**Week 1 Assignment**

Name- Vivek Bhurke

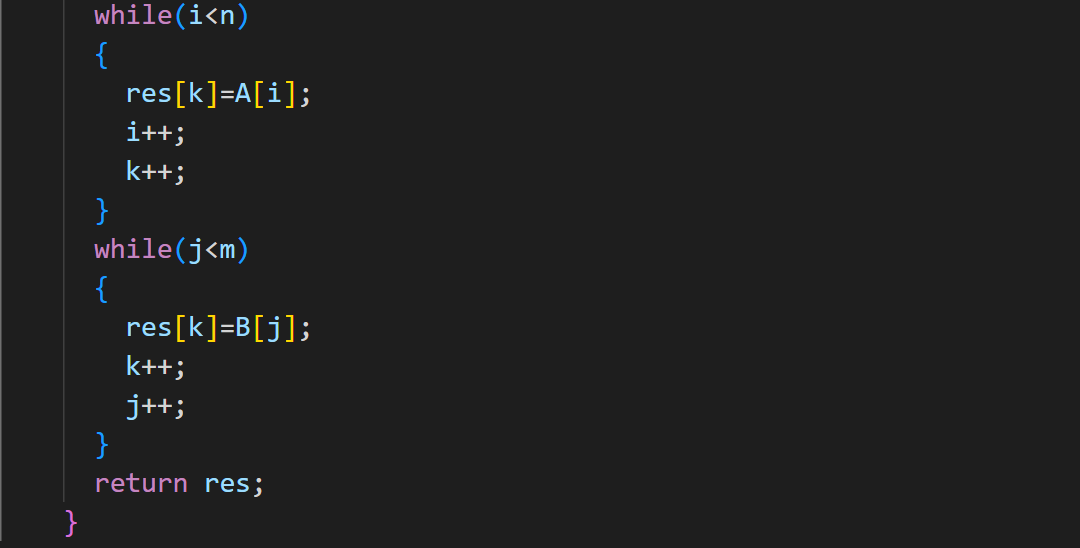
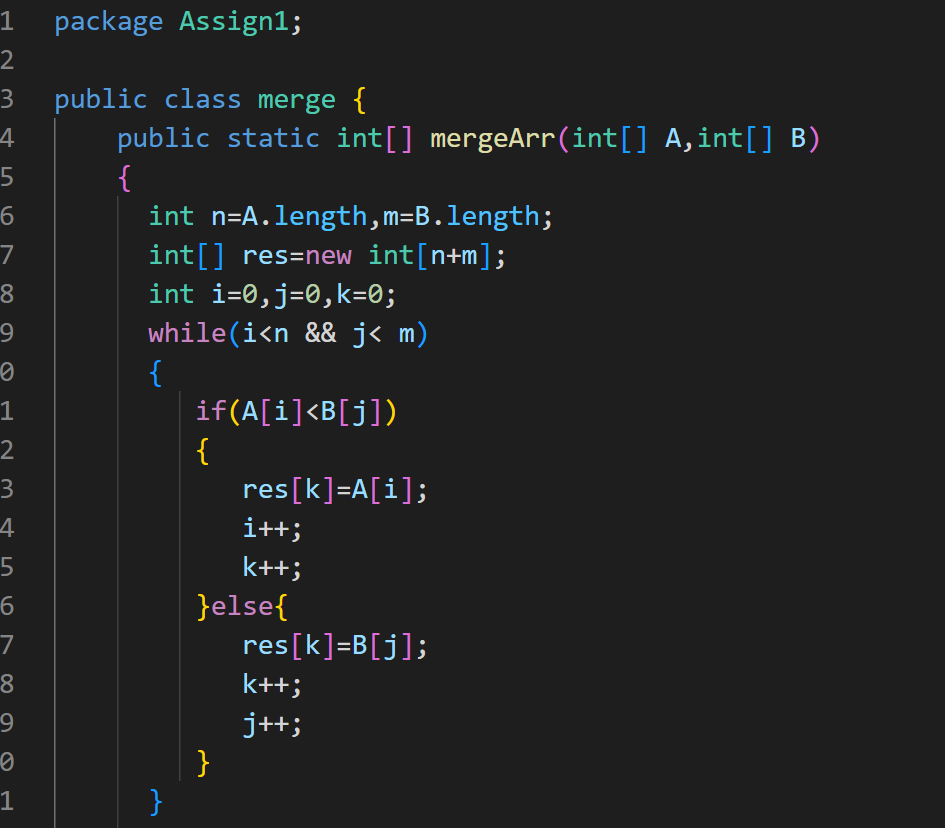
PRN-21510083

