Engineering and Applied Science Programs for Professionals

Whiting School of Engineering

Johns Hopkins University

685.621 Algorithms for Data Science

Homework 1 – SANDESH IYER

Assigned at the start of Module 1

Total Points 100/100

1. **Problem 1 Chapter 2** ***Note this is a Collaborative Problem***
   20 Points Total
   Use induction to prove $\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2$.

Base case: n = 1

$$\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2$$

$$\sum_{i=1}^{n} 1^3 = \left(\frac{1(1+1)}{2}\right)^2$$

$$1 = 1$$

Inductive hypothesis

Assume

$$\sum_{i=1}^{k} i^3 = \left(\frac{k(k+1)}{2}\right)^2$$

$$1^3 + 2^3 + 3^3 + \dots k^3 = \left(\frac{k(k+1)}{2}\right)^2$$

Inductive step: We want to prove where n=k+1

$$\sum_{i=1}^{k+1} i^3 = \left(\frac{(k+1)((k+1)+1)}{2}\right)^2 = \left(\frac{(k+1)(k+2)}{2}\right)^2$$

$$1^3 + 2^3 + 3^3 + \dots k^3 + (k+1)^3 = \left(\frac{(k+1)(k+2)}{2}\right)^2$$

By induction hypothesis,

$$\left(\frac{k(k+1)}{2}\right)^2 + (k+1)^3 = \left(\frac{(k+1)(k+2)}{2}\right)^2$$

$$\frac{k^2(k+1)^2}{4} + (k+1)^3 = \left(\frac{(k+1)(k+2)}{2}\right)^2$$

$$\frac{(k+1)^2(k^2+4(k+1))}{4} = \left(\frac{(k+1)(k+2)}{2}\right)^2$$

$$\frac{(k+1)^2(k+2)^2}{4} = \left(\frac{(k+1)(k+2)}{2}\right)^2$$

$$\left(\frac{(k+1)(k+2)}{2}\right)^2 = \left(\frac{(k+1)(k+2)}{2}\right)^2$$

Hence by the induction rule

$$\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2$$

2. **Problem 2 Parts a, b, c, d, e and f**
30 Points Total 5 Points Each

Although merge sort runs in $\Theta(nlgn)$ 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 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.

(a) Use insertion sort to sort the unsorted array $< 40, 17, 45, 82, 62, 32, 30, 44, 93, 10 >$. Make sure to show the array after every pass.

(b) Use merge sort to sort the unsorted array $< 75, 56, 85, 90, 49, 26, 12, 48, 40, 47 >$. Make sure to show the steps of splitting the array then merging the array.

(c) Show that insertion sort can sort the n/k sublists, each of length k, in $\Theta(nk)$ worst-case time.

(d) Show how to merge the sublists in $\Theta(nlg(n/k))$ worst-case time.

(e) Given that the modified algorithm runs in $\Theta(nk + nlg(n/k))$ 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 standard merge sort, in terms of Θ-notation?

(f) How should we choose k in practice?

a) Unsorted Array: < 40, 17, 45, 82, 62, 32, 30, 44, 93, 10 >

Iteration 1: < 17, 40, 45, 82, 62, 32, 30, 44, 93, 10 >

Iteration 2: < 17, 40, 45, 82, 62, 32, 30, 44, 93, 10 >

Iteration 3: < 17, 40, 45, 82, 62, 32, 30, 44, 93, 10 >

Iteration 4: < 17, 40, 45, 62, 82, 32, 30, 44, 93, 10 >

Iteration 5: < 17, 32, 40, 45, 62, 82, 30, 44, 93, 10 >

Iteration 6: < 17, 30, 32, 40, 45, 62, 82, 44, 93, 10 >

Iteration 7: < 17, 30, 32, 40, 44, 45, 62, 82, 93, 10 >

Iteration 8: < 17, 30, 32, 40, 44, 45, 62, 82, 93, 10 >

Iteration 9: < 10, 17, 30, 32, 40, 44, 45, 62, 82, 93 > (Sorted)

