**Longest Increasing Subsequence**

**package** pack4;

**import** java.io.\*;

**import** java.lang.Math;

**import** java.util.\*;

**class** LIS {

**static** **int** LongestIncreasingSubsequenceLength(**int** v[])

{

**if** (v.length == 0) // boundary case

**return** 0;

**int**[] tail = **new** **int**[v.length];

**int** length = 1; // always points empty slot in tail

tail[0] = v[0];

**for** (**int** i = 1; i < v.length; i++) {

**if** (v[i] > tail[length - 1]) {

// v[i] extends the largest subsequence

tail[length++] = v[i];

}

**else** {

// v[i] will extend a subsequence and

// discard older subsequence

// find the largest value just smaller than

// v[i] in tail

// to find that value do binary search for

// the v[i] in the range from begin to 0 +

// length

**int** idx = Arrays.*binarySearch*(

tail, 0, length - 1, v[i]);

// binarySearch in java returns negative

// value if searched element is not found in

// array

// this negative value stores the

// appropriate place where the element is

// supposed to be stored

**if** (idx < 0)

idx = -1 \* idx - 1;

// replacing the existing subsequene with

// new end value

tail[idx] = v[i];

}

}

**return** length;

}

// Driver program to test above function

**public** **static** **void** main(String[] args)

{

**int** v[] = { 2, 5, 3, 7, 11, 8, 10, 13, 6 }; System.***out***.println(

"Length of Longest Increasing Subsequence is "

+ *LongestIncreasingSubsequenceLength*(v));

}

}

