3. Longest increasing subsequence

Subsequence : A subsequence of a given array is sequence formed by using subset of items from the original sequence maintaining their relative ordering.

[5,2,3,6,8]

[5,3,8] is a sub sequence.

Subarray : A sub segment of a given array.

[5,2,3] ,[2,3,6], [6,8]

Increasing subsequence : A subsequence in which elements are sorted in ascending order.

[2,3,6] [3,6,8] [2,3,8]

Longest increasing subsequence [2,3,6,8]

1. State

1. State

Parameters

i - Index of the last element. We process one item at a time.

Cost function

lis(i,A) - Longest increasing subsequence in the array ending at index i.

A - Given array

2. Transitions

2. Transitions

lis(i,A)

Base case

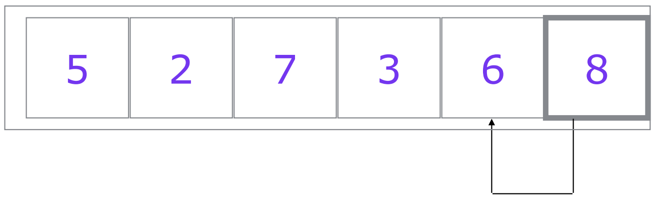
i=0 , return 1.

There is only one element and the length of this subsequence is 1.