b) Unsorted array: < 75, 56, 85, 90, 49, 26, 12, 48, 40, 47 >

Step 1: < 75, 56, 85, 90, 49> <26, 12, 48, 40, 47 >

Step 2: < 75, 56> <85, 90, 49> <26, 12> < 48, 40, 47 >

Step 3: <75> <56> <85> <90,49> <26> <12> <48> <40,47 >

Step 4: <75> <56> <85> <90,49> <26> <12> <48> <40,47 >

Step 5: <75> <56> <85> <90> <49> <26> <12> <48> <40> <47 >

Step 6: <75> <56> <85> <49,90> <26> <12> <48> <40,47>

Step 7: <56,75> <49,85,90> <12,26> <40,47,48>

Step 8: <49,56,75,85,90> <12,26,40,47,48>

Step 9: < 12, 26, 40, 47, 48, 49, 56, 75, 85, 90 > (Sorted)

c) Sorting each list takes $ak^2 + bk + c$ for constants a, b and c

Since we have n/k sub-lists then

$\frac{n}{k}(ak^2 + bk + c) = ank + bn + \frac{cn}{k} = \Theta(nk)$

d) We know that in order to sort a sub-lists of length k,

$$T(a) = \begin{cases} 0 & if\ a = 1 \\ 2T\left(\frac{a}{2}\right) + ak & if\ a = 2^p, if\ p > 0 \end{cases}$$

Merging a sublists happens when splitting into two groups of $a/2$ lists. Merge each list recursively and combine in ak steps since we have two arrays of length $ak/2$

Proof by induction for formula $ak \lg a$

Base case: $T(1) = 1k \lg 1 = 0$

Induction hypothesis and steps: To calculate $T(2a)$

$$\begin{aligned} T(2a) &= 2T(a) + 2ak = 2(T(a) + ak) \\ &= 2(ak \lg a + ak) = 2ak\ (\lg a + 1) = 2ak(\lg a + \lg 2) = 2ak\ \lg(2a) \end{aligned}$$

Substituting number of sub-lists n/k for a

$$T\left(\frac{n}{k}\right) = \left(\frac{n}{k}\right) k \lg\frac{n}{k} = n\lg(\frac{n}{k})\ \textit{this proves that the time complexity of merging is}\ \Theta(n\lg(\frac{n}{k})))$$

e) Largest value: $k = \lg n$

Substituting in equation $\Theta(nlg\ n + n\lg(n/\lg n\ )) = \ \Theta(n\lg n)$

When the k values we use is greater than $\lg n$ then we know that the running time will be larger than merge sort.

f) In practice, we would choose k such that it is the length of the largest list where the insertion sort method will be faster than the merge sort method.

3. **Problem 3**
15 Points Total

Write a $\Theta(m + n)$ algorithm that prints the in-degree and the out-degree of every vertex in an m-edge, n-vertex directed graph where the directed graph is represented using adjacency lists.

Solution

The adjacency list consists of a list of directed edges and has count mappings for in-degrees and out-degrees. Initially we could set every vertex to be mapped to a zero. Once that's done, we can loop through each edge (e, v) and increment out-degree counter which is 'e' and in degree counter which is 'v'. After looping through the edges, we can loop through each vertex and then print the result from the mapping.

