compiles the final result, i.e duplication parcel ids and their respective minimum weights.

Let us now analyze the time complexity of each section. Let us assume that $n =$ number of parcels and $m =$ number of duplicate parcels.

Section	Time Complexity	Reason
Section 1	$O(1)$	Initialization of variables and data structures takes constant time, independent of the values of n and m.
Section 2	$O(n)$	The loop executes n times. Hash map and set operations (insert, find, update) take $O(1)$ on average. Therefore, the total average time complexity is $O(n)$.
Section 3	$O(m)$	The loop iterates through the m duplicate IDs. For each ID, it performs an $O(1)$ hash map lookup. The total time is therefore $O(m)$.

Because we know that the number of duplicate parcels is definitely less than the total number of parcels, we can state that $m < n$. From this, we can state that the asymptotic worst case complexity of the code is $O(n)$.

Space Complexity

The asymptotic space complexity of this algorithm is $O(n)$.

Experimental Setup

Empirical Observations

Additional Insights

Some further optimizations can be made if the data was sorted on basis of ID. In such a case, a simple for loop would suffice. This would essentially enhance the time complexity to $O(n)$. The space complexity, however, shall remain at $O(n)$.

Name:	Suhas Kamath	**Assignment No:**	Assignment 1
SR No:	06-18-01-10-51-25-1-25945	**Course Code:**	DS221
Email ID:	suhaskamath@iisc.ac.in	**Course Name:**	Intro. to Scalable Systems
Date:	August 29, 2025	**Term:**	AUG 2025

Question 2

Solution Approach

This question is a variation of the lowest common ancestor problem. The conveyor network of the parcel routing system can be modeled after a binary tree.

Hence, the initial idea was to build a binary tree from the pre-order and in-order traversals provided. After the tree has been created, we can map the parcels their respective leaf nodes. We can then analyze the parents of the leaf nodes to find the common ancestor, i.e the last node where all the queried parcels were together. This could be easily done by traversing the tree in a post-order fashion.

However, during the coding of the tree construction, I realized that the tree was being constructed in post-order itself. In other words, the left and right sub-tree were constructed before the parent. Because of this, we can complete the analysis as the tree is being built.

After this was implemented, I got another idea: why do we need to finish the construction of the tree if all queries have been resolved? In the previous implementation, I was building the tree in full, without checking if there are any queries left. As mentioned above, the tree is constructed in post-order fashion, which means that the parent is the last to be constructed. If all the queries in the query list are resolved by the left and right sub-trees, there is no need to construct the tree.

Time and Space Complexity Analysis

Time Complexity

First, let us analyze the program to determine its time complexity. The code can be divided into the following major sections, namely:

1. Converting the in-order traversal vector to an unordered hash map, so that it would be easier to construct the tree.
2. Construction of the tree and resolution of the queries. This is done in the following steps, for each individual node in the parcel routing system:
   (a) Creating an object from the TreeNode class for the node.
   (b) Constructing the left and right subtrees of the node.
   (c) Listing the parcels that passed through this particular node.
      i. If the node is a leaf node, its parcels are requested from the Helper class. This class shall return a list of the parcels passing through the particular node.
      ii. If the node is not a leaf node (i.e it is an internal node), its parcel list will be constructed by merging the parcel lists of both of its children.
   (d) Finally, the queries in the query list are checked again this particular node. If all the parcels in a particular query are present at this node, it is added to the q2_result vector.
3. By the time that the above section is done, all queries with a solution will be resolved, and their respective node numbers will be added to the q2_result vector.

Let us now analyze the time complexity of each section. Let us assume that $n =$ number of nodes, $p =$ number of parcels and $q =$ number of queries.

Section	Time Complexity	Reason
Section 1	$O(n)$	Inserting n nodes into an unordered hash map takes $O(1)$ average time per insertion, leading to a total time complexity of $O(n)$.
Section 2	$O(n \cdot p \cdot q)$	The tree construction is a recursive process that visits each of the n nodes once. At each node, we merge parcel lists and check queries. If p is the average number of parcels in a query and q is the number of queries, this step can be complex.
Section 3	$O(1)$	The results are already compiled during the tree construction in Section 2, so this step is effectively instantaneous.

From this, we can state that the asymptotic worst case complexity of the code is $O(n \cdot p \cdot q)$.

Name:	Suhas Kamath	**Assignment No:**	Assignment 1
SR No:	06-18-01-10-51-25-1-25945	**Course Code:**	DS221
Email ID:	suhaskamath@iisc.ac.in	**Course Name:**	Intro. to Scalable Systems
Date:	August 29, 2025	**Term:**	AUG 2025

Space Complexity

The space complexity is dictated by the storage for the tree itself, the recursion stack, and the parcel sets.

- The in-order index map requires $O(n)$ space.
- The recursion stack for 'buildTree' can go as deep as $O(n)$ in the case of a skewed tree.
- The sets of parcels stored at each node can, in the worst case, lead to a total storage of $O(n \cdot p)$.

Thus, the asymptotic space complexity is dominated by the parcel sets, resulting in $O(n \cdot p)$

Experimental Setup

Empirical Observations

Additional Insights

For a very large number of queries, a different approach might be more scalable. After building the tree in $O(n)$, one could pre-process it to answer lowest common ancestor queries in $O(\log n)$ time per query using techniques like binary lifting. Binary lifting is a technique used to answer lowest common ancestor queries in $O(\log n)$ time by pre-processing each node's ancestors.

This would involve a more significant pre-processing time ($O(n \log n)$) but would be much faster if the number of queries q is extremely large, relative to the number of nodes (i.e. $q >>> n$).

Name:	Suhas Kamath	**Assignment No:**	Assignment 1
SR No:	06-18-01-10-51-25-1-25945	**Course Code:**	DS221
Email ID:	suhaskamath@iisc.ac.in	**Course Name:**	Intro. to Scalable Systems
Date:	August 29, 2025	**Term:**	AUG 2025

Question 3

Solution Approach

Time and Space Complexity Analysis

Experimental Setup

Empirical Observations

Additional Insights