List 1: Learning and Practice (Focus on Understanding and Applying Patterns)

1.Basic Binary Search Template

Code:

```
public int binarySearch(int[] nums, int target) {
int left = 0, right = nums.length - 1;


   while (left <= right) {        int mid = left + (right -
left) / 2;  // Avoid overflow        if (nums[mid] ==
target) {            return mid;  // Target found        } else
if (nums[mid] < target) {            left = mid + 1;  //
Target is in the right half
       } else {
          right = mid - 1;  // Target is in the left half
       }
   }


   return -1;  // Target not found
}
```

Question

1. **Search in Rotated Sorted Array - LeetCode 33**
2. **Find First and Last Position of Element in Sorted Array - LeetCode 34**
3. **First Bad Version - LeetCode 278**
4. **Find Peak Element - LeetCode 162**
5. **Search a 2D Matrix - LeetCode 74 6. Search Insert Position - LeetCode 35**
7.

2. Binary Search on Answer (Minimize/Maximize)

Template Code:

```
public int binarySearchOnAnswer(int[] nums, int k) {
int left = 1, right = nums.length;


   while (left < right) {        int mid = left +
(right - left) / 2;        if
(canSatisfyCondition(nums, mid, k)) {
right = mid;  // Look for a smaller value
       } else {
          left = mid + 1;  // Increase the value
       }
   }


   return left;
}


// Example: Function to check if condition is satisfied private
boolean canSatisfyCondition(int[] nums, int mid, int k) {
   // Implement condition logic here     return true;
// Modify as per problem constraints
}
```

Question:

1. **Koko Eating Bananas - LeetCode 875**
2. **Capacity To Ship Packages Within D Days - LeetCode 1011**
3. **Split Array Largest Sum - LeetCode 410**
4. **Magnetic Force Between Two Balls - LeetCode 1552**
5. **Minimize Maximum Distance to Gas Station - LeetCode 774**
6. **Minimum Speed to Arrive on Time - LeetCode 1870**

must revision when you going to give interview

3. First and Last Occurrence (Binary Search for Bounds)

Template Code

Template For First Occurance: public int findFirstOccurrence(int[] nums, int target) { int left = 0, right = nums.length - 1; int result = -1; // Track the first occurrence

```
   while (left <= right) {        int mid = left + (right - left) / 2;        if (nums[mid] == target) {
          result = mid;  // Track the first occurrence
right = mid - 1;  // Narrow the search to the left side
      } else if (nums[mid] < target) {
left = mid + 1;
      } else {
right = mid - 1;
      }
   }

   return result;
}
```

Template For Last Occurance:

```
public int findLastOccurrence(int[] nums, int target) {
   int left = 0, right = nums.length - 1;
   int result = -1;  // Track the last occurrence

   while (left <= right) {        int mid
= left + (right - left) / 2;        if
(nums[mid] == target) {
          result = mid;  // Track the last occurrence            left
= mid + 1;  // Narrow the search to the right side
       } else if (nums[mid] < target) {
          left = mid + 1;
       } else {
right = mid - 1;
       }
   }

   return result;
}
```

Question:

1. **Find First and Last Position of Element in Sorted Array - LeetCode 34**
2. **Time Based Key-Value Store - LeetCode 981**
3. **Find Peak Element - LeetCode 162**
4. **Find Smallest Letter Greater Than Target - LeetCode 744**

5.Search in Rotated Sorted Array

Template Code:

```
public int searchInRotatedSortedArray(int[] nums, int target) {
   int left = 0, right = nums.length - 1;
while (left <= right) {
      int mid = left + (right - left) / 2;

      if (nums[mid] == target) {
return mid;
      }
 // Check if the left half is sorted
if (nums[left] <= nums[mid]) {
         if (nums[left] <= target && target < nums[mid]) {
right = mid - 1;  // Target is in the left half
         } else {
            left = mid + 1;  // Target is in the right half
         }
      }
      // Right half must be sorted
      else {
         if (nums[mid] < target && target <= nums[right]) {
            left = mid + 1;  // Target is in the right half
         } else {
            right = mid - 1;  // Target is in the left half
         }
      }
   }

   return -1;  // Target not found
}
```

Question:

1. **Find Minimum in Rotated Sorted Array - LeetCode 153**
2. **Search in Rotated Sorted Array II - LeetCode 81**
3. **Find Peak Element - LeetCode 162**

4.Binary Search in a Matrix

```
public boolean searchMatrix(int[][] matrix, int target) {     if (matrix
== null || matrix.length == 0 || matrix[0].length == 0) {        return
false;
   }

   int rows = matrix.length, cols = matrix[0].length;
   int left = 0, right = rows * cols - 1;

   while (left <= right) {        int mid
= left + (right - left) / 2;
      int midValue = matrix[mid / cols][mid % cols];

      if (midValue == target) {
return true;
      } else if (midValue < target) {
         left = mid + 1;
} else {           right =
mid - 1;
      }
   }

   return false;
}
```

