**Maximum Sum Increasing Subsequence**

**WE'LL COVER THE FOLLOWING**

* + - [Problem Statement](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N#problem-statement)
    - [Basic Solution](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N#basic-solution)
    - [Top-down Dynamic Programming with Memoization](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N#top-down-dynamic-programming-with-memoization)
    - [Bottom-up Dynamic Programming](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N#bottom-up-dynamic-programming)

**Problem Statement**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N#problem-statement)

Given a number sequence, find the increasing subsequence with the highest sum. Write a method that returns the highest sum.

**Example 1:**

Input: {4,1,2,6,10,1,12}  
Output: 32  
Explanation: The increaseing sequence is {4,6,10,12}.   
Please note the difference, as the LIS is {1,2,6,10,12} which has a sum of '31'.

**Example 2:**

Input: {-4,10,3,7,15}  
Output: 25  
Explanation: The increaseing sequences are {10, 15} and {3,7,15}.

**Basic Solution**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N#basic-solution)

The problem is quite similar to the [Longest Increasing Subsequence](https://www.educative.io/collection/page/5668639101419520/5633779737559040/5733679603122176/). The only difference is that, instead of finding the increasing subsequence with the maximum length, we need to find an increasing sequence with the maximum sum.

A basic brute-force solution could be to try all the subsequences of the given array. We can process one number at a time, so we have two options at any step:

1. If the current number is greater than the previous number that we included, we include that number in a running sum and make a recursive call for the remaining array.
2. We can skip the current number to make a recursive call for the remaining array.

The highest sum of any increasing subsequence would be the max value returned by the two recurse calls from the above two options.

Here is the code:

* [Java](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N)
* [JS](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N)
* [Python3](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N)
* [C++](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

class MSIS {

  public int findMSIS(int[] nums) {

      return findMSISRecursive(nums, 0, -1, 0);

  }

  private int findMSISRecursive(int[] nums, int currentIndex, int previousIndex, int sum) {

    if(currentIndex == nums.length)

      return sum;

    // include nums[currentIndex] if it is larger than the last included number

    int s1 = sum;

    if(previousIndex == -1 || nums[currentIndex] > nums[previousIndex])

      s1 = findMSISRecursive(nums, currentIndex+1, currentIndex, sum + nums[currentIndex]);

    // excluding the number at currentIndex

    int s2 = findMSISRecursive(nums, currentIndex+1, previousIndex, sum);

    return Math.max(s1, s2);

  }

  public static void main(String[] args) {

    MSIS msis = new MSIS();

    int[] nums = {4,1,2,6,10,1,12};

    System.out.println(msis.findMSIS(nums));

    nums = new int[]{-4,10,3,7,15};

    System.out.println(msis.findMSIS(nums));

  }

}





RUN

SAVERESET

The time complexity of the above algorithm is exponential O(2^n)*O*(2​*n*​​), where ‘n’ is the lengths of the input array. The space complexity is O(n)*O*(*n*) which is used to store the recursion stack.

**Top-down Dynamic Programming with Memoization**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N#top-down-dynamic-programming-with-memoization)

We can use memoization to overcome the overlapping subproblems.

The three changing values for our recursive function are the current index, the previous index, and the sum. An efficient way of storing the results of the subproblems could be a hash-table whose key would be a string (currentIndex + “|” + previousIndex + “|” + sum).

Here is the code:

* [Java](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N)
* [JS](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N)
* [Python3](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N)
* [C++](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N)

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

  private int findMSISRecursive(Map<String, Integer> dp, int[] nums, int currentIndex, int previousIndex, int sum) {

    if(currentIndex == nums.length)

      return sum;

    String subProblemKey = currentIndex + "-" + previousIndex + "-" + sum;

    if(!dp.containsKey(subProblemKey)) {

      // include nums[currentIndex] if it is larger than the last included number

      int s1 = sum;

      if(previousIndex == -1 || nums[currentIndex] > nums[previousIndex])

        s1 = findMSISRecursive(dp, nums, currentIndex+1, currentIndex, sum + nums[currentIndex]);

      // excluding the number at currentIndex

      int s2 = findMSISRecursive(dp, nums, currentIndex+1, previousIndex, sum);

      dp.put(subProblemKey, Math.max(s1, s2));

    }

    return dp.get(subProblemKey);

  }

  public static void main(String[] args) {

    MSIS msis = new MSIS();

    int[] nums = {4,1,2,6,10,1,12};

    System.out.println(msis.findMSIS(nums));

    nums = new int[]{-4,10,3,7,15};

    System.out.println(msis.findMSIS(nums));

    nums = new int[]{1,3,8,4,14,6,14,1,9,4,13,3,11,17,29};

    System.out.println(msis.findMSIS(nums));

  }

}





RUN

SAVERESET

**Bottom-up Dynamic Programming**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N#bottom-up-dynamic-programming)

The above algorithm tells us two things:

1. If the number at the current index is bigger than the number at the previous index, we include that number in the sum for an increasing sequence up to the current index.
2. But if there is a maximum sum increasing subsequence (MSIS), without including the number at the current index, we take that.

So we need to find all the increasing subsequences for a number at index i, from all the previous numbers (i.e. numbers till index i-1), to find MSIS.

If i represents the currentIndex and ‘j’ represents the previousIndex, our recursive formula would look like:

    if num[i] > num[j] => dp[i] = dp[j] + num[i] if there is no bigger MSIS for 'i'

Let’s draw this visually for {-4,10,3,7,15}. Start with a subsequence of length ‘1’, as every number can represent an MSIS:

num[] dp[] -4 -4 10 10 3 3 7 7 15 15

Every number is a LIS of length '1'

**1** of 11

From the above visualization, we can clearly see that the maximum sum of any increasing subsequence is ‘25’ – as shown by dp[4].

Here is the code for our bottom-up dynamic programming approach:

* [Java](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N)
* [JS](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N)
* [Python3](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N)
* [C++](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/B8rgqKEW05N)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

class MSIS {

  public int findMSIS(int[] nums) {

    int[] dp = new int[nums.length];

    dp[0] = nums[0];

    int maxSum = nums[0];

    for (int i=1; i<nums.length; i++) {

      dp[i] = nums[i];

      for (int j=0; j<i; j++) {

        if (nums[i] > nums[j] && dp[i] < dp[j] + nums[i])

          dp[i] = dp[j] + nums[i];

      }

      maxSum = Math.max(maxSum, dp[i]);

    }

    return maxSum;

  }

  public static void main(String[] args) {

    MSIS msis = new MSIS();

    int[] nums = {4,1,2,6,10,1,12};

    System.out.println(msis.findMSIS(nums));

    nums = new int[]{-4,10,3,7,15};

    System.out.println(msis.findMSIS(nums));

    nums = new int[]{1,3,8,4,14,6,14,1,9,4,13,3,11,17,29};

    System.out.println(msis.findMSIS(nums));

  }

}





RUN

SAVERESET

The time complexity of the above algorithm is O(n^2)*O*(*n*​2​​) and the space complexity is O(n)*O*(*n*).

**MARK AS COMPLETED**

[**←    Back**](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/YQ0y0QOJQ69)

Longest Increasing Subsequence

[**Next    →**](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/3jjLPyLGnAR)

Shortest Common Super-sequence

Stuck? Get help on

[**DISCUSS**](https://discuss.educative.io/c/grokking-dynamic-programming-patterns-for-coding-interviews-design-gurus/pattern-5-longest-common-substring-maximum-sum-increasing-subsequence)

Send feedback

9 Recommendations