Batch -T4

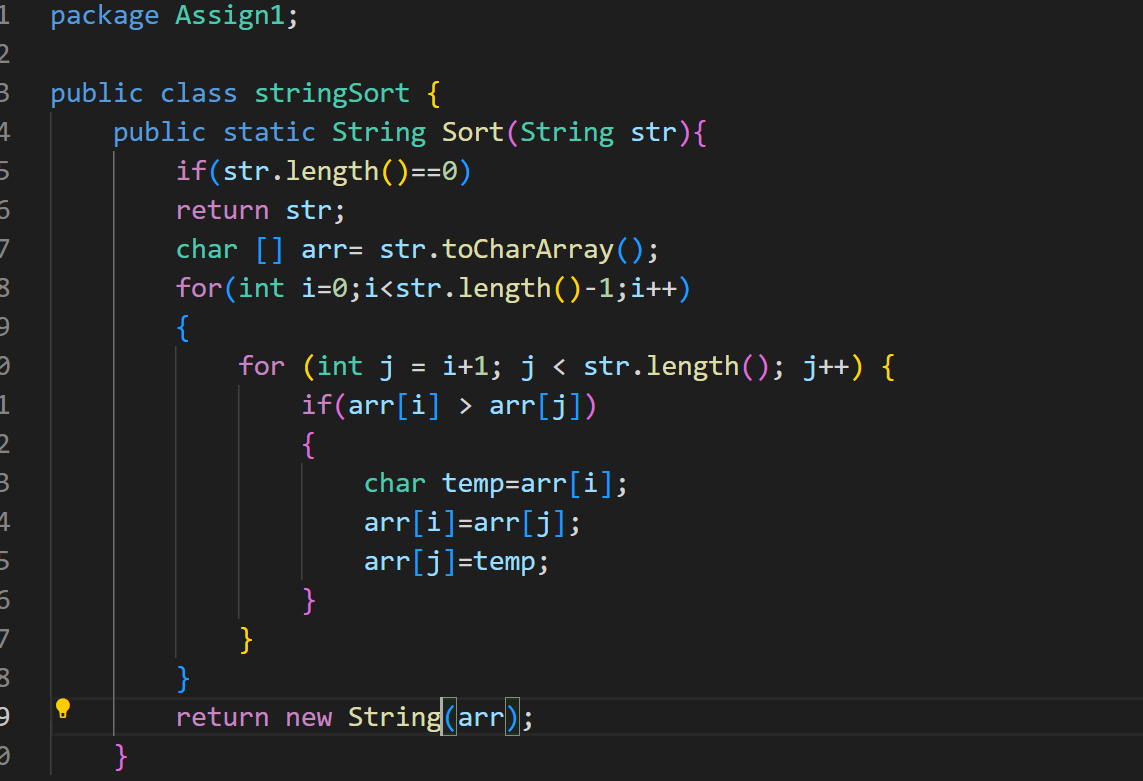
Part: 1

**Sorting Algorithm**

Q) You are given two sorted array, A and B, where A has a large enough buffer at the end to hold B. Write a method to merge B into A in sorted order.



Q) Write a method to sort an array of string so that all the anagrams are next to each other.

