Longest Increasing Subsequence

by

Jane Alam Jan

Introduction

LIS abbreviated as 'Longest Increasing Subsequence', consists of three parts.

Longest - stands for its own meaning.

Increasing - means that it must be an increasing something, for example [1, 2, 3, 7, 20] is an increasing sequence but [1, 4, 2, 5] is definitely not an increasing sequence because we have 2 after 4. Formally, a sequence $[a_1, a_2 \ldots a_n]$ is increasing if $a_1 < a_2 < a_3 < \ldots < a_n$.

Subsequence - means a new sequence consisting of some numbers from the original sequence but the numbers maintain the relative ordering. For example if the sequence is [1, 5, 2, 7, 3] then [2, 7, 3], [1, 5, 3] and [1, 2, 7] etc are valid subsequences but [1, 3, 5] is not. Cause in the original sequence 3 is listed after 5. Formally, if $[a_1, a_2 \ldots a_n]$ is a sequence then $[a_i, a_j \ldots a_z]$ will be a subsequence of the given sequence if $i < j < \ldots < z$.

So, we are given a sequence, we have to find the increasing subsequence which is longest, that means the subsequence which is increasing and contains maximum numbers.

For example a sequence is given - [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]. Now some increasing subsequences are [8, 9], [1, 9], [1, 8], [1, 3, 4, 6], [1, 3, 4, 5], [1, 2], [9], [8], [3, 4, 6], [3, 4, 5], [4, 6], [4, 5], [6], [1, 5], [1, 2], [5], [1]. Amongst them all [1, 3, 4, 6] and [1, 3, 4, 5] both are longest.

An $O(n^2)$ approach

This idea is relatively easy. For this problem we create another array where the ith item contains the longest increasing subsequence found from the first item to the ith item, and which included the ith item. Let the array be L[]. L[i] will contain the increasing subsequence length which includes the ith item.

1) If we take any item from the sequence, it's a valid increasing subsequence, but it may not be longest. So, initially we say that all the values in L[] are 1.

```
Sequence  [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L         [1, 1, 1, 1, 1, 1, 1, 1, 1, 2]
```

2) Now, we start from the leftmost item which is 8. Now we try to find all the numbers which are greater than 8 and which lie after this 8. The only item we can find is 9. Since the L[] value of 8 is 1, so, we can make a subsequence [8, 9] so, we will update the L[] value of 9 as 2. Note that we will not update the next 8 because [8, 8] is not increasing.

```
Sequence  [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L         [1, 1, 2, 1, 1, 1, 1, 1, 1, 2]
```

3) Now the next item is 1 and the L[] value is also 1. The items which are in right and greater than 1 are 9, 8, 3, 4, 6, 5, and 2. So, definitely we can make sequences [1, 9], [1, 8], [1, 3], [1, 4], [1, 6], [1, 5] and [1, 2]. So, their L[] value will be 2. Note that the L[] value of 9 is already 2, so we don't need to update it again.

```
Sequence  [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L         [1, 1, 2, 2, 2, 2, 2, 1, 2, 2]
```

4) The next item is 9 and the L[] value is 2. Since no item in right side is greater than 9, so L[] will remain same.
5) Now we have 8 whose L[] value is 2. Again no item is found to update.
6) Now there is 3 whose L[] value is 2. The items which are greater than 3 and lie in right are 4, 6 and 5. Now observe that since the L[] value of 3 is 2, that means if we take 3 we can somehow make e subsequence which contains 3 and whose length is 2. And now we have an item which is greater than 3. So, obviously their L[] value will be 3.

```
Sequence  [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L         [1, 1, 2, 2, 2, 3, 3, 1, 3, 2]
```

7) The next item is 4 whose L[] value is 3, that means we can form a subsequence with 4 which contains 3 numbers. The numbers which are greater than 4 and lie in right of 4 are 6 and 5. So, L[] value of them will be 4.

```
Sequence  [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L         [1, 1, 2, 2, 2, 3, 4, 1, 4, 2]
```

8) The next item is 6 but there is no item in right which is greater than 6.
9) The next item is 1 whose L[] value is 1. The item which is greater than 1 and in right is 5. Now watch that the L[] value of 5 is already 4, but if we add 5 after 1 then the L[] value of 5 will be 2. So, we will keep 4 since its better.
10) The next item is 5 but there is no item in right which is greater than 5.
11) And finally we have 2 and we don't have to update. So, finally

```
Sequence  [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L         [1, 1, 2, 2, 2, 3, 4, 1, 4, 2]
```

12) Now we iterate through the L[] and find the maximum element, which is the result. So, for the given sequence the result is 4.