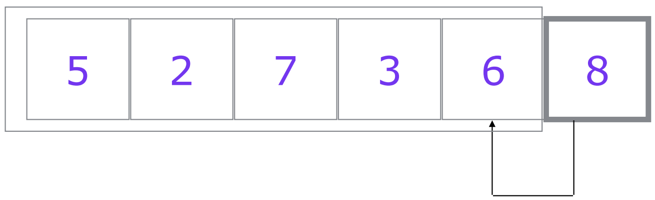
Longest increasing sequence

5	2	7	3	6	8

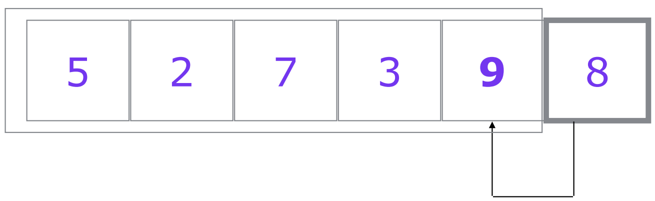
Longest increasing sequence



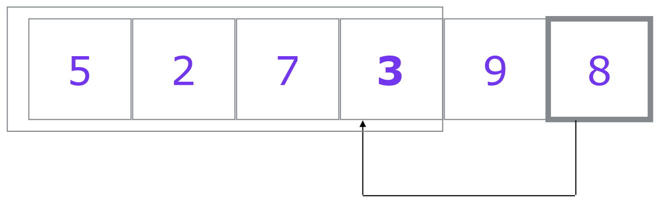
Longest increasing sequence



Longest increasing sequence



Longest increasing sequence

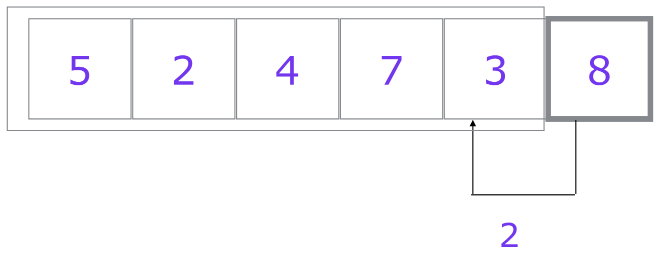


Longest increasing sequence

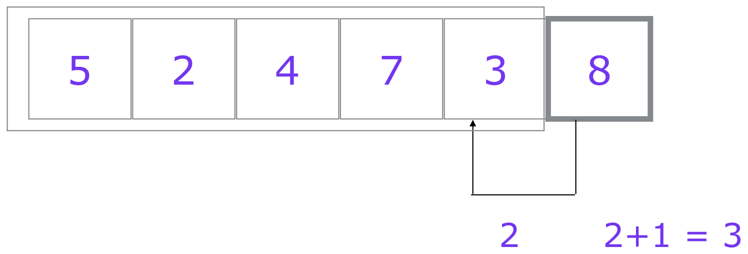
5	2	4	7	3	8

2

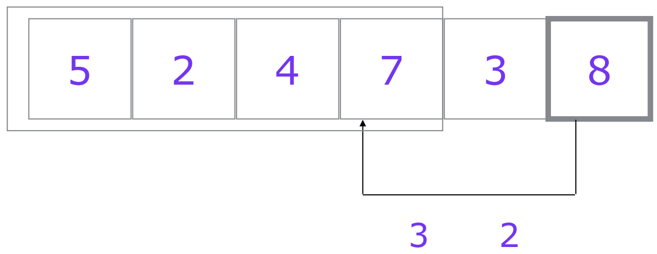
Longest increasing sequence



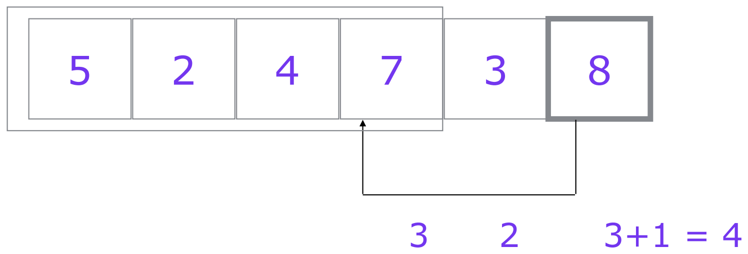
Longest increasing sequence



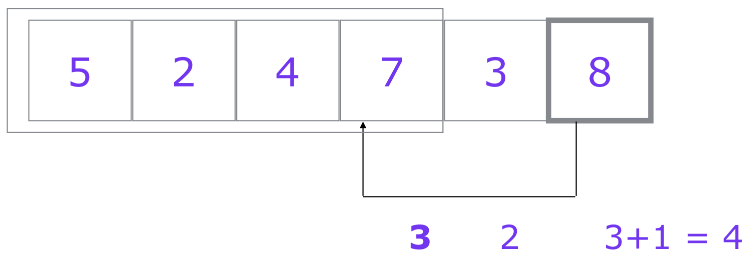
Longest increasing sequence



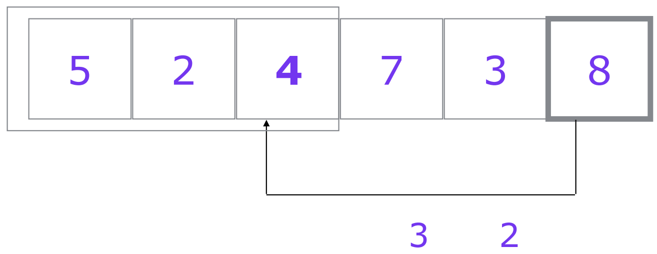
Longest increasing sequence



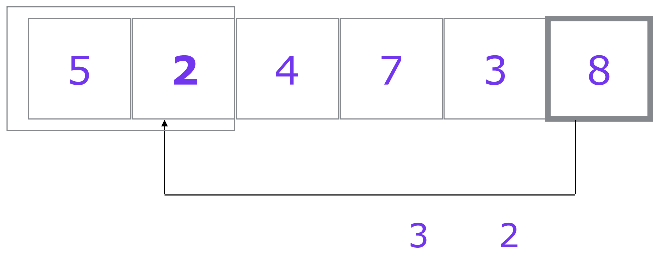
Longest increasing sequence



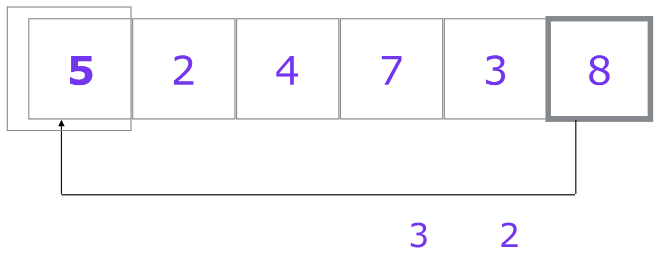
Longest increasing sequence



Longest increasing sequence



Longest increasing sequence



Longest increasing sequence

5	2	4	7	3	8	2

Longest increasing sequence

Longest increasing sequence

2. Transitions

lis(i,A)

Base case

i=0 , return 1.

There is only one element and the length of this subsequence is 1.

Recurrence relation

lis(i,A) = MAX [lis(j,A)+1 if A[i] > A[j]

[lis(j,A) if A[i] < A[j] for all j=0,1,2,3,4,5,6…i-1

3. Recursive solution

We use the recurrence relation to implement a recursive solution

Psuedo code

```
lis(i,A){
    if(i == 0){
        return 1;
    }
   max = 1
   for(j=0;j<i;j++){
        lis = lis(j,A)
        if(A[i] > A[j]){
            lis = lis+1
        }
        max = MAX(max,lis)
    }
    return max
}
```

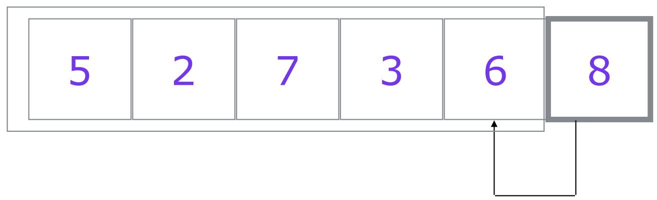
Java

```java
public static int lis(int i, int[] A) {
    if (i == 0) {
        return 1;
    }
    int max = 0;
    for (int j = 0; j < i; j++) {
        int lis = lis(j, A);
        if (A[i] > A[j]) {
            lis += 1;
        }
        max = Math.max(max, lis);
    }
    return max;
}
```

