Problem 1: Longest Increasing Subsequence (LIS)

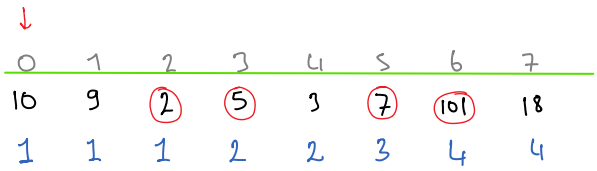
Given an unsorted array a[] of integers, find the length of the longest increasing subsequence.

A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous. So we

can skip some elements in the array in order to make the longest subsequence.

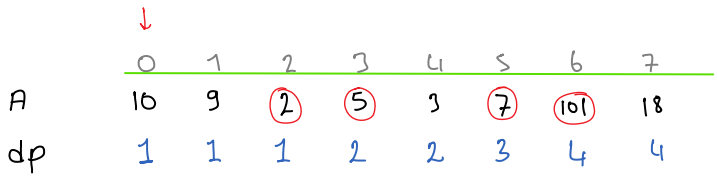
Sample Input
8
10 9 2 5 3 7 101 18

Sample Output
4
Output Details
10 9 2 5 3 7 101 18



- Sub-problems: prefixes of the input array.
- Partial solution of i: Len of the LIS ending in position i

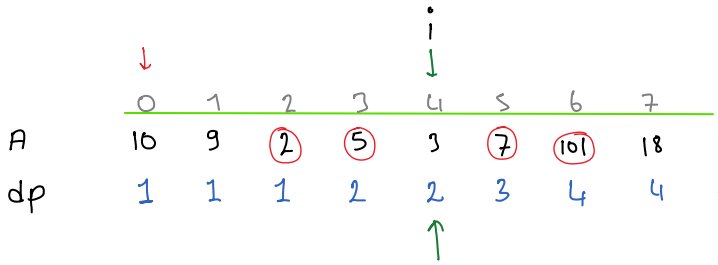
Sol 1: O(N²)



← res is the max element

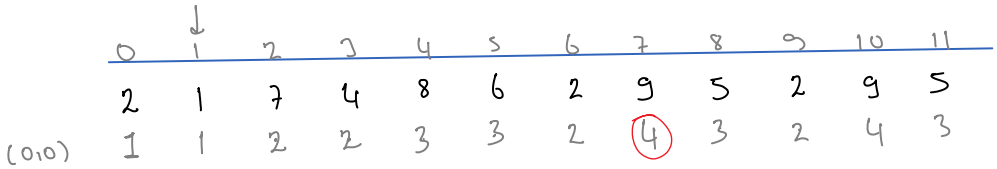
```
int main()
{
    vector<int> dp(N,1);

    for(int i=1; i<N; i++)
        for(int j=0; j<i; j++)
            if(A[j] < A[i])
                dp[i] = max(dp[i], dp[j]+1);

    cout << *max_element(dp.begin(), dp.end()) << endl;
```

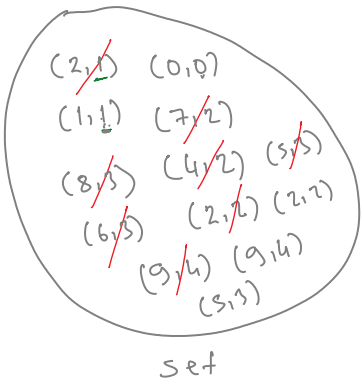
Sol 2: O(NlgN)



← res is the max element

- For the position i we made i-1 comparisons. Do we need to make all those comparisons?
  if (A[j] ≥ A[i]) then A[j] is redundant.
- How can we get only the useful numbers for the position i?
  Set ?



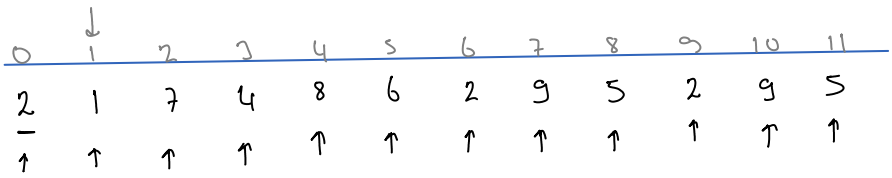


solution bag →

(a, len) ← increasing subseq. with len numbers and ending with a

Eliminated the redundant partial solutions

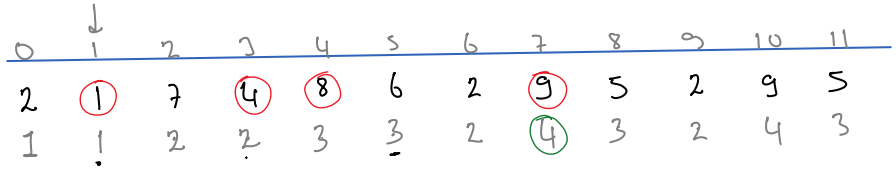
Implementing with an array.



sol array: 2 1 7 4 8 6 9
2 5

```
vector<int> solArray;  // empty array
for (int i=0; i<N; i++)
{
    auto it = lower_bound(solArray.begin(), solArray.end(), A[i]);
    if (it == solArray.end()) // A[i] is the largest
        solArray.push_back(A[i]);
    else
        *it = A[i];
}
cout << solArray.size() << endl;
```

