

## Longest Increasing Subsequence

### Problem Introduction:

The Longest Increasing Subsequence (LIS) problem is to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order.

### Method used: (Dynamic Programming)

To accomplish this task, we define an array  $d[0..n-1]$ , where  $d[i]$  is the length of the longest increasing subsequence that ends in the element at index  $i$ . We will compute this array gradually. So let the current index be  $i$ . i.e. we want to compute the value  $d[i]$  and all previous values  $d[0]$  ...,  $d[i-1]$  are already known. Then there are two options:

- $d[i]=1$ : the required subsequence consists of only the element  $a[i]$ .
- $d[i]>1$ : then in the required subsequence is another number before the number  $a[i]$ .

If the second option is true, then we append the element  $a[j]$  to a third array which will hold our wanted LIS.

This method is a little complex, although it takes time equal to  $n^2$  but its main problem is in consuming much memory.

```
vector<int> longestSubSeq(vector<int> const& a) {  
    int n = a.size();  
    vector<vector<int>> lis;  
    for(size_t i=0; i<n; i++) {  
        lis.push_back({});  
    }  
    vector<int> d(n, 1);  
    for (int i = 0; i < n; i++) {  
        for (int j = 0; j < i; j++) {  
            if (a[i] > a[j] && d[i] < d[j]+1) {  
                d[i] = d[j] + 1;  
                lis[i].push_back(a[j]);  
            }  
        }  
    }  
    auto pos = distance(d.begin(), std::max_element(d.begin(), d.end()));  
    lis[pos].push_back(a[pos]);  
    return lis[pos];  
}
```

The Original Subsequence is: 7 3 8 4 2 6  
Length of Longest Increasing Subsequence is: 3  
The Longest Increasing Subsequence is: 3 4 6