For time complexity purposes, when we loop through each edge it takes O(m) time and for looping through each vertex it takes about O(n) time, Hence the total time complexity takes about O(m+n) time. Below is sample code that demonstrates this algorithm. 'm' is the number of edges and 'n' is the number of vertices

```
Let 'vertex' be a set of vertices and 'edge' be a set of edges in the directed graph represented using
adjacency lists
in_degree = {}
out_degree = {}

O(n) -> time complexity
for e in vertex:
    in_degree[e] = 0
    out_degree[e] = 0

O(m) - > time complexity
for e, v in edge:
    out_degree[e] += 1
    in_degree[v] += 1

O(n) -> time complexity
```

```
for e in vertex:
    print(out_degree[e])
    print(in_degree[e])
```

4. **Problem 4 Chapter 12 Binary Search Trees**
25 Points Total 5 Points Each

Exercise 12.2-1. Suppose that we have numbers between 1 and 1000 in a binary search tree and we want to search for the number 363. Which of the following sequences could not be the sequence of nodes examined?

i. 2, 252, 401, 398, 330, 397, 363.

ii. 924, 220, 911, 244, 898, 258, 362, 363.

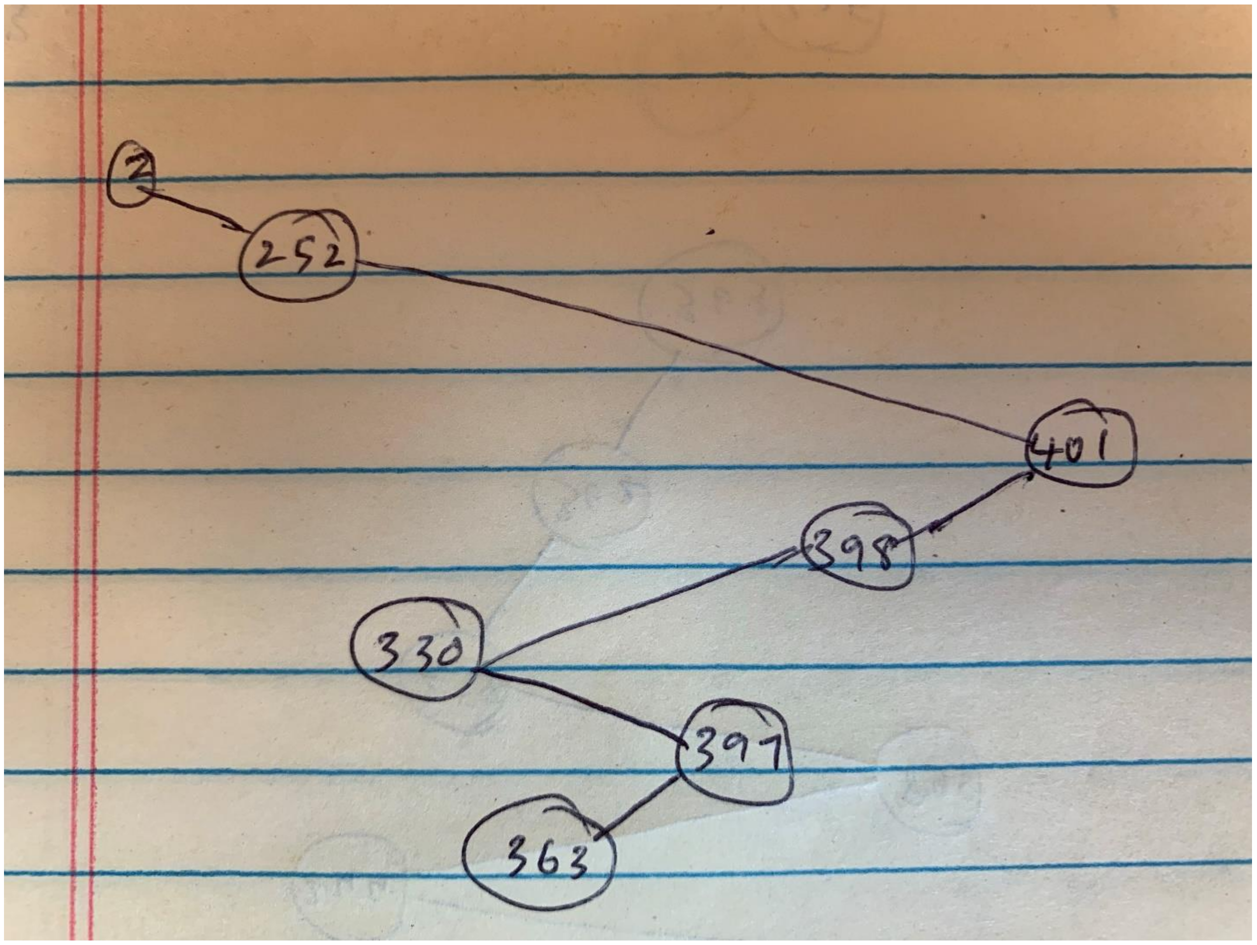
iii. 925, 202, 911, 240, 912, 245, 363.