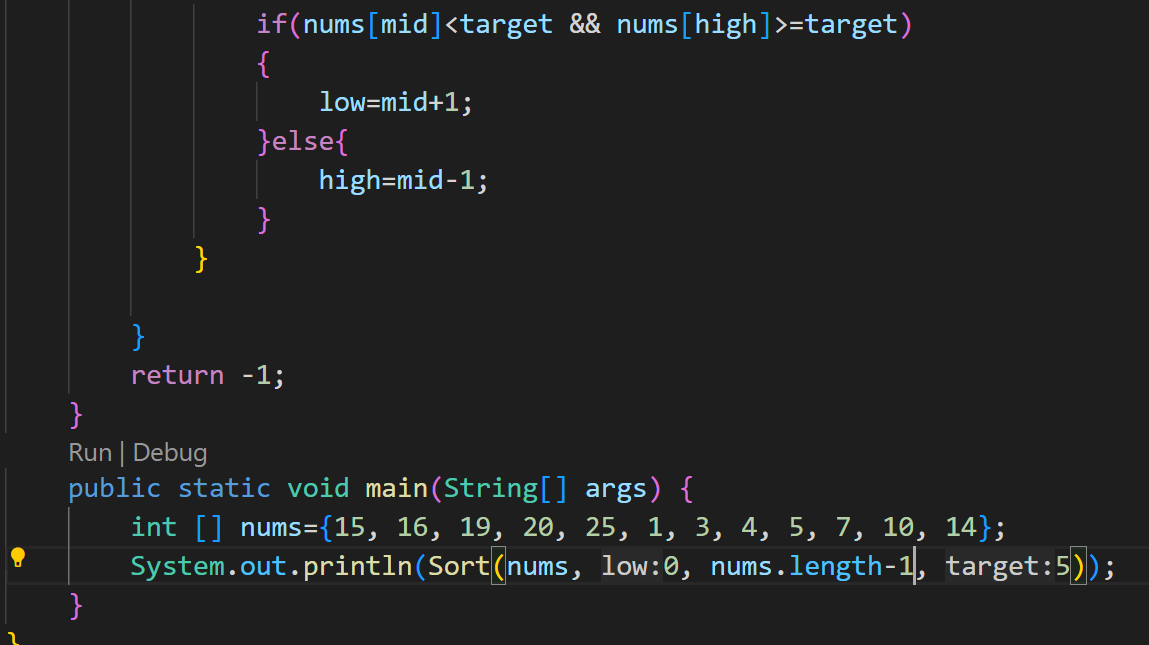
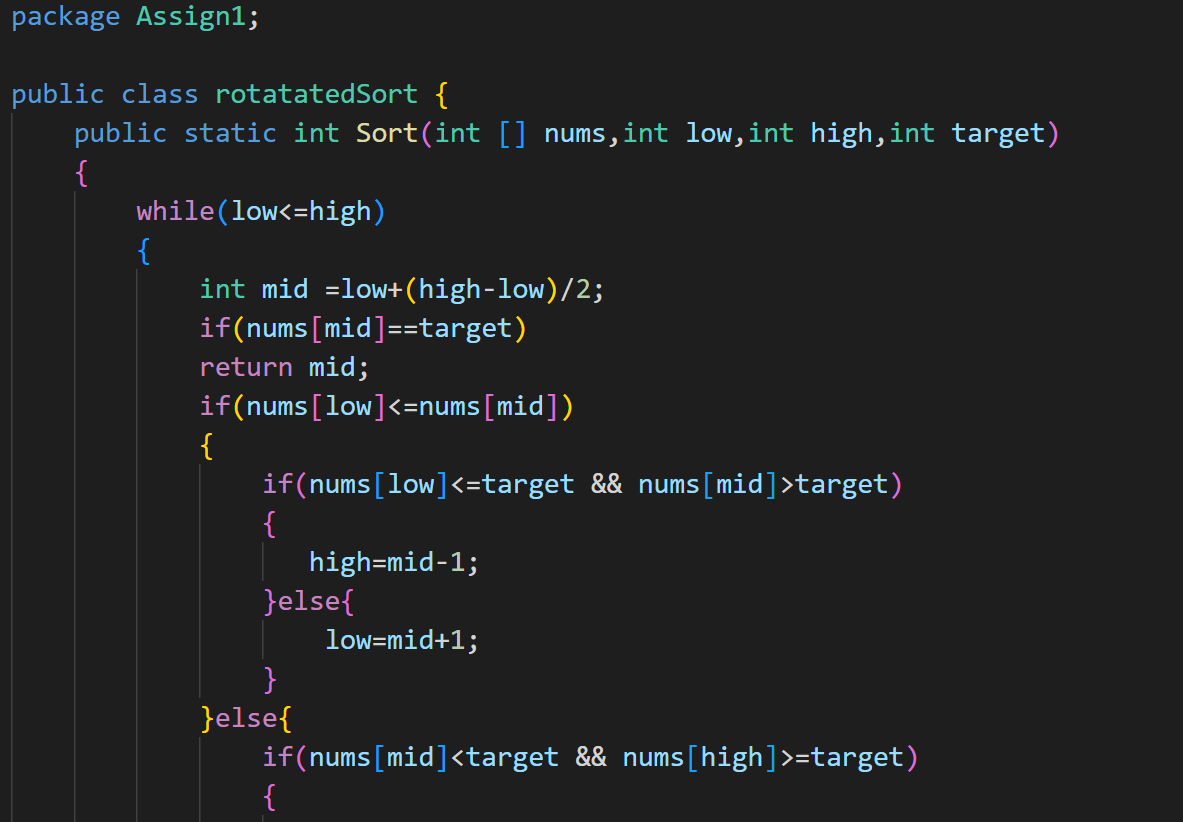


Q) Given a sorted array of *n* integers that has been rotated an unknown number of times, write code to find an element in the array. You may assume that the array was originally sorted in increasing order.

EXAMPLE

Input: find 5 in {15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10, 14}

Output: 8 (the index of 5 in the array)



Q) Imagine you have a 20GB file with one string per line. Explain how you would sort the file.

---à

Sorting a 20GB file with one string per line can be a challenging task to do in-memory due to its large size. However, we can use external sorting techniques to efficiently sort the file even when it doesn't fit entirely into memory. Here's a high-level overview of how we can achieve this:

1. **Divide the File**: First, we divide the large file into smaller manageable chunks that can fit into memory. For example, we can read a certain number of lines (or a fixed-size block) from the file into memory, sort them in-memory, and then write them back to temporary smaller files.
2. **Sort Each Chunk**: Now that we have divided the file into smaller chunks, we sort each chunk in-memory using an efficient sorting algorithm like quicksort or mergesort.
3. **Merge Sorted Chunks**: Once all the chunks are sorted, we merge them back into a single sorted file. We use a k-way merge algorithm, where we maintain a min-heap (priority queue) of the first lines from each sorted chunk and keep selecting the smallest line to write to the output file.
4. **Repeat Merging (if necessary)**: If the number of chunks is still too large to fit into memory for merging, we repeat the merging process recursively until we obtain the final sorted file.

