

## Assignment 3 [ENSF 594]

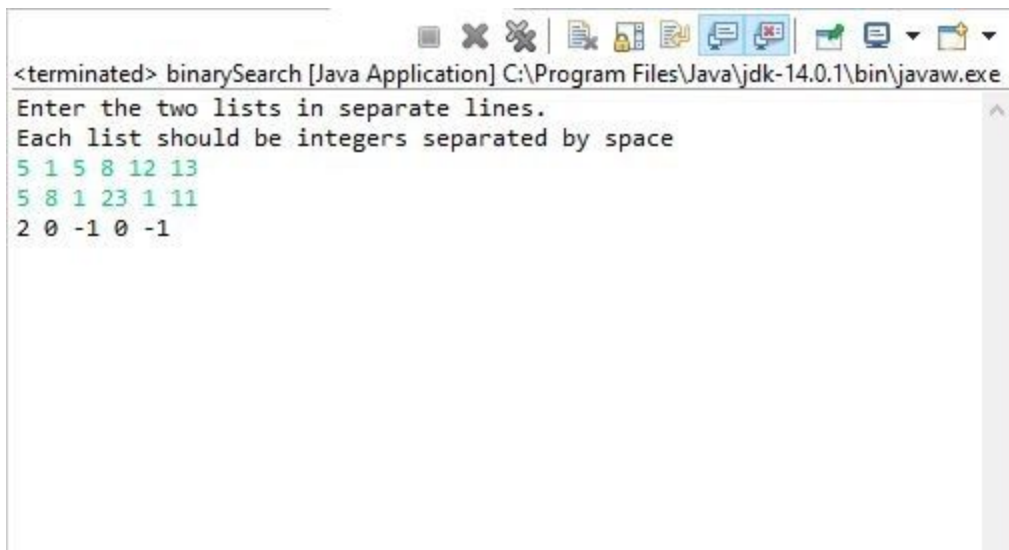
### Ans 1 - Binary Search

Binary Search algorithm is implemented for the two lists entered by the user. The elements in the second list are searched in the first list and corresponding indices are returned. I have implemented the code with a constructor and three different functions, search(), getIndex(), and printIndex().

In the constructor the input is read from the console and is assigned to the two array lists corresponding to the first and the second list. getIndex() reads the elements from the second list one by one, and passes them as a key to the search(). In 'search(int key)' the binary search algorithm is implemented and the index for that particular key is found. If the element is found then the index is returned else -1 is returned. printIndex() prints the list of all the indices.

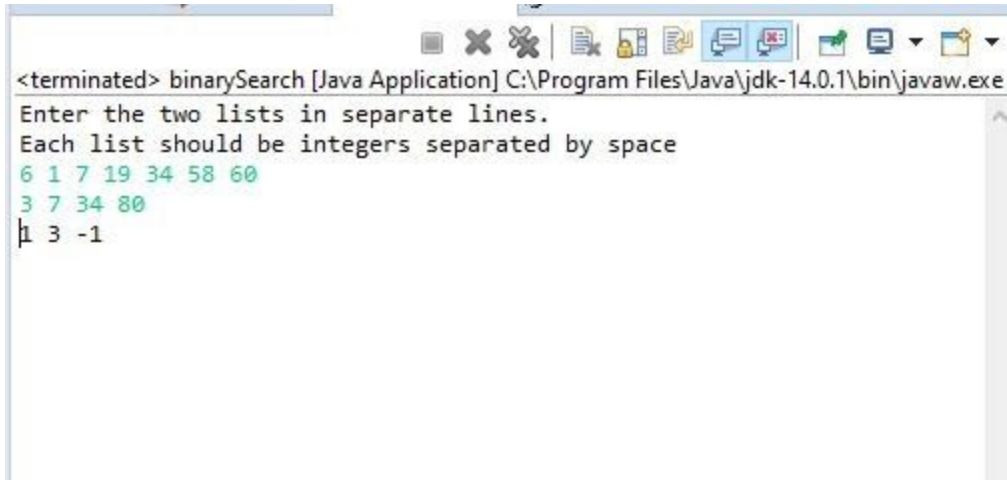
### Console Outputs

#### Positive Test Case 1



```
<terminated> binarySearch [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe
Enter the two lists in separate lines.
Each list should be integers separated by space
5 1 5 8 12 13
5 8 1 23 1 11
2 0 -1 0 -1
```

## Positive Test Case 2



```
<terminated> binarySearch [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe
Enter the two lists in separate lines.
Each list should be integers separated by space
6 1 7 19 34 58 60
3 7 34 80
1 3 -1
```

## Ans 2 - Majority Element

Majority Element finds the element that is present in the data for more than half the length of the data. The algorithm used to do so is divide and conquer.