iv. 2, 399, 387, 219, 266, 382, 381, 278, 363.

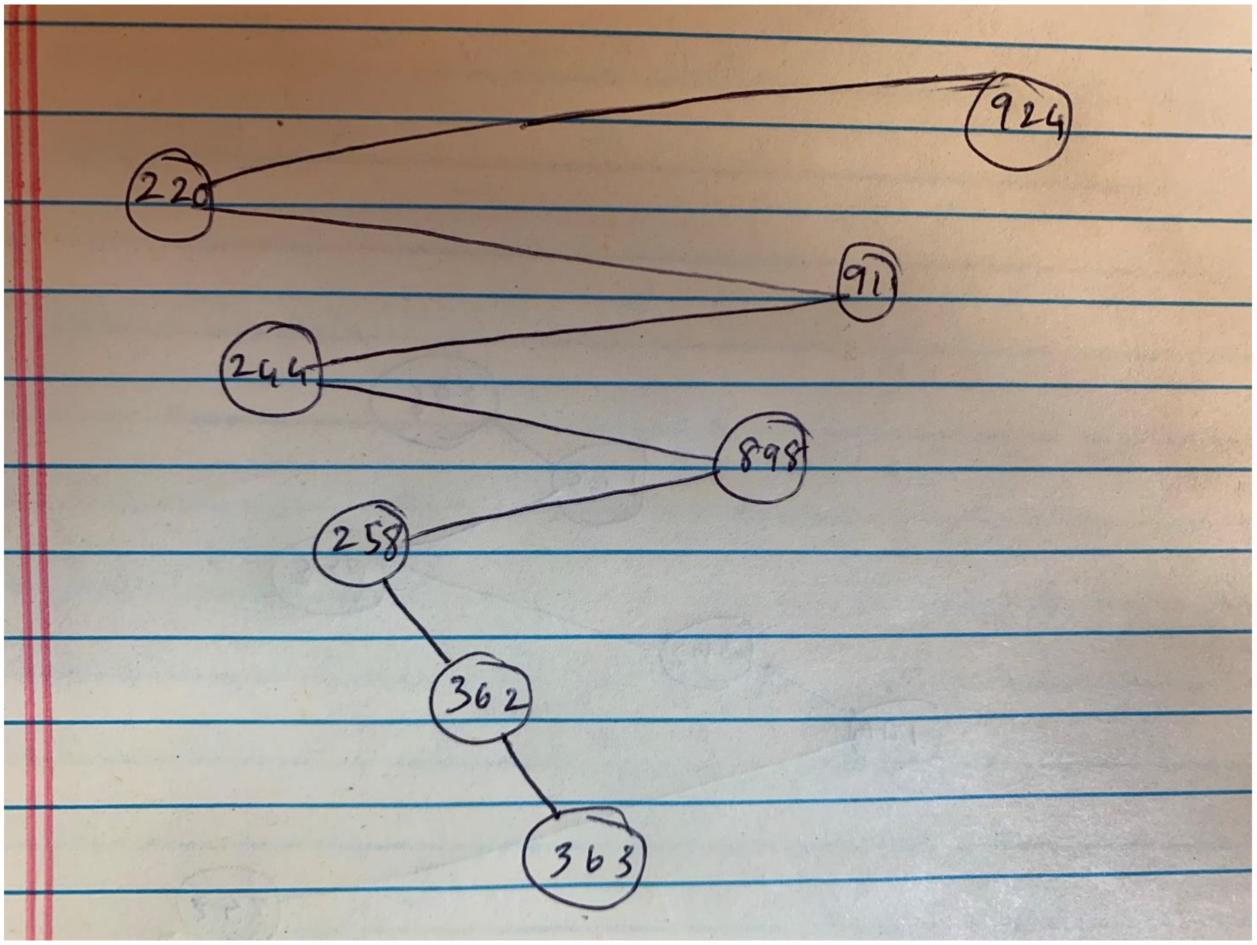
v. 935, 278, 347, 621, 299, 392, 358, 363.