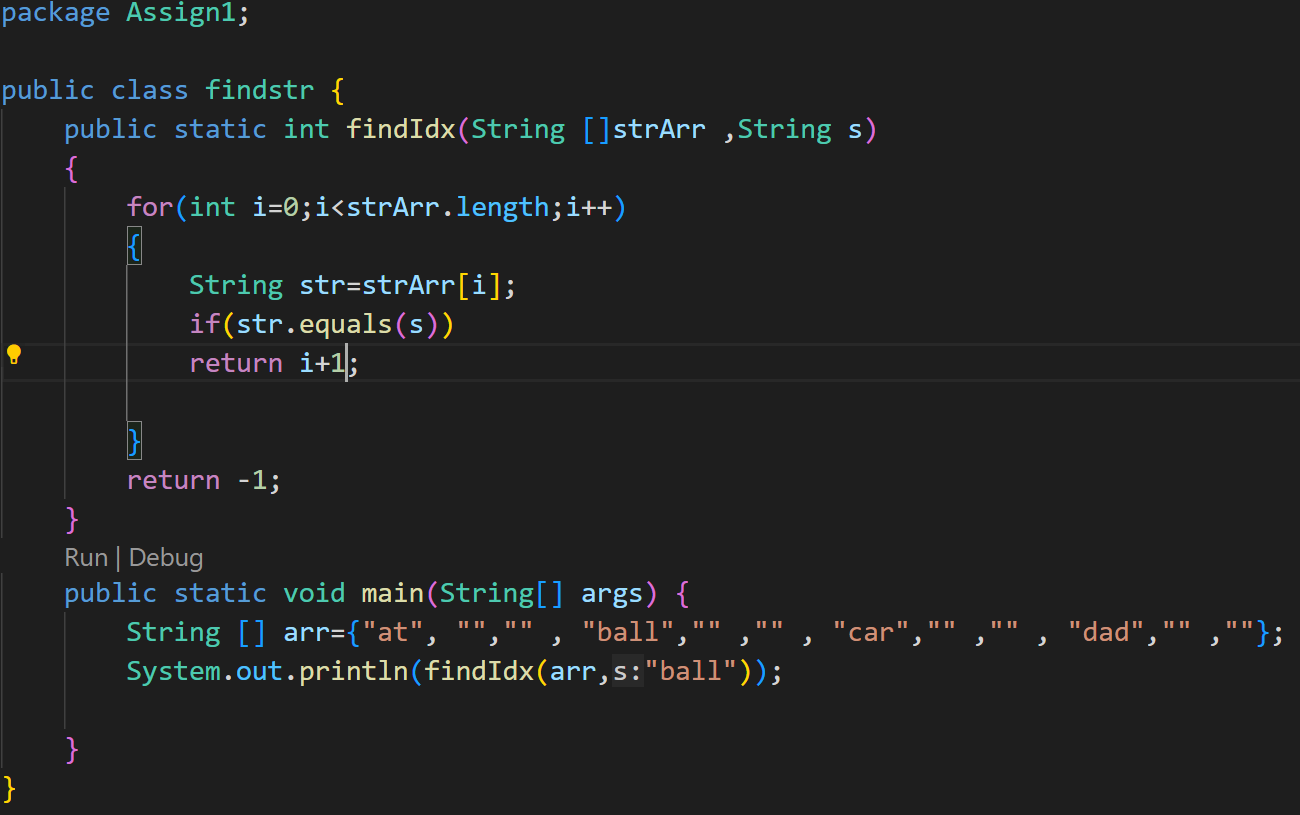
The external sorting technique allows us to sort files that are larger than the available memory by breaking them into manageable chunks that can be sorted and merged efficiently.

Q) Given a sorted array of string which is interspersed with empty string, write a method to find the location of a given string.