Question:

1. **Search a 2D Matrix - LeetCode 74**
2. **Kth Smallest Element in a Sorted Matrix - LeetCode 378**
3. **Search a 2D Matrix II - LeetCode 240**

5. Binary Search for Float/Double Precision Template

Code:

```
public double binarySearchFloat(double left, double right) {
   double precision = 1e-6;  // Define the precision limit

   while (right - left > precision) {
      double mid = left + (right - left) / 2.0;

      if (meetsCondition(mid)) {
         right = mid;  // Move towards the smaller value
      } else {
         left = mid;  // Move towards the larger value
      }
   }

   return left;  // or return right, as they will be very close
}

// Condition to check
private boolean meetsCondition(double mid) {
   // Example condition: mid meets the criteria for the problem
return true;  // Implement logic here
}
```

Question:

6. **Find the Square Root - LeetCode 69**
7. **Median of Two Sorted Arrays - LeetCode 4**
8. **Minimize Maximum Distance to Gas Station - LeetCode 774**

List 2: Practice (Consolidate Learning by Solving More Problems)

These are practice problems that you should attempt after going through List 1 to reinforce the concepts and patterns.

1. Basic Binary Search

1. **H-Index II - LeetCode 275**
2. **Guess Number Higher or Lower II - LeetCode 375**
3. **Intersection of Two Arrays II - LeetCode 350**
4. **Count of Smaller Numbers After Self - LeetCode 315**

2. Binary Search on Answer

1. **Divide Chocolate - LeetCode 1231**
2. **Aggressive Cows - (Classic Competitive Programming Problem)**
3. **Magnetic Force Between Two Balls - LeetCode 1552**
4. **Find Kth Smallest Pair Distance - LeetCode 719**

3. First and Last Occurrence

1. **Find Right Interval - LeetCode 436**
2. **Find Peak Element II - LeetCode 1901**
3. **Closest Binary Search Tree Value II - LeetCode 272**
4. **Smallest Good Base - LeetCode 483**

4. Search in Rotated Sorted Array

1. **Find Minimum in Rotated Sorted Array II - LeetCode 154**
2. **Search in Rotated Sorted Array II - LeetCode 81**
3. **Find Kth Largest Element in an Array - LeetCode 215**

5. Matrix and Multi-dimensional Search

1. **Matrix Cells in Distance Order - LeetCode 1030**
2. **Kth Smallest Element in a Sorted Matrix - LeetCode 378 3. Falling Squares - LeetCode 699**
4.

6. Advanced Binary Search

1. **Russian Doll Envelopes - LeetCode 354**
2. **Skyline Problem - LeetCode 218**
3. **Kth Smallest Element in a BST - LeetCode 230** **How to Approach These Lists:**

List 1 is for learning and practicing key binary search patterns. Start with this list to understand the fundamental use cases of binary search, such as searching in sorted arrays, working with rotated arrays, handling range queries, and applying binary search on answer.

List 2 is purely for practice. Once you've grasped the techniques in List 1, use the second list to solidify your knowledge. These problems will help you become more proficient in recognizing when and how to apply binary search to different types of problems.

Conceptual but very much Important

```
public class InfiniteArraySearch {

    // Function to perform binary search within the identified bounds
    public static int binarySearch(int[] arr, int low, int high, int target) {
        while (low <= high) {
            int mid = low + (high - low) / 2;

            // Check if the target is at mid
            if (arr[mid] == target) {
                return mid;
            }

            // If target is greater, ignore the left half
            if (arr[mid] < target) {
                low = mid + 1;
            }
            // If target is smaller, ignore the right half
            else {
                high = mid - 1;
            }
        }
        // Target is not present in the array
        return -1;
    }

    // Function to find the range and call binary search
    public static int findPositionInInfiniteArray(int[] arr, int target) {
        // Start with a range of [0, 1]
        int low = 0;
        int high = 1;

        // Expand the range exponentially until the target is within the range
        while (arr[high] < target) {
            low = high;         // Move low to the current high
            high = 2 * high;    // Double the range size
        }
```

```
        // Perform binary search within the found range
        return binarySearch(arr, low, high, target);
    }

    public static void main(String[] args) {
        // Example of an infinite array (in reality, we use a large enough array)
        int[] arr = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31};

        // The target element we are searching for
        int target = 15;

        // Call the function to find the position of the target in the infinite array
        int result = findPositionInInfiniteArray(arr, target);

        // Output the result
        if (result != -1) {
            System.out.println("Target found at index: " + result);
        } else {
            System.out.println("Target not found.");
        }
    }
}
```