Solution

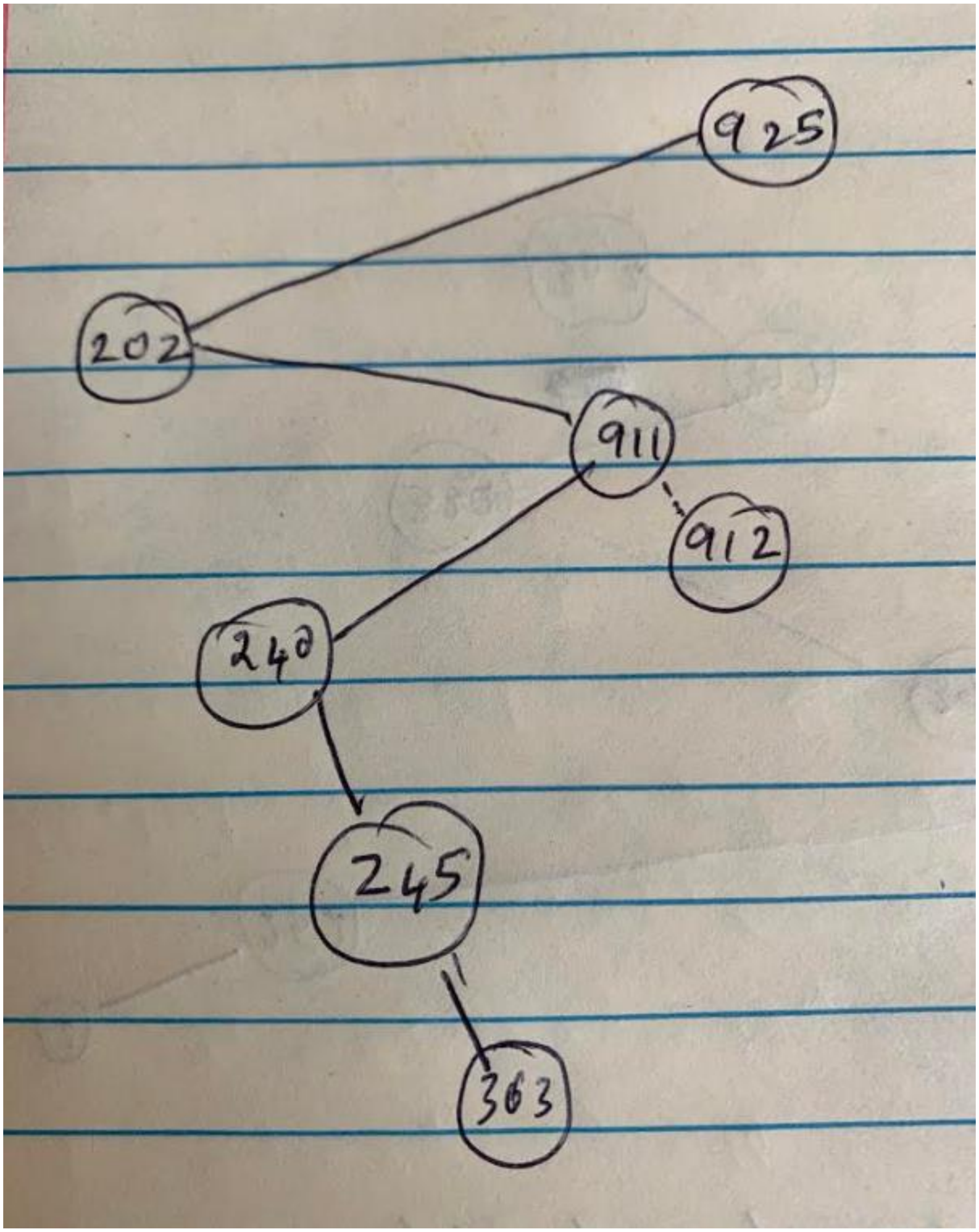
1. It is possible to draw a binary search tree with the given sequence of numbers and it follows the key property of binary search tree. While searching for 363 using key property of BST, all numbers are examined in the given sequence. Hence the given sequence could occur when searching for the number 363.