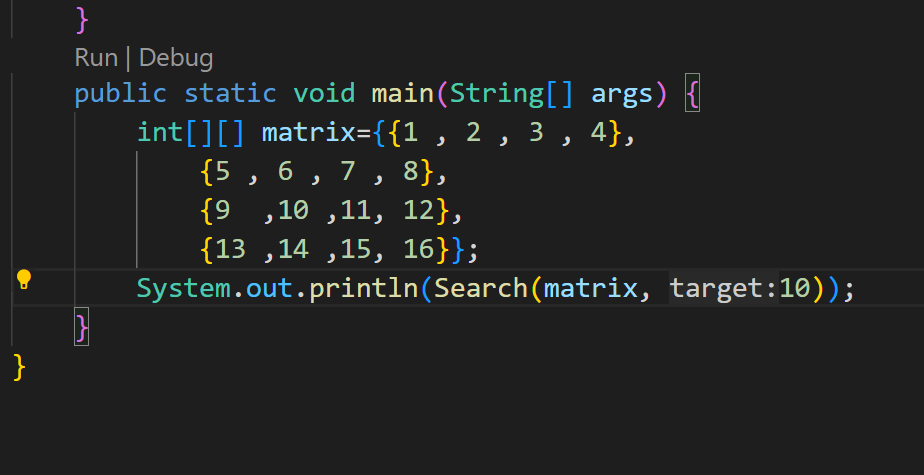
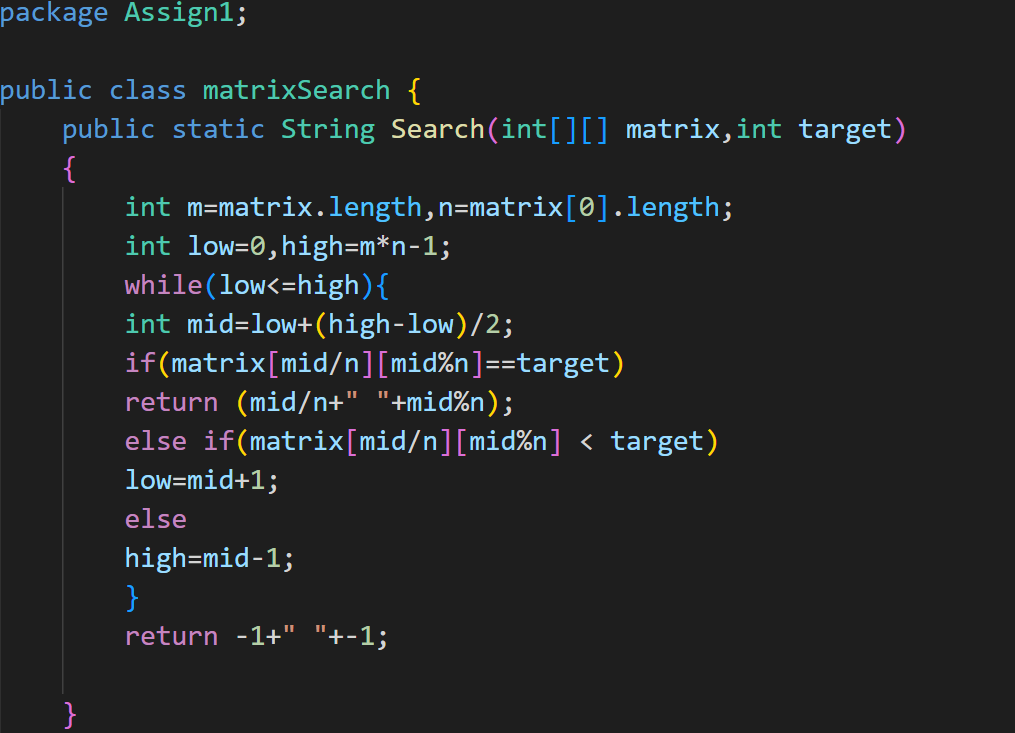
EXAMPLE

Input: find “ball” in {“at”, “”, “”, “ball”, “”, “”, “car”, “”, “”, “dad”, “”,””}

Output: 4



Q) Given an M\*N matrix in which each row and each column is sorted in ascending order, write a method to find an element.



Q) A circus is designing a tower routine consisting of people standing atop one another’s shoulders. For practical and aesthetic reasons, each person must be both shorter and lighter than the person below him or her. Given the heights and weight of each circus, write a method to compute the largest possible number of people in such tower.

EXAMPLE:

*Input(ht,wt):* (65, 100) (70, 150) (56, 90) (75,190) (60, 95) (68, 110).

Output: The longest tower is length 6 and includes from top to bottom:

(56, 90) (60, 95) (65, 100) (68, 110) (70, 150) (75, 190)

Q) Imagine you are reading in stream of integers. Periodically, you wish to be able to look up the rank of number *x* (the number of values less than or equal to *x*). Implement the data structures and algorithms to support these operations. That is, Implement the method *track (int x),* which is called when each number is generated, and the method *getRankOfNumber (int x)*, which return the number of values less than or equal to *x* (not including x itself).

