Program to find the longest increasing subsequence from a list of random numbers

```
package com.longest.increasing.subsequence;
import java.util.ArrayList;
import java.util.List;

public class LongestApp {

    // Method 1: Dynamic Programming Approach
    public static List<Integer> findLongestIncreasingSubsequenceDP(int[] nums)
{
        if (nums == null || nums.length == 0) {
            return new ArrayList<>();
        }
        int n = nums.length;
        int[] dp = new int[n];
        int[] prev = new int[n];
        for (int i = 0; i < n; i++) {
            dp[i] = 1;
            prev[i] = -1;
            for (int j = 0; j < i; j++) {
                if (nums[i] > nums[j] && dp[i] < dp[j] + 1) {
                    dp[i] = dp[j] + 1;
                    prev[i] = j;
            }
        int maxLength = 0;
        int endIndex = 0;
```

```
        for (int i = 0; i < n; i++) {
            if (dp[i] > maxLength) {
                maxLength = dp[i];
                endIndex = i;
            }
}
        List<Integer> longestIncreasingSubsequence = new ArrayList<>();
        while (endIndex != -1) {
            longestIncreasingSubsequence.add(nums[endIndex]);
            endIndex = prev[endIndex];
        }

        // Reverse the list to get the actual LIS
        List<Integer> result = new ArrayList<>();
        for (int i = longestIncreasingSubsequence.size() - 1; i >= 0; i--) {
            result.add(longestIncreasingSubsequence.get(i));
        }

        return result;
    }

    // Method 2: Brute-Force Approach
    public static List<Integer> findLongestIncreasingSubsequenceBF(int[] nums)
{
        if (nums == null || nums.length == 0) {
            return new ArrayList<>();
        }
```

```
        List<Integer> currentSubsequence = new ArrayList<>();
        List<Integer> longestIncreasingSubsequence = new ArrayList<>();

        findLIS(nums, 0, currentSubsequence, longestIncreasingSubsequence);

        return longestIncreasingSubsequence;
    }

    private static void findLIS(int[] nums, int currentIndex, List<Integer>
currentSubsequence,
            List<Integer> longestIncreasingSubsequence) {
        if (currentIndex == nums.length) {
            if (currentSubsequence.size() > longestIncreasingSubsequence.size()) {
                longestIncreasingSubsequence.clear();
                longestIncreasingSubsequence.addAll(currentSubsequence);
            }
            return;
        }

        if (currentSubsequence.isEmpty() || nums[currentIndex] >
currentSubsequence.get(currentSubsequence.size() - 1)) {
            currentSubsequence.add(nums[currentIndex]);
            findLIS(nums, currentIndex + 1, currentSubsequence,
longestIncreasingSubsequence);
            currentSubsequence.remove(currentSubsequence.size() - 1);
        }

        findLIS(nums, currentIndex + 1, currentSubsequence,
longestIncreasingSubsequence);
```

```
    }
    public static void main(String[] args) {
        int[] nums = { 22,10, 22, 9, 33, 21, 50, 41, 60, 80 };


        System.out.println("Method 1 (Dynamic Programming) - Longest
Increasing Subsequence:");
        List<Integer> lisDP = findLongestIncreasingSubsequenceDP(nums);
        System.out.println(lisDP);


        Integer n = lisDP.size();
        System.out.println("Method 2 (Brute Force) - Longest Increasing
Subsequence:");
        List<Integer> lisBF = findLongestIncreasingSubsequenceBF(nums);
        System.out.println(lisBF);


        System.out.println("The size of Longest incearing subsequence is : "+n);


    }
}
```

Increasing Subsequence Source Code

Main Method(Common for both)

```
public static void main(String[] args) {
    int[] nums = { 22,10, 22, 9, 33, 21, 50, 41, 60, 80 };

    System.out.println("Method 1 (Dynamic Programming) - Longest Increasing Subsequence:");
    List<Integer> lisDP = findLongestIncreasingSubsequenceDP(nums);
    System.out.println(lisDP);

    Integer n = lisDP.size();
    System.out.println("Method 2 (Brute Force) - Longest Increasing Subsequence:");
    List<Integer> lisBF = findLongestIncreasingSubsequenceBF(nums);
    System.out.println(lisBF);

    System.out.println("The size of Longest incearing subsequence is : "+n);
```

D-p approach(Method 01)

```
package com.longest.increasing.subsequence;
import java.util.ArrayList;
import java.util.List;

public class LongestApp {
 // Method 1: Dynamic Programming Approach
    public static List<Integer> findLongestIncreasingSubsequenceDP(int[] nums) {
        if (nums == null || nums.length == 0) {
            return new ArrayList<>();
        }
        int n = nums.length;
        int[] dp = new int[n];
        int[] prev = new int[n];
        for (int i = 0; i < n; i++) {
            dp[i] = 1;
            prev[i] = -1;
            for (int j = 0; j < i; j++) {
                if (nums[i] > nums[j] && dp[i] < dp[j] + 1) {
                    dp[i] = dp[j] + 1;
                    prev[i] = j;
                }
            }
        }
```

```
    int maxLength = 0;
    int endIndex = 0;

    for (int i = 0; i < n; i++) {
        if (dp[i] > maxLength) {
            maxLength = dp[i];
            endIndex = i;
        }
    }

    List<Integer> longestIncreasingSubsequence = new ArrayList<>();
    while (endIndex != -1) {
        longestIncreasingSubsequence.add(nums[endIndex]);
        endIndex = prev[endIndex];
    }
    // Reverse the list to get the actual LIS
    List<Integer> result = new ArrayList<>();
    for (int i = longestIncreasingSubsequence.size() - 1; i >= 0; i--) {
        result.add(longestIncreasingSubsequence.get(i));
    }

    return result;
}
```

Method 02

```
// Method 2: Brute-Force Approach
public static List<Integer> findLongestIncreasingSubsequenceBF(int[] nums) {
    if (nums == null || nums.length == 0) {
        return new ArrayList<>();
    }
    List<Integer> currentSubsequence = new ArrayList<>();
    List<Integer> longestIncreasingSubsequence = new ArrayList<>();

    findLIS(nums, 0, currentSubsequence, longestIncreasingSubsequence);
    return longestIncreasingSubsequence;
}
private static void findLIS(int[] nums, int currentIndex, List<Integer> currentSubsequence,
        List<Integer> longestIncreasingSubsequence) {
    if (currentIndex == nums.length) {
        if (currentSubsequence.size() > longestIncreasingSubsequence.size()) {
            longestIncreasingSubsequence.clear();
            longestIncreasingSubsequence.addAll(currentSubsequence);
        }
        return;
    }

    if (currentSubsequence.isEmpty() || nums[currentIndex] > currentSubsequence.get(currentSubsequence.size() - 1)) {
        currentSubsequence.add(nums[currentIndex]);
        findLIS(nums, currentIndex + 1, currentSubsequence, longestIncreasingSubsequence);
        currentSubsequence.remove(currentSubsequence.size() - 1);
    }
    findLIS(nums, currentIndex + 1, currentSubsequence, longestIncreasingSubsequence);
}
```