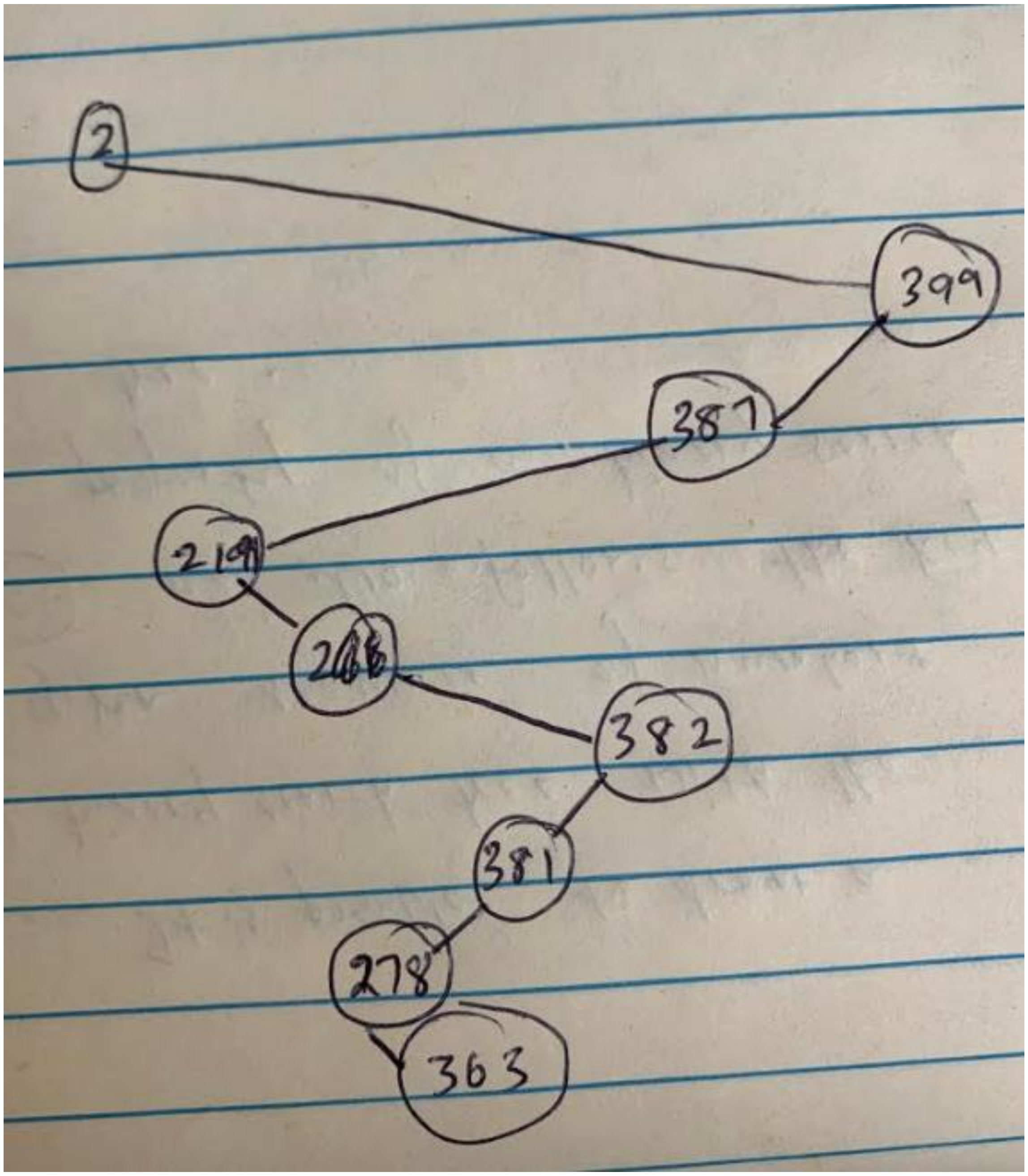
2. It is possible to draw a binary search tree with the given sequence of numbers and it follows the key property of binary search tree. While searching for 363 using key property of BST, all numbers are examined in the given sequence. Hence the given sequence could occur when searching for the number 363.