See the code which finds the length of LIS.

```
int n; // the number of items in the sequence
int Sequence[32]; // the sequence of integers
int L[32]; // L[] as described in the algorithm

void takeInput() {
    scanf("%d", &n); // how many numbers in the sequence ?

    // take the sequence
    for( int i = 0; i < n; i++ )
        scanf("%d", &Sequence[i]);
}

int LIS() { // which runs the LIS algorithm and returns the result
    int i, j; // auxilary variables for iteration

    // initialize L[] with 1
    for( i = 0; i < n; i++ )
        L[i] = 1;

    // start from the left most item and itetare right
    for( i = 0; i < n; i++ ) {
        // for the ith item item find all items that are in right
        for( j = i + 1; j < n; j++ ) {

            if( Sequence[j] > Sequence[i] ) {
                // the item is greater than the ith item

                // so, L[j] = L[i] + 1, since jth item can be added after ith
                // item. if L[j] is already greater than or equal to L[i] + 1
                // then ignore it
                if( L[j] < L[i] + 1 )
                    L[j] = L[i] + 1;
            }
        }
    }
    // now find the item whose L[] value is maximum
    int maxLength = 0;
    for( i = 0; i < n; i++ ) {
        if( maxLength < L[i]  )
            maxLength = L[i];
    }
    // return the result
    return maxLength;
}

int main() {
    takeInput();
    int result = LIS();
    printf("The LIS length is %d\n", result);
    return 0;
}
```

Reconstructing a Longest Increasing Subsequence

Now you may want to reconstruct one of the longest increasing subsequences. The technique is quite easy.

1) Find an item whose L[] value is maximum. Let 5 be the item.

```
Sequence  [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L         [1, 1, 2, 2, 2, 3, 4, 1, 4, 2]
```

2) Now iterate to left and find an item which is less than 5 and whose L[] value is one less that means 3. So, we have 4 whose L[] value is 3 and 4 < 5.

```
Sequence  [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L         [1, 1, 2, 2, 2, 3, 4, 1, 4, 2]
```

3) Again iterate left for the next item which is less than 4 and whose L[] value is 2. WE have 3.

```
Sequence  [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L         [1, 1, 2, 2, 2, 3, 4, 1, 4, 2]
```

4) Iterate left for an item which is less than 3 and whose L[] value is 1, which is 1.

```
Sequence  [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L         [1, 1, 2, 2, 2, 3, 4, 1, 4, 2]
```

5) Since we have got a L[] value with 1 so, we can stop or we may continue but we will get none. So, a longest increasing subsequence is [1, 3, 4, 5].

If we have chosen 6 we would get [1, 3, 4, 6] as the longest increasing subsequence which is valid, too.

The following code finds a longest increasing subsequence.

```
int LisSequence[32]; // for storing the sequence

void findSequence( int maxLength ) { // finds a valid sequence
    int i, j; // variable used for iteration

    // at first find the position of the item whose L[] is maximum
    i = 0;
    for( j = 1; j < n; j++ ) {
        if( L[j] > L[i] )
            i = j;
    }

    // initialize the position in LisSequence where the items can be added.
    // observe that the data are saving from right to left!
    int top = L[i] - 1;

    // insert the item in i-th position to LisSequence
    LisSequence[top] = Sequence[i];
    top--; // is decreasing such that a new item can be added in a new place

    // now find the other valid numbers to form the sequence
    for( j = i - 1; j >= 0; j-- ) {
        if( Sequence[j] < Sequence[i] && L[j] == L[i] - 1 ) {
            // we have found a valid item so, we will save it
            i = j; // as in our algorithm
            LisSequence[top] = Sequence[i]; // stored
            top--; // decreased for new items
        }
    }

    // so, we have got the sequence, now we want to print it
    printf("LIS is");
    for( i = 0; i < maxLength; i++ ) {
        printf(" %d", LisSequence[i]);
    }
    puts("");
}

int main() {
    takeInput();
    int result = LIS();
    printf("The LIS length is %d\n", result);
    findSequence( result );
    return 0;
}
```

It's also clear that the complexity to reconstruct the sequence is **O(n)**.