Printing One of the LISs : Backward searching



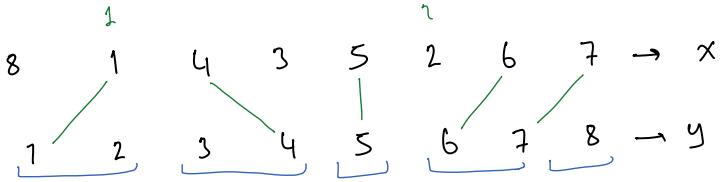
dp

1 4 6 9

Building Bridges

Sample Input

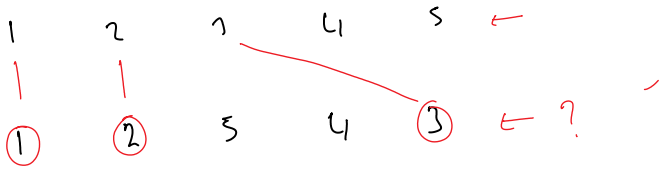
8
8 1 4 3 5 2 6 7
1 2 3 4 5 6 7 8



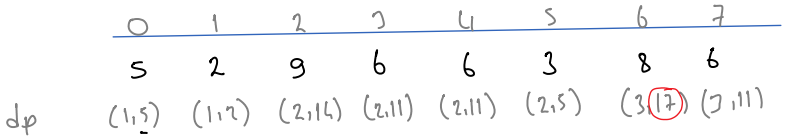
- sort (xi, yi) pairs in increasing order by xi

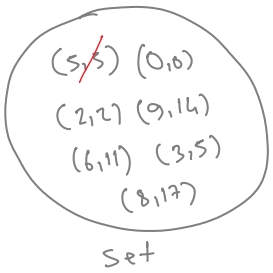
(1,7) (2,6) (3,4) (4,3) (5,5) (6,7) (7,8) (8,1)

- Answer is the LIS of y values.



Maximum Sum Increasing Subsequence



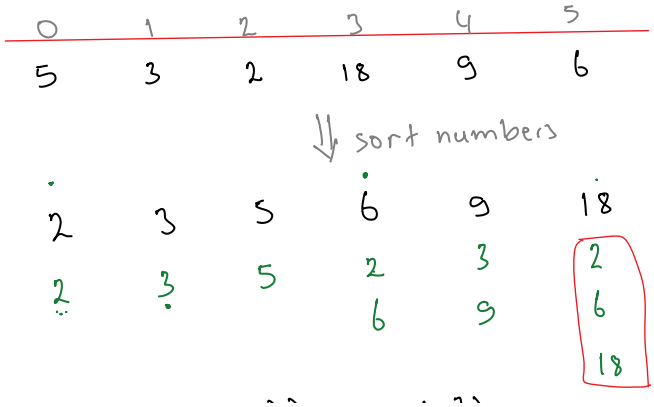


Largest Divisible Subset

Given a set of N (1 <= N <= 1000) distinct positive integers, find the largest subset such that every pair (Si, Sj) of elements in this subset satisfies:
Si % Sj = 0 or Sj % Si = 0.
If there are multiple solutions, return any subset is fine.

Sample Input
6
5 3 2 18 9 6
Sample Output
3 6 18

dp[N][N]



$O(N(N+N)) \Rightarrow O(N^2)$

Problem 2: Longest Common Subsequence (LCS)

Given two sequences, find the length of longest subsequence present in both of them.

$O(NM)$

String1: a b c d e f

String2: b d e f

S1 = x x x x x x

S2 = x x x x x

$\Rightarrow$ S1[n] == S2[m]

$1 + f(n-1, m-1)$

else max { $f(n-1, m)$, $f(n, m-1)$ }

		A	C	B	A	D	C
	0	0	0	0	0	0	0
C	0	0	1	1	1	1	1
A	0	1	1	1	2	2	2
C	0	1	2	2	2	2	3
D	0	1	2	2	2	3	3
B	0	1	2	3	3	3	3

(S1)

(S2)

$O(NM)$ time

$O(N)$ space

Problem 3: Longest Common Substring (LCS)

		C	B	A	D	C	B
	0	0	0	0	0	0	0
C	0	1	0	0	0	1	0
B	0	0	2	0	0	0	2
A	0	0	0	3	0	0	0
D	0	0	0	0	4	0	0
B	0	1	1	0	0	0	1
C	0	1	0	0	0	1	0

$O(NM)$ time

$O(N)$ space

Problem 4: Edit (Levenshtein) Distance

Given two words word1 and word2, find the minimum number of operations required to convert word1 to word2.

You have the following three operations permitted on a word:

- Insert a character
- Delete a character
- Replace a character

Sample Input

Horse

ros

Sample Output

3

H O ~~R~~ S ~~E~~ $\Rightarrow$ R O S

R

		H	O	R	S	E
	0	1	2	3	4	5
R	1	1	2	2	3	4
O	2	2	1	2	3	4
S	3	3	2	2	2	3

$O(NM)$ time

$O(N)$ space