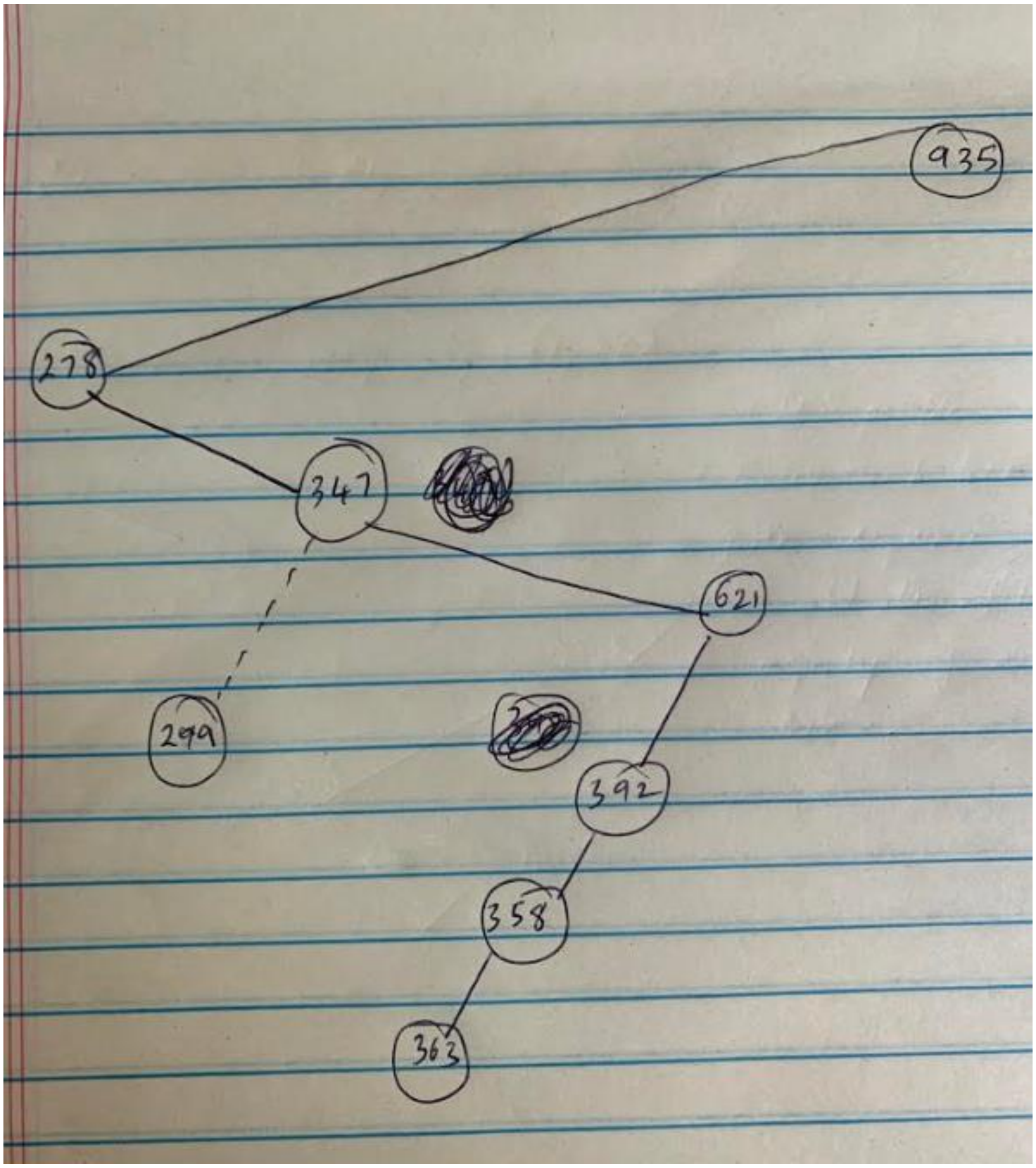
3. When we observe the binary tree with the given sequence of numbers, while searching for 363 the node 912 is not examined since 363 < 911. Hence the sequence of nodes can't be examined when searching for 363.



