**House thief**

**WE'LL COVER THE FOLLOWING**

* + - [Problem Statement](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3#problem-statement)
    - [Basic Solution](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3#basic-solution)
    - [Top-down Dynamic Programming with Memoization](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3#top-down-dynamic-programming-with-memoization)
    - [Bottom-up Dynamic Programming](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3#bottom-up-dynamic-programming)
      * [Code](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3#code)
    - [Memory optimization](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3#memory-optimization)
    - [Fibonacci number pattern](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3#fibonacci-number-pattern)

There are ‘n’ houses built in a line. A thief wants to steal maximum possible money from these houses. The only restriction the thief has is that he can’t steal from two consecutive houses, as that would alert the security system. How should the thief maximize his stealing?

**Problem Statement**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3#problem-statement)

Given a number array representing the wealth of ‘n’ houses, determine the maximum amount of money the thief can steal without alerting the security system.

**Example 1:**

Input: {2, 5, 1, 3, 6, 2, 4}  
Output: 15  
Explanation: The thief should steal from houses 5 + 6 + 4

**Example 2:**

Input: {2, 10, 14, 8, 1}  
Output: 18  
Explanation: The thief should steal from houses 10 + 8

Let’s first start with a recursive brute-force solution.

**Basic Solution**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3#basic-solution)

For every house ‘i’, we have two options:

1. Steal from the current house (‘i’), skip one and steal from (‘i+2’).
2. Skip the current house (‘i’), and steal from the adjacent house (‘i+1’).

The thief should choose the one with the maximum amount from the above two options. So our algorithm will look like:

* [Java](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3)
* [JS](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3)
* [Python3](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3)
* [C++](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3)

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

class HouseThief {

  public int findMaxSteal(int[] wealth) {

    return findMaxStealRecursive(wealth, 0);

  }

  private int findMaxStealRecursive(int[] wealth, int currentIndex) {

    if( currentIndex >= wealth.length)

      return 0;

    // steal from current house and skip one to steal from the next house

    int stealCurrent = wealth[currentIndex] + findMaxStealRecursive(wealth, currentIndex + 2);

    // skip current house to steel from the adjacent house

    int skipCurrent = findMaxStealRecursive(wealth, currentIndex + 1);

    return Math.max(stealCurrent, skipCurrent);

  }

  public static void main(String[] args) {

    HouseThief ht = new HouseThief();

    int[] wealth = {2, 5, 1, 3, 6, 2, 4};

    System.out.println(ht.findMaxSteal(wealth));

    wealth = new int[]{2, 10, 14, 8, 1};

    System.out.println(ht.findMaxSteal(wealth));

  }

}





RUN

SAVERESET

The time complexity of the above algorithm is exponential O(2^n)*O*(2​*n*​​). 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/m2EOxJ0Nkp3#top-down-dynamic-programming-with-memoization)

To resolve overlapping subproblems, we can use an array to store the already solved subproblems.

Here is the code:

* [Java](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3)
* [JS](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3)
* [Python3](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3)
* [C++](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3)

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

31

1

  public int findMaxSteal(int[] wealth) {

    int dp[] = new int[wealth.length];

    return findMaxStealRecursive(dp, wealth, 0);

  }

  private int findMaxStealRecursive(int[] dp, int[] wealth, int currentIndex) {

    if( currentIndex >= wealth.length)

      return 0;

    if(dp[currentIndex] == 0) {

      // steal from current house and skip one to steal next

      int stealCurrent = wealth[currentIndex] + findMaxStealRecursive(dp, wealth, currentIndex + 2);

      // skip current house to steel from the adjacent house

      int skipCurrent = findMaxStealRecursive(dp, wealth, currentIndex + 1);

      dp[currentIndex] = Math.max(stealCurrent, skipCurrent);

    }

    return dp[currentIndex];

  }

  public static void main(String[] args) {

    HouseThief ht = new HouseThief();

    int[] wealth = {2, 5, 1, 3, 6, 2, 4};

    System.out.println(ht.findMaxSteal(wealth));

    wealth = new int[]{2, 10, 14, 8, 1};

    System.out.println(ht.findMaxSteal(wealth));

  }

}

class HouseThief {





RUN

SAVERESET

**Bottom-up Dynamic Programming**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3#bottom-up-dynamic-programming)

Let’s try to populate our dp[] array from the above solution, working in a bottom-up fashion. As we saw in the above code, every findMaxStealRecursive() is the maximum of the two recursive calls; we can use this fact to populate our array.

**Code**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3#code)

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

* [Java](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3)
* [JS](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3)
* [Python3](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3)
* [C++](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

class HouseThief {

  public int findMaxSteal(int[] wealth) {

    if(wealth.length == 0) return 0;

    int dp[] = new int[wealth.length+1]; // '+1' to handle the zero house

    dp[0] = 0; // if there are no houses, the thief can't steal anything

    dp[1] = wealth[0]; // only one house, so the thief have to steal from it

    // please note that dp[] has one extra element to handle zero house

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

      dp[i+1] = Math.max(wealth[i] + dp[i-1], dp[i]);

    return dp[wealth.length];

  }

  public static void main(String[] args) {

    HouseThief ht = new HouseThief();

    int[] wealth = {2, 5, 1, 3, 6, 2, 4};

    System.out.println(ht.findMaxSteal(wealth));

    wealth = new int[]{2, 10, 14, 8, 1};

    System.out.println(ht.findMaxSteal(wealth));

  }

}





RUN

SAVERESET

The above solution has time and space complexity of O(n)*O*(*n*).

**Memory optimization**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3#memory-optimization)

We can optimize the space used in our previous solution. We don’t need to store all the previous numbers up to ‘n’, as we only need two previous numbers to calculate the next number in the sequence. Let’s use this fact to further improve our solution:

* [Java](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3)
* [JS](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3)
* [Python3](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3)
* [C++](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

class HouseThief {

  public int findMaxSteal(int[] wealth) {

    if(wealth.length == 0) return 0;

    int n1=0, n2=wealth[0], temp;

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

      temp = Math.max(n1 + wealth[i], n2);

      n1 = n2;

      n2 = temp;

    }

    return n2;

  }

  public static void main(String[] args) {

    HouseThief ht = new HouseThief();

    int[] wealth = {2, 5, 1, 3, 6, 2, 4};

    System.out.println(ht.findMaxSteal(wealth));

    wealth = new int[]{2, 10, 14, 8, 1};

    System.out.println(ht.findMaxSteal(wealth));

  }

}





RUN

SAVERESET

The above solution has a time complexity of O(n)*O*(*n*) and a constant space complexity O(1)*O*(1).

**Fibonacci number pattern**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2EOxJ0Nkp3#fibonacci-number-pattern)

We can clearly see that this problem follows the Fibonacci number pattern. The only difference is that every Fibonacci number is a sum of the two preceding numbers, whereas in this problem every number (total wealth) is the maximum of previous two numbers.

**MARK AS COMPLETED**

[**←    Back**](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nDNy6JDP1G)

Minimum jumps with fee

[**Next    →