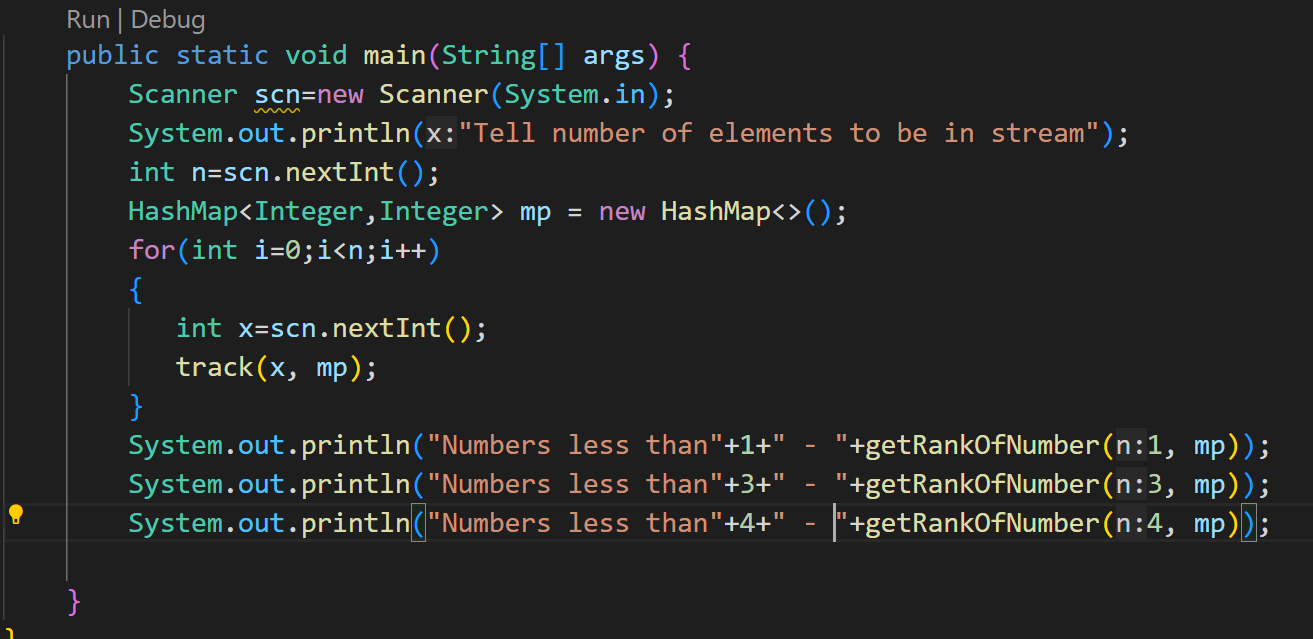
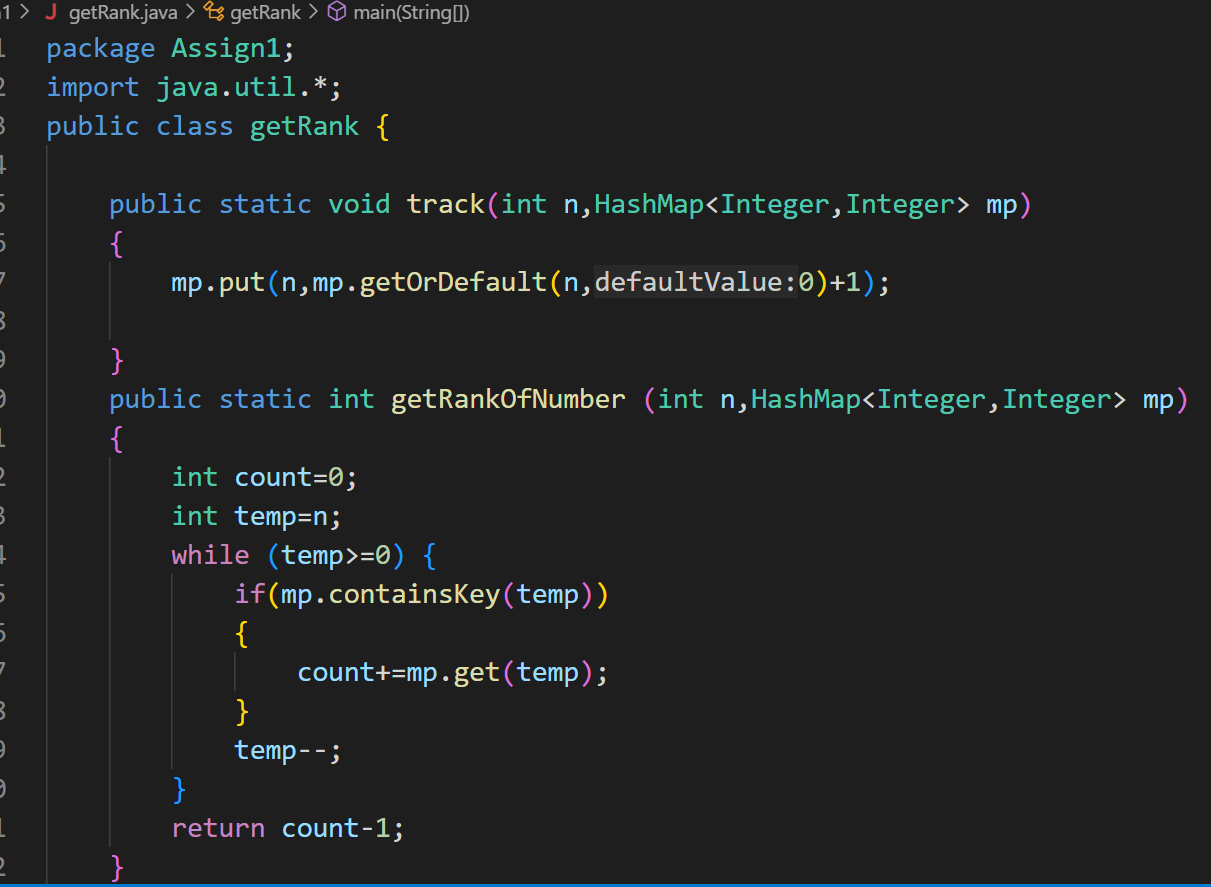
EXAMPLE

