LONGEST INCREASING SUBSEQUENCE

WHAT IS IT?

- The Longest Increasing Subsequence (LIS) is a problem in which you must find the longest possible subsequence, where all element are ordered from lowest to highest. There may be multiple answers with the same maximum length
- For example, In the sequence [3, 4, 1, 6, 6, 5, 10, 2]
- The longest increasing subsequence has a length of 4, and is

[3, 4, 5, 10] or [3, 4, 6, 10]

(both of these sequences can be found by removing elements from the original sequence, ie: the element order has not changed)

*There is a version of this problem called the longest nondecreasing subsequence in which duplicates are allowed, however I will just cover strictly increasing subsequences – ie: [3, 4, 6, 6, 10] is not allowed

OPTION 1: BRUTE FORCE

- We can calculate every subsequence, ignore all of those that are not strictly increasing, and take the largest subsequence of what is left
- As a sequence of size n has 2^n subsets, the time complexity is $O(2^n)$, which is very slow and will not be fast enough for most competitive coding usage

OPTION 1: BRUTE FORCE

- For example: in the sequence [2,5,1,7]

- The subsequences are:

[2] [5] [1] [7]

[2,5] [2,1] [2,7] [5,1] [5,7] [1,7]

[2,5,1] [2,5,7] [2,1,7] [5,1,7]

[2,5,1,7]

OPTION 1: BRUTE FORCE

- For example: in the sequence [2,5,1,7]

- The increasing subsequences are:

[2] [5] [1] [7]
[2,5] [2,7] [5,7] [1,7]
[2,5,7]

And thus, the LIS is [2,5,7] and has a length of 3

OPTION 2: USE PREVIOUS LIS

- Store the LIS ending at every point, and use the LIS ending at every element < n to calculate the LIS ending at n
- The LIS ending at element n is either just element n, or n added onto the LIS ending at some element < n
- This uses 2 for loops, making the time complexity $O(n^2)$, which is better than the previous option, however not optimal

OPTION 2: USE PREVIOUS LIS

- For example: in the sequence [2,5,1,7]

- Arr [2,5,1,7]
- LIS [1,1,1,1]

The LIS ending at every element is instantiated as 1, as it will always be >= to 1

Arr[1] > Arr[0] 5 > 2
So LIS[1] = max(LIS[1], LIS[0]+1)
So LIS[1] = 1 + 1 = 2

The LIS ending at element 2 (Arr[1]) is either itself or itself with the LIS ending at element 1

LIS [1,2,1,1]

OPTION 2: USE PREVIOUS LIS

- For example: in the sequence [2,5,1,7]

- Arr [2,5,1,7]
- LIS [1,2,1,1]

Arr[2] < Arr[0] 1 <= 2

So we do nothing – as 1 < 2, 1 could not Be added onto any LIS that ended at 2

The LIS ending at element 3 (Arr[2]) is either itself or itself with the LIS ending at element 1 or 2

LIS [1,2,1,1]

OPTION 2: USE PREVIOUS LIS

- For example: in the sequence [2,5,1,7]

- Arr [2,5,1,7]
- LIS [1,2,1,1]

Arr[2] < Arr[1] 1 <= 5

So we do nothing

LIS [1,2,1,1]

OPTION 2: USE PREVIOUS LIS

- For example: in the sequence [2,5,1,7]

- Arr [2,5,1,7]
- LIS [1,2,1,1]

Arr[3] > Arr[0] 7 > 2
So LIS[3] = max(LIS[3], LIS[0]+1)
So LIS[3] = 1 + 1 = 2

LIS [1,2,1,2]

The LIS ending at element 4 (Arr[3]) is either itself or itself with the LIS ending at element 1, 2 or 3

OPTION 2: USE PREVIOUS LIS

- For example: in the sequence [2,5,1,7]

- Arr [2,5,1,7]
- LIS [1,2,1,2]

Arr[3] > Arr[1] 7 > 5
So LIS[3] = max(LIS[3], LIS[1]+1)
So LIS[3] = 2 + 1 = 3

LIS [1,2,1,3]

OPTION 2: USE PREVIOUS LIS

- For example: in the sequence [2,5,1,7]

- Arr [2,5,1,7]
- LIS [1,2,1,3]

Arr[3] > Arr[2] 7 > 1

So LIS[3] = max(LIS[3], LIS[2]+1)

So LIS[3] = 3

LIS [1,2,1,3]

Thus, the LIS will be the largest element in the LIS array, which is 3

OPTION 2: USE PREVIOUS LIS

```
#include <bits/stdc++.h>
using namespace std;

//the array of values, and the size of the array
int lis(int arr[], int n) {

	int lis[n];

	//Initialize all LIS values to 1
	for (int i = 0; i < n; i++) {
		lis[i] = 1;
	}

	for (int i = 1; i < n; i++) {
		for (int k = 0; k < i; k++) {
			if (arr[i] > arr[k]) {
				lis[i] = max(lis[i], lis[k] + 1);
			}
		}
	}

	//returns the max element of the array lis
	return *max_element(lis, lis + n);
}
```

```
int main() {
	int arr[] = { 10, 22, 9, 33, 21, 50, 41, 60 };
	int n = sizeof(arr) / sizeof(arr[0]);
	cout << "Length of the LIS is " << lis(arr, n) << "\n";
	return 0;
}
```