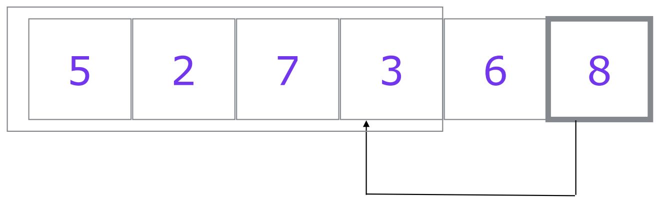
Python

```python
def lis(i, A):
    if i == 0:
        return 1
    max_l = 1
    for j in range(0, i):
        l = lis(j, A)
        if A[j] < A[i]:
            l += 1
        max_l = max(max_l, l)
    return max_l
```

Longest increasing subsequence



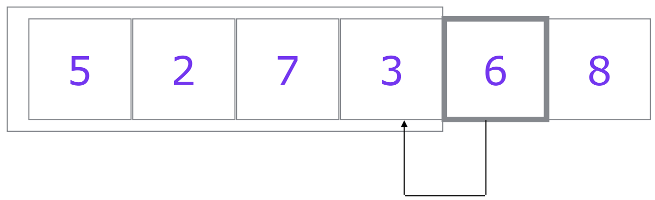
Longest increasing subsequence



Longest increasing subsequence

5	2	7	3	6	8

Longest increasing subsequence



Longest increasing subsequence

5	2	7	3	6	8

4. Memoize

We can use an array of size N as cache,

key - key is the index i

Value - longest increasing subsequence between 0 and i.

Default value - 0

Java

```java
public static int lisMemo(int[] A){
    int[] cache = new int[A.length];
    return lisMemo(A.length-1,A,cache);
}
public static int lisMemo(int i, int[] A,int[] cache){
    if(i == 0){
        return 1;
    }
    if(cache[i] != 0){
        return cache[i];
    }
```

Java

```java
    int max = 1;
    for (int j = 0; j < i; j++) {
        int lis = lis(j, A);
        if (A[i] > A[j]) {
            lis += 1;
        }
        max = Math.max(max, lis);
    }
    cache[i] = max;
    return max;
}
```

Python

```python
def lis_memo(i, A, cache):
    if i == 0:
        return 1
    if cache[i] != 0:
        return cache[i]
    max_l = 1
    for j in range(0, i):
        l = lis(j, A)
        if A[j] < A[i]:
            l += 1
        max_l = max(max_l, l)
    cache[i] = max_l
    return max_l
```

5. Bottom up approach

5. Bottom up approach

Recurrence relation

lis(i,A) = MAX [lis(j,A)+1 if A[i] > A[j]

[lis(j,A)   if A[i] < A[j]   for all j=0,1,2,3,4,5,6...i-1

Bottom up equation

dp[i] = MAX [dp[j]+1 if A[i] > A[j]

[dp[j]   if A[i] < A[j]   for all j=0,1,2,3,4,5,6...i-1

We solve all the problems dp[i] starting from i=0 , all the way up to i=N-1

Java

```java
public static int lisDP(int[] A) {
    int N = A.length;
    int[] dp = new int[N];
    dp[0] = 1;
    for (int i = 1; i < N; i++) {
        dp[i] = 1;
        for (int j = 0; j < i; j++) {
            int lis = dp[j];
            if (A[i] > A[j]) {
                lis += 1;
            }
            dp[i] = Math.max(dp[i], lis);
        }
    }
    return dp[N - 1];
}
```

Python

```python
def lis_dp(A):
    N = len(A)
    dp = [1 for _ in range(0, N)]
    for i in range(0, N):
        for j in range(0, i):
            l = dp[j]
            if A[j] < A[i]:
                l += 1
            dp[i] = max(dp[i], l)
    return dp[N - 1]
```

Time and space complexity analysis

1. Recursive solution

Recurrence equation

$$T(n) = 1 + \sum_{i=0}^{n-1} T(i) = 2^N$$

$$T(0) = 1$$

Time complexity $O(2^N)$, Exponential

Space complexity O(1)

Dynamic programming

Two for loops

Inner for loop runs from j=0,1,2,3…i-1

Outer

$$\sum_{i=0}^{N}\sum_{j=0}^{i} 1 = \sum_{i=0}^{N} i = 1+2+3+\ldots+N = N(N+1)/2 = (N^2+N)/2 = O(N^2)$$

Time complexity $O(N^2)$

Space complexity $O(N)$

For Longest increasing subsequence, there is O(NlgN) solution which is based on binary search.

I suggest reading about it more, link given below