4. It is possible to draw a binary search tree with the given sequence of numbers and it follows the key property of binary search tree. While searching for 363 using key property of BST, all numbers are examined in the given sequence. Hence the given sequence could occur when searching for the number 363.



5. When we observe the binary tree with the given sequence of numbers, while searching for 363 the node 299 is not examined since 363 > 347. Hence the sequence of nodes can't be examined when searching for 363.


935
278
347
621
299
392
358
363

5. **Problem 5 Sorting Iris Plants** ***Note this is a Collaborative Problem***
10 Points

The Iris Plants Database contains 3 classes of 50 instances each, where each class refers to a type of Iris plant. Five attributes/features were collected for each plant instance. The dataset can be downloaded from iris.arff on the Sample Weka Data Sets webpage (https://storm.cis.fordham.edu/ gweiss/data-mining/datasets.html). Develop an algorithm to sort the five features in the dataset to determine if any of the five sorted features can separate the three plant types. Show the efficiency of your sorting algorithm based on its ability to sort the five sets of features.

Algorithm explanation using python libraries

Step 1: Import the dataset from the website, can be achieved by using loadarff python arff library to load the arff file data for the 3 classes of 50 instances. Each class refers to a type of Iris plant. The dataset also contains attributes/features collected for each plant instance

Step 2: Using pandas the arff data can be modified into a dataframe with all attributes using pandas dataframe method.

Step 3: From the dataframe, we can use 1 attribute at a time and plot it against the number of total points and color the points by grouping by the three classes. The plotting can be done with the help of a python library called seaborn and matplotlib. This would show the before sort graph.

Step 4: In order to sort the values we can use the sort_values method which is an inbuilt python function. This sort method can be set to merge sort. We can also use insertion sort since the number of elements is less. Merge is advantageous when 'n' is large, memory is not a constraint and the array is completely unordered. We can sort each dataset per attribute and then plot the same grouping by three classes once again in terms of coloring on the graph. Each sort runs in O(nlogn) time.

Step 5: From the plots we can find out the accuracy of our sorting algorithm based on its ability to sort the sets of features. For example, if there is a clear indication or separation in the datasets on the graph showing accurate representation of each class in a set or cluster we can say that the efficiency of that particular sorting algorithm is high.

Step 6: From the graphs it would be evident that using petal length and petal width and sorting we can see a clear separation in classes