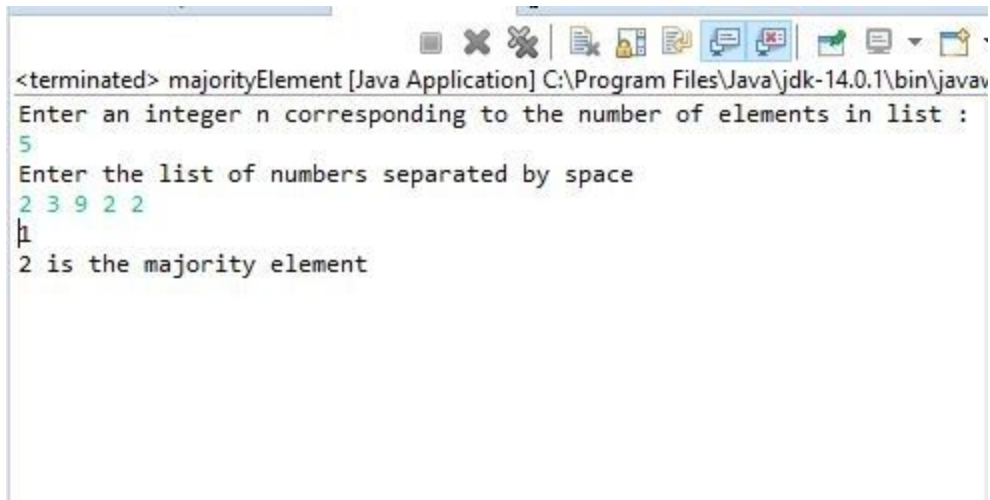
The array is divided into two halves recursively and the majority element is found in each of the divided arrays. The majority element found in the divided array is then passed above the stack where it is again checked whether it is still in majority. This process is repeated until we find the majority element. If there is no majority element then the -1 is passed.

The final output displays 1 or 0 based on if any majority element is found or not.

The time complexity of the implemented algorithm is  $n \log n$  as it solves the problem recursively.

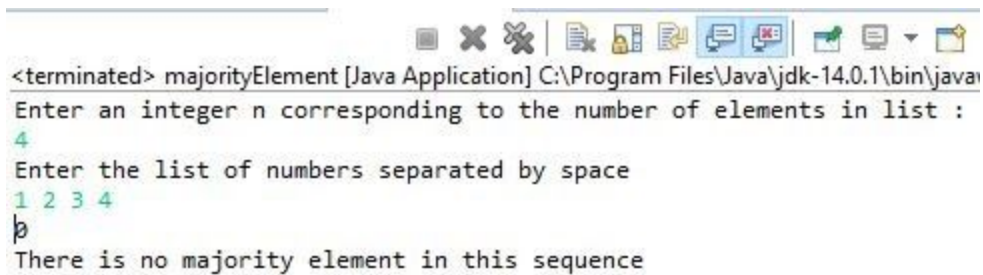
## Console Outputs

### Test Case 1



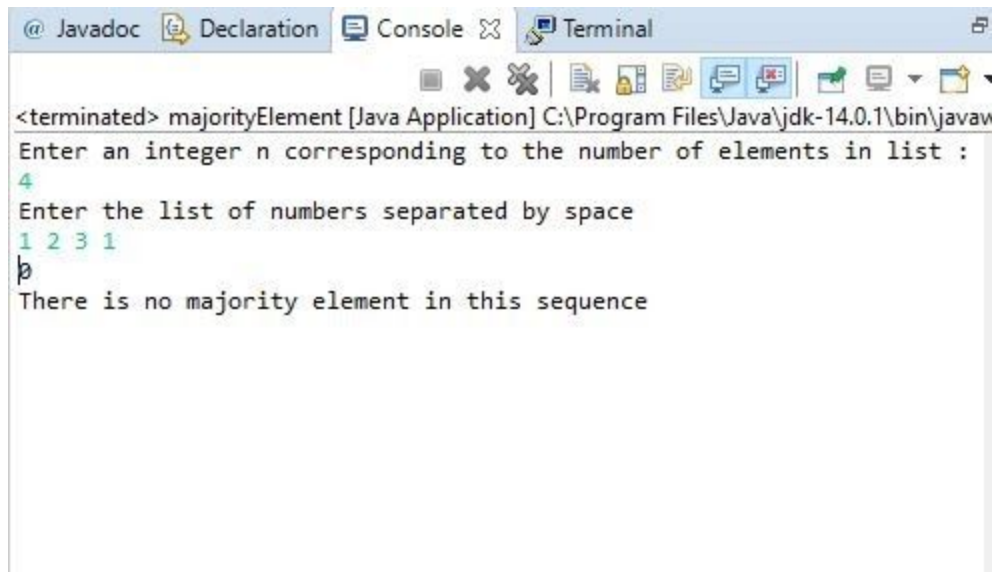
```
<terminated> majorityElement [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\java
Enter an integer n corresponding to the number of elements in list :
5
Enter the list of numbers separated by space
2 3 9 2 2
1
2 is the majority element
```

### Test Case 2



```
<terminated> majorityElement [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\java
Enter an integer n corresponding to the number of elements in list :
4
Enter the list of numbers separated by space
1 2 3 4
0
There is no majority element in this sequence
```

### Test Case 3



The screenshot shows a Java IDE with a terminal window. The terminal title bar includes tabs for '@ Javadoc', 'Declaration', 'Console', and 'Terminal'. The terminal output shows the execution of a Java application named 'majorityElement'. It prompts the user to enter an integer 'n' and a list of numbers. The user enters '4' and '1 2 3 1' respectively. The application then outputs 'There is no majority element in this sequence'.

```
<terminated> majorityElement [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw
Enter an integer n corresponding to the number of elements in list :
4
Enter the list of numbers separated by space
1 2 3 1
There is no majority element in this sequence
```