OPTION 3: ACTIVE SUBSEQUENCES

- We store the last number of all active Increasing Subsequences (IS) in an array (tail[]) - (in the IS [1,4,5], tail[k] = 5)
- If the next number (arr[i]) is greater than any before it, we clone the largest sequence and add the new element to it (but in practice, the element is just stored in tail[m+1] where m was the entry of the last greatest number)
- If it is not the greatest, we find the smallest element >= to it, and replace that element (tail[k]) with said number (arr[i]) – done through binary search
- This uses 1 for loop as well as n binary searches, making the time complexity O(nlogn), which is optimal
- Binary search can be used, as every time a new sequence is created, its final element is the largest element in tail[] (thus tail[] is always sorted smallest to largest)

OPTION 3: ACTIVE SUBSEQUENCES

- Active subsequences are all the subsequences that could be used in the optimal LIS, all different subsequence lengths
- For example: in [2,3,6,8…], the 2 subsequences of length 3 are [2,3,6] and [2,3,8], however [2,3,6] is the active subsequence of length 3, as it is always optimal to use the subsequence with smaller values (if the next number was 7, only the [2,3,6] subsequence could include it)
- There will always be at most 1 active sequence for each subsequence length

OPTION 3: ACTIVE SUBSEQUENCES

- For example: in the sequence [2,5,1,7,6,3,9,12,10,11,8,6]

- Arr [2,5,1,7,6,3,9,12,10,11,8,6]
- tail [2,0,0,0,0,0,0,0,0,0,0,0] length=1

Active sequences:

2

OPTION 3: ACTIVE SUBSEQUENCES

- For example: in the sequence [2,5,1,7,6,3,9,12,10,11,8,6]

- Arr [2,5,1,7,6,3,9,12,10,11,8,6]
- tail [2,5,0,0,0,0,0,0,0,0,0,0] length=2

Active sequences:

2

2, 5

OPTION 3: ACTIVE SUBSEQUENCES

- For example: in the sequence [2,5,1,7,6,3,9,12,10,11,8,6]

- Arr [2,5,1,7,6,3,9,12,10,11,8,6]
- tail [1,5,0,0,0,0,0,0,0,0,0,0] length=2

Active sequences:

~~2~~ 1

2, 5

OPTION 3: ACTIVE SUBSEQUENCES

- For example: in the sequence [2,5,1,7,6,3,9,12,10,11,8,6]

- Arr [2,5,1,7,6,3,9,12,10,11,8,6]
- tail [1,5,7,0,0,0,0,0,0,0,0,0] length=3

Active sequences:

1

2, 5

2, 5, 7

OPTION 3: ACTIVE SUBSEQUENCES

- For example: in the sequence [2,5,1,7,6,3,9,12,10,11,8,6]

- Arr [2,5,1,7,6,3,9,12,10,11,8,6]
- tail [1,5,6,0,0,0,0,0,0,0,0,0] length=3

Active sequences:

1

2, 5

~~2, 5, 7~~ 2, 5, 6

OPTION 3: ACTIVE SUBSEQUENCES

- For example: in the sequence [2,5,1,7,6,3,9,12,10,11,8,6]

- Arr [2,5,1,7,6,3,9,12,10,11,8,6]
- tail [1,3,6,0,0,0,0,0,0,0,0,0] length=3

Active sequences:

1

~~2, 5~~ 2, 3

2, 5, 6

OPTION 3: ACTIVE SUBSEQUENCES

- For example: in the sequence [2,5,1,7,6,3,9,12,10,11,8,6]

- Arr [2,5,1,7,6,3,9,12,10,11,8,6]
- tail [1,3,6,9,0,0,0,0,0,0,0,0] length=4

Active sequences:

1

2, 3

2, 5, 6

2, 5, 6, 9

OPTION 3: ACTIVE SUBSEQUENCES

- For example: in the sequence [2,5,1,7,6,3,9,12,10,11,8,6]

- Arr [2,5,1,7,6,3,9,12,10,11,8,6]
- tail [1,3,6,9,12,0,0,0,0,0,0,0] length=5

Active sequences:

1

2, 3

2, 5, 6

2, 5, 6, 9

2, 5, 6, 9, 12

OPTION 3: ACTIVE SUBSEQUENCES

- For example: in the sequence [2,5,1,7,6,3,9,12,10,11,8,6]

- Arr [2,5,1,7,6,3,9,12,10,11,8,6]
- tail [1,3,6,9,10,0,0,0,0,0,0,0] length=5

Active sequences:

1

2, 3

2, 5, 6

2, 5, 6, 9

~~2, 5, 6, 9, 12~~ 2, 5, 6, 9, 10

OPTION 3: ACTIVE SUBSEQUENCES

- For example: in the sequence [2,5,1,7,6,3,9,12,10,11,8,6]

- Arr [2,5,1,7,6,3,9,12,10,11,8,6]
- tail [1,3,6,9,10,11,0,0,0,0,0,0] length=6

Active sequences:

1

2, 3

2, 5, 6

2, 5, 6, 9

2, 5, 6, 9, 10

2, 5, 6, 9, 10, 11

OPTION 3: ACTIVE SUBSEQUENCES

- For example: in the sequence [2,5,1,7,6,3,9,12,10,11,8,6]

- Arr [2,5,1,7,6,3,9,12,10,11,8,6]
- tail [1,3,6,8,10,11,0,0,0,0,0,0] length=6

Active sequences:

1

2, 3

2, 5, 6

~~2, 5, 6, 9~~ 2, 5, 6, 8

2, 5, 6, 9, 10

2, 5, 6, 9, 10, 11

OPTION 3: ACTIVE SUBSEQUENCES

- For example: in the sequence [2,5,1,7,6,3,9,12,10,11,8,6]

- Arr [2,5,1,7,6,3,9,12,10,11,8,6]
- tail [1,3,6,8,10,11,0,0,0,0,0,0] length=6

Active sequences:

1

2, 3

~~2, 5, 6~~ 2, 3, 6

2, 5, 6, 8

2, 5, 6, 9, 10

2, 5, 6, 9, 10, 11

As you can see here, when we replace the 6 in tail[2] for 6, what we are actually doing is adding a 6 to the Active Subsequence of length 2 (so the new sequence is [2,3,6] and not [2,5,6])

OPTION 3: ACTIVE SUBSEQUENCES

- For example: in the sequence [2,5,1,7,6,3,9,12,10,11,8,6]

- Arr [2,5,1,7,6,3,9,12,10,11,8,6]
- tail [1,3,6,8,10,11,0,0,0,0,0,0] length=6

Active sequences:

1

2, 3

2, 3, 6

2, 5, 6, 8

2, 5, 6, 9, 10

2, 5, 6, 9, 10, 11

There are 6 active sequences, so the LIS has a length of 6

As the algorithm (shown on the next slide) only holds the last element of every active sequence, you cannot reconstruct the LIS without changing the algorithm

OPTION 3: ACTIVE SUBSEQUENCES

```
#include <bits/stdc++.h>
using namespace std;

int lis(int arr[], int n) {

	int tail[n]; //holds the last element of lis sequences
	int length = 1; //points to the next empty slot in tail

	tail[0] = arr[0];
	for (int i = 1; i < n; i++) {

		//index of the first number >=arr[i] in the tails array
		int index = lower_bound(tail, tail + length, arr[i]) - tail;

		//if no numbers >=arr[i], add it to the tail array
		if (index == length) {
			tail[length] = arr[i];
			length++;

		} else { //change the number in the tails array to the smaller or equal arr[i]
			tail[index] = arr[i];
		}
	}
	return (n==0)? 0 : length; //return 0 if the array is empty (base case)
}

int main() {
	int arr[] = { 2,5,1,7,6,3,9,12,10,11,8,6 };
	int n = sizeof(arr) / sizeof(arr[0]);
	cout << "Length of the LIS is " << lis(arr, n) << "\n";
	return 0;
}
```

EXAMPLE

- Example problem (Codeforce 486E - LIS of Sequence)
- Every number in a sequence can be put in 3 groups:

1) The number belongs to no LIS
2) The number belongs to some but not all LIS's
3) The number belongs to all LIS's

Given a sequence of numbers, print what group every number of said sequence is in

Eg: [1,3,2,9,5,6]

The answer is 322133,
As the LIS is either
[1,3,5,6] or [1,2,5,6]

EXAMPLE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;

int a[100002], dp[100001], x[100005], out[100005], MAX[100005], cnt[100005];

int main()
{
    int n;
    cin >> n;
    for (int i = 0; i < n; i++)
        cin >> a[i];
    int ans = 0;
    for(int i = 0; i < n; i++)
    {
        int lo = -1, hi = ans;
        while(lo < hi - 1)
        {
            int mid = (lo + hi) / 2;
            if(x[mid] >= a[i])
                hi = mid;
            else
                lo = mid;
        }
        dp[i] = hi + 1;
        if(hi == ans)
            x[ans++] = a[i];
        else
            x[hi] = min(x[hi], a[i]);
    }

    fill(MAX, MAX + n, -1);
    for(int i = n - 1; i >= 0; i--)
    {
        if(dp[i] == ans)
        {
            MAX[ans] = max(MAX[ans], a[i]);
            out[i] = 2;
            continue;
        }
        int x = MAX[dp[i] + 1];
        if(x == -1 || x <= a[i])
            out[i] = 1;
        else
            out[i] = 2;
        if(out[i] == 2)
            MAX[dp[i]] = max(MAX[dp[i]], a[i]);
    }
    for(int i = 0; i < n; i++)
        if(out[i] == 2)
            cnt[dp[i]]++;
    for(int i = 0; i < n; i++)
        if(out[i] == 2 && cnt[dp[i]] == 1)
            cout << 3;
        else
            cout << out[i];
    cout << endl;
    return 0;
}
```

Solution Taken from https://codeforces.com/contest/486/submission/8657105

Problem tutorial: https://codeforces.com/blog/entry/14678