Stream (in order of appearance) : 5, 1, 4, 4, 5, 9, 7, 13, 3

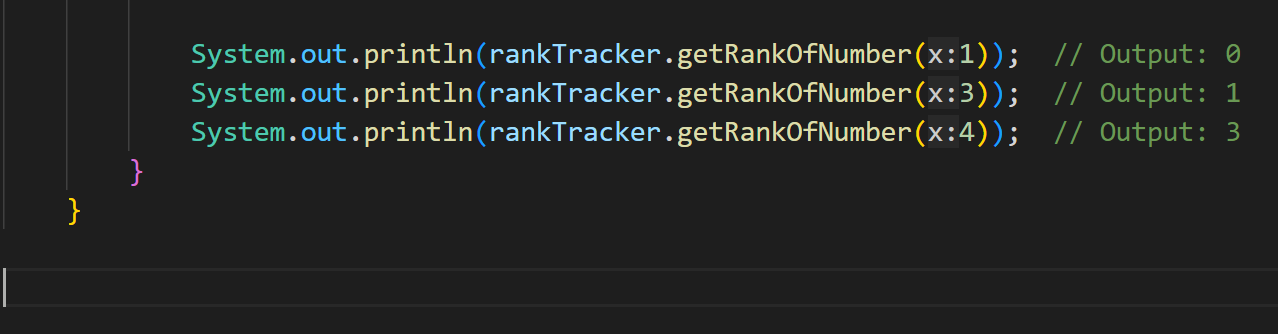
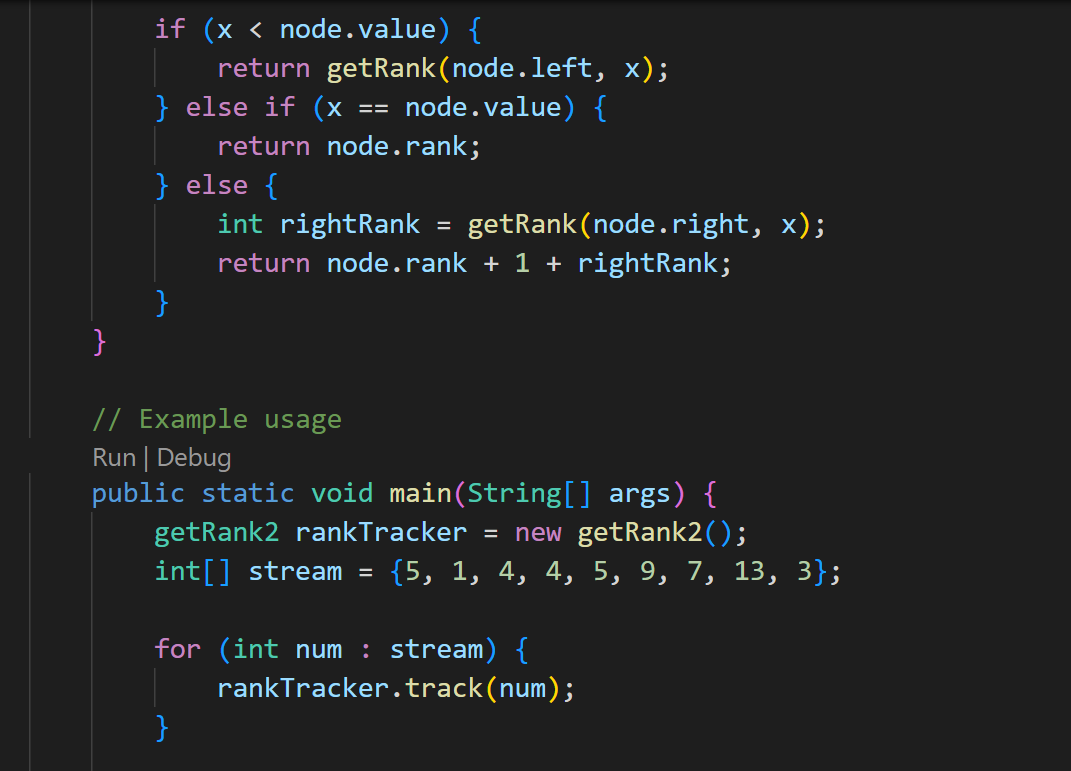
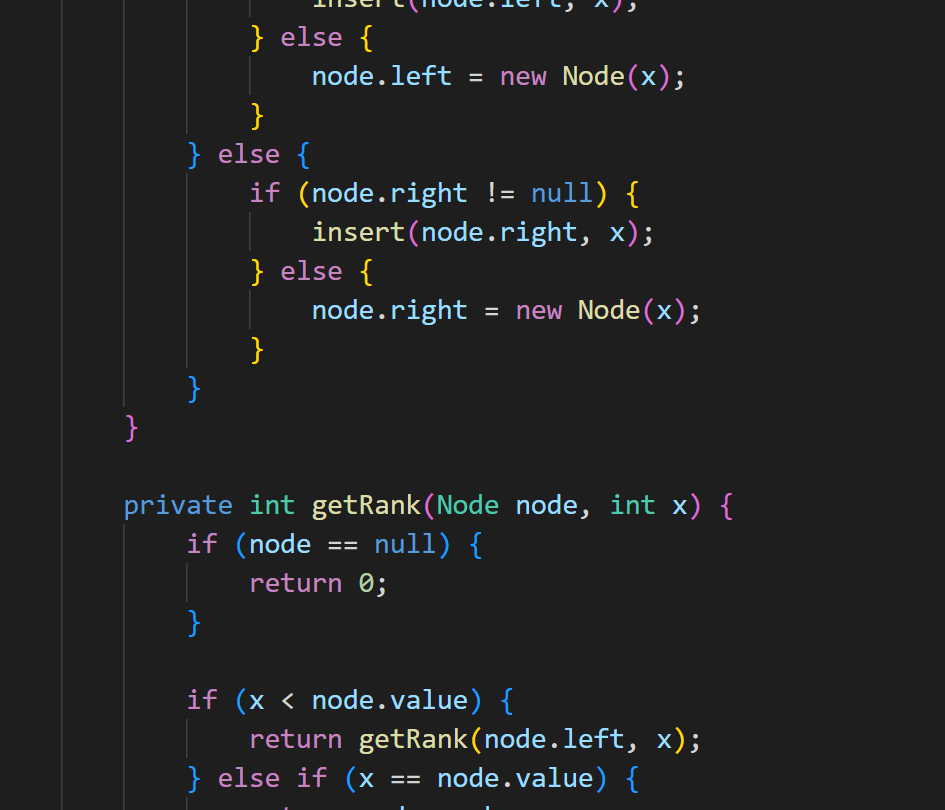