An O(nlogk) Approach

From the previous sections we are sure that if we can find the L[] array, then the reconstruction is not a problem, since its complexity is linear. Here we describe another algorithm for finding L[].

1) Initially we make a new array, let the name be I[], initially all the values of the array are infinite, only the 0^{th} element contains negative infinite. The size of I[] will be total elements in the sequence + 1.
2) We iterate from left and we pick the numbers from Sequence one by one and insert them into I[]. When inserting a number, we find the position where all the numbers in left are strictly smaller than the number.
3) If we insert the numbers in this fashion, if you think a while, you will find that the numbers in I[] will always be in ascending order.
4) And another important thing is that, if a number is inserted into the ith place, and all the numbers from 1^{st} place to (i-1)th place are smaller than that, so, the L[] value of that number should be i. It can be proved inductively. However, the next example will show the procedure fully.

Let the sequence be

```
Sequence  [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
```

There are 10 elements, so, I will contain 11 elements with index from 0 to 10. So, as described, the elements of I[] will be initially,

```
I          [-i, i, i, i, i, i, i, i, i, i, i]
index        0  1  2  3  4  5  6  7  8  9 10

Sequence       [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L              [n, n, n, n, n, n, n, n, n, n]
```

Here i denotes ‘infinite’ and n denotes ‘not calculated yet’.

Now let’s insert the numbers from sequence into I[].

1) At first we have 8, so the position for 8 in I[] is 1. Since the left item is –i which is smaller than 8. So, we will assign I[1] = 8. Since 8 is inserted in 1^{st} place, so the L[] value of 8 will be 1. After the first number we will get

```
I          [-i, 8, i, i, i, i, i, i, i, i, i, i]
index        0  1  2  3  4  5  6  7  8  9 10

Sequence       [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L              [1, n, n, n, n, n, n, n, n, n]
```

2) Now we have 1, the position for 1 in I[] is 1. So, we will assign I[1] = 1 and since 1 is inserted in 1st place, the L[] value of 1 will be 1. Observe that I[1] was 8, but after this iteration 8 will be replaced by 1.

```
I          [-i, 1, i, i, i, i, i, i, i, i, i]
index        0  1  2  3  4  5  6  7  8  9 10

Sequence       [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L              [1, 1, n, n, n, n, n, n, n, n]
```

3) The next number is 9. The position for 9 in I[] is definitely the 2nd position. So, we will assign I[2] = 9, and since 9 is inserted in the second position, so the L[] value of 9 will be 2.

```
I          [-i, 1, 9, i, i, i, i, i, i, i, i]
index        0  1  2  3  4  5  6  7  8  9 10

Sequence       [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L              [1, 1, 2, n, n, n, n, n, n, n]
```

4) Now we have 8, and the position of this 8 in I[] is the second position. So, we put 8 in the second position of I[] (thus replacing 9). And the L[] value of 8 will be 2.

```
I          [-i, 1, 8, i, i, i, i, i, i, i, i]
index        0  1  2  3  4  5  6  7  8  9 10

Sequence       [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L              [1, 1, 2, 2, n, n, n, n, n, n]
```

5) The next item is 3, and the position in I[] is still the second position. So, we place 3 in the second position of I[]. And the L[] value of 3 is 2.

```
I          [-i, 1, 3, i, i, i, i, i, i, i, i]
index        0  1  2  3  4  5  6  7  8  9 10

Sequence       [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L              [1, 1, 2, 2, 2, n, n, n, n, n]
```

6) Now we have 4, and the position in I[] is 3. So, the L[] value of 4 is 3.

```
I          [-i, 1, 3, 4, i, i, i, i, i, i, i]
index        0  1  2  3  4  5  6  7  8  9 10

Sequence       [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L              [1, 1, 2, 2, 2, 3, n, n, n, n]
```

7) Next item is 6, it will be inserted in the 4th position of I[]. So, the L[] value of 6 will be 4.

```
I          [-i, 1, 3, 4, 6, i, i, i, i, i, i]
index        0  1  2  3  4  5  6  7  8  9 10

Sequence       [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L              [1, 1, 2, 2, 2, 3, 4, n, n, n]
```

8) Next item is 1, it will be inserted in the 1st position of I[]. So, the L[] value of 1 will be 1.

```
I          [-i, 1, 3, 4, 6, i, i, i, i, i, i]
index        0  1  2  3  4  5  6  7  8  9 10

Sequence       [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L              [1, 1, 2, 2, 2, 3, 4, 1, n, n]
```

9) Next item is 5, it will be inserted in the 4th position of I[]. So, the L[] value of 5 will be 4.

```
I          [-i, 1, 3, 4, 5, i, i, i, i, i, i]
index        0  1  2  3  4  5  6  7  8  9 10

Sequence       [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L              [1, 1, 2, 2, 2, 3, 4, 1, 4, n]
```

10) Last item is 2, it will be inserted in the 2nd position of I[]. So, the L[] value of 2 will be 2.

```
I          [-i, 1, 2, 4, 5, i, i, i, i, i, i]
index        0  1  2  3  4  5  6  7  8  9 10

Sequence       [8, 1, 9, 8, 3, 4, 6, 1, 5, 2]
L              [1, 1, 2, 2, 2, 3, 4, 1, 4, 2]
```

Now, see that throughout the process the values in I[] are in ascending order. And since we place a number in such a place where all the numbers in left are strictly less than the number so, the L[] value will be correct for all the numbers.

One more thing, after all the iterations are over, **don't think that I[] will also contain a sequence**. Because in the example above, see that 1, 2, 4, 5 is not the correct sequence. To get a sequence, since we already have the L[] values, we can use the reconstruction method described earlier.

Now since the values in I[] will always be in ascending order, so, to find a position of a number we can definitely use binary search. So, if the final LIS length is k and there are n numbers then

to insert a number using binary search the complexity will be `O(logk)`. Thus the overall complexity will be `O(nlogk)`, since we are inserting n items.

Here is the code which uses the above idea to find LIS.

```
const int inf = 2000000000; // a large value as infinity

int n; // the number of items in the sequence
int Sequence[32]; // the sequence of integers
int L[32]; // L[] as described in the algorithm
int I[32]; // I[] as described in the algorithm

void takeInput() {
    scanf("%d", &n); // how many numbers in the sequence ?
    for( int i = 0; i < n; i++ ) // take the sequence
        scanf("%d", &Sequence[i]);
}

int LisNlogK() { // which runs the NlogK LIS algorithm
    int i; // auxilary variable for iteration

    I[0] = -inf; // I[0] = -infinite
    for( i = 1; i <= n; i++ ) // observe that i <= n are given
        I[i] = inf; // I[1 to n] = infinite

    int LisLength = 0; // keeps the maximum position where a data is inserted

    for( i = 0; i < n; i++ ) { // iterate left to right
        int low, high, mid; // variables to perform binary search
        low = 0; // minimum position where we to put data in I[]
        high = LisLength; // the maximum position

        while( low <= high ) { // binary search to find the correct position
            mid = ( low + high ) / 2;
            if( I[mid] < Sequence[i] )
                low = mid + 1;
            else
                high = mid - 1;
        }
        // observe the binary search carefully, when the binary search ends
        // low > high and we put our item in I[low]
        I[low] = Sequence[i];
        if( LisLength < low ) // LisLength contains the maximum position
            LisLength = low;
    }

    return LisLength; // return the result
}

int main() {
    takeInput();
    int result = LisNlogK();
    printf("The LIS length is %d\n", result);
    return 0;
}
```

Discussion

The above idea can also be used for finding Longest Decreasing Subsequence or LDS. You should try to construct it yourself to understand it fully. There is another technique also. Just reverse the ordering of the numbers in Sequence[] and then run the usual LIS. If you think a bit you will find that the sequence you have found in LIS for the reversed Sequence is actually an LDS for the main Sequence.