Longest Increasing Sub-sequence in $O(NlogN)$

MD. Sakib Khan

November 2, 2018

We have seen $O(N^2)$ solution of LIS(Longest Increasing Sub-sequence) using Dynamic Programming. Now we will see the $O(NlogN)$ solution using Searching Algorithm.

Lets Say, We have an array(**Array**[]). We have to find the LIS of that array. For this, we will iterate the array from the first index and as much as the iteration done, we will save some result in an another array. (**LisTillNow**[]). Using that result we'll find the LIS.

So in this way we'll follow two rules to get the LIS :

- If all the values in **LisTillNow**[] is less than the **Array**[**i**], then we'll insert **Array**[**i**] at the end of the **LisTillNow**[] array.
- And if all the values in **LisTillNow**[] is not less than the **Array**[**i**] (Greater or equal), then the first value in **LisTillNow**[] which is greater than or equal to **Array**[**i**], will be replaced by **Array**[**i**].

Our main goal is to make the **LisTillNow**[] array Sorted and Longest.

Let's see a simulation :

$$Array[] = \{7, 10, 8, 12, 5, 5, 17, 15\}$$

Step 1:

Though **LisTillNow**[] array is empty. so we simply insert 7 into that array.. So,

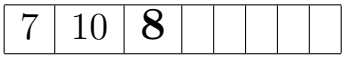
LisTillNow[] = { 7 }

Step 2:

Though All the values in **LisTillNow**[] is less than **Array[1] = 10**, we simply insert 10 at the end of the array. So,

LisTillNow[] = { 7, 10 }

Step 3:



Though all the values in **LisTillNow**[] is not less than **Array[2] = 8**, So we have to find the first value in **LisTillNow**[] which is greater than or equal to 8, will be replaced by 8. We found 10 the first value which is greater than 8. we replace 10 by 8. So,

LisTillNow[] = { 7, 8 }

Step 4:

7	10	8	**12**				

Like Step 2, we will insert **Array[3] = 12** at the at end of the **LisTillNow**[] array. So,

LisTillNow[] = { 7, 8, 12 }

Step 5:

7	10	8	12	**5**			

Like Step 3, we will replace 7 by **Array[4] = 5**. So,

LisTillNow[] = { 5, 8, 12 }

Step 6:

7	10	8	12	5	**5**		

Like Step 3, we will replace 5 by **Array[5] = 5**. So,

LisTillNow[] = { 5, 8, 12 }

Step 7:

7	10	8	12	5	5	**17**	

Like Step 2, we will insert **Array[6] = 17** at the at end of the **LisTillNow**[] array. So,

LisTillNow[] = { 7, 8, 12, 17 }

Step 8:

7	10	8	12	5	5	17	**15**

Like Step 3, we will replace 17 by **Array[7] = 15** at the at end of the **LisTillNow**[] array. So,

LisTillNow[] = { 7, 8, 12, 15 }

And it is done. :)

Following this way, we found a complete array **LisTillNow**[]. But it isn't a valid sub-sequence of the original array(**Array**[]). But the way, we get the **LisTillNow**[] array, the size of the **LisTillNow**[] array and the length of the LIS will be equal. We can easily prove that,

LisTillNow[] = { 7, 8, 12, 15 }

Here, before 15, there must be a (3 length) Increasing Sub-sequence because before iterating **Array[7] = 15**, we had already found a (3 length) Increasing Sub-sequence. which is also true for 12.Before 12, there must be a (2 length) Increasing Sub-sequence because before iterating **Array[3] = 12**, we had already found a (2 length) Increasing Sub-sequence. Think a little bit, you will understand it clearly.
So, we have found the size of LIS, but how to find the original LIS array. Not to worry, slight modification of these process, can give you the original array.

Now let's see how to code these things.
The main thing we need to do here is that, we have a value(**Array[i]**). We have to find a value which equal or immediate greater than (**Array[i]**). We can easily do it using **Binary Search** or we can use **lower_bound()** Function of STL whose work is basically these, finding a value greater than or equal to a value in a sorted container. **C++ Code:**

```
vector<int>:: iterator itr;
for(int i = 0 ; i < N ; i++) {
    itr = lower_bound(ListTillNow.begin() , ListTillNow.end() , Array[i]);
    // We are finding equal or immediate greater value than Array[i] in ListTillNow[]
    if(itr == ListTillNow.end()) ListTillNow.push_back(Array[i]);
    // Here Arrayay[i] is greater than any other value in ListTillNow[]
    else ListTillNow[itr - ListTillNow.begin()] = Array[i];
    // Here we find the proper place for Arrayay[i] in ListTillNow[].
}
cout << ListTillNow.size() << endl;
```

To get the actual LIS array, we have to store another thing. When Placing **Array[i]** in **ListTillNow[]**, if we can store that index and the index of the value placed before **Array[i]** in **ListTillNow[]** Array, then we can simply get the original Array by traversing those indexes.

C++ Code:

```
int own[N + 1];
// Here "own" Array is for the storing the index of the proper place
// or Array[i] in ListTillNow;
int prev[N + 1];
// "Prev" Array stores the index of the value placed before Array[i]
// in ListTillNow[] Array
for(int i = 0 ; i < N ; i++) {
    itr = lower_bound(ListTillNow.begin() , ListTillNow.end() , Array[i]);
    if(itr == ListTillNow.end()) {
        int len = ListTillNow.size();
        if(len) prev[i] = own[len - 1], own[len] = i;
        else prev[i] = i, own[0] = i;
        ListTillNow.push_back(Array[i]);
    }
    else {
        int idx = itr - ListTillNow.begin();
        if(idx)prev[i] = own[idx - 1], own[idx] = i;
        else prev[i] = i, own[0] = i;
        ListTillNow[idx] = Array[i];
    }
}
```

How to Print The Path? Well, just traverse from the end (Stored index wise).

C++ Code:

```
int done = own[(int)LisTillNow.size()-1];
vector<int> Path;
while(true) {
    Path.push_back(Array[done]);
    if(done == prev[done]) break;
    done = prev[done];
}
reverse(Path.begin() , Path.end());
for(int i = 0 ; i < Path.size() ; i++) {
    cout << Path[i] << " ";
}
```

Some Variations of LIS:

- Given an Array, You have to find **Longest Decreasing Sub-sequence** of that array. [Hints: Reverse the array. Now Find LIS] Also print The LDS array.
- If LIS means Longest Increasing **(Not Strictly)**, Then Find the LIS of an array. [Hints: Use Upper_bound() instead of lower_bound()] Also Print the LIS array.
- Find Maximum Sum Increasing Sub-sequence (If Maximum Sum is equal For two sub-sequence, find the longest sub-sequence) Also Print that Sub-sequence.
- Given pair of numbers, Find LIS such that $A_i < B_i$ and $A_j < B_j$ where (i < j)
- Given pair of numbers, Find LIS such that $B_i < A_j$ where (i < j)
- Find number of unique longest increasing sub-sequence in Array

Here is some UVA Online Judge Problems Based on LIS:- 111, 231, 437, 481, 497, 1196, 10131, 10534, 11368, 11456, 11790.