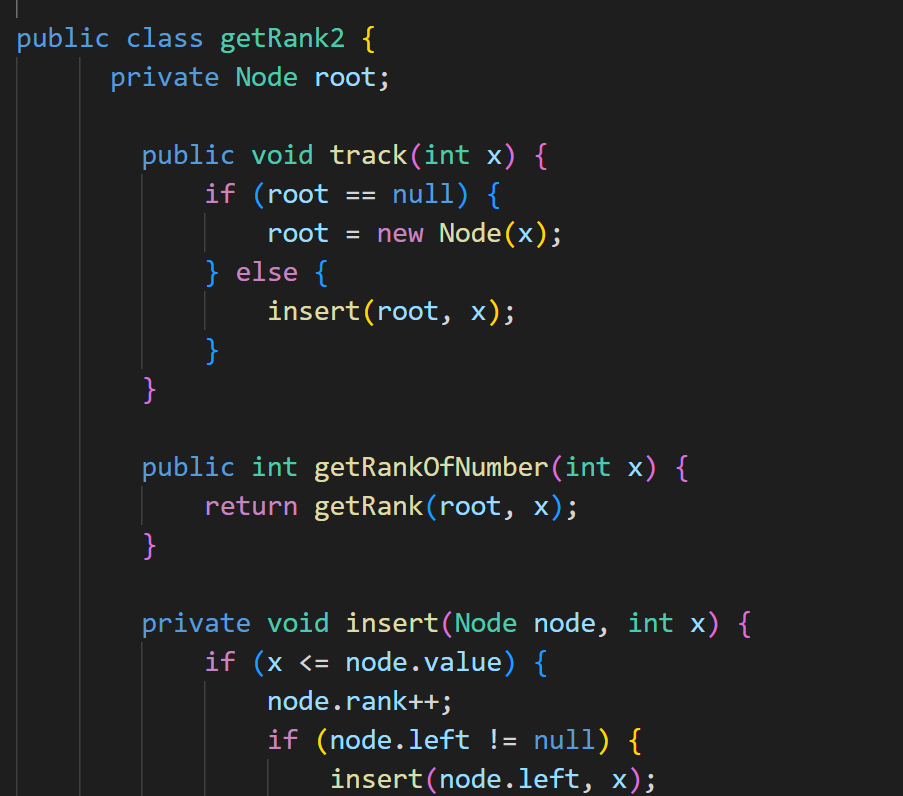
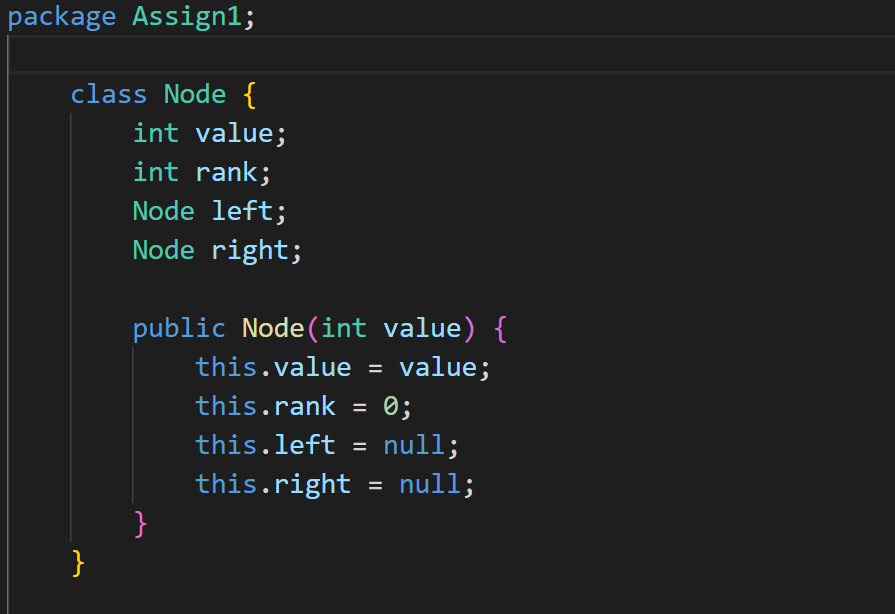
*getRankOfNumber(1) = 0*

*getRankOfNumber(3) = 1*

*getRankOfNumber(4) =3*



*Method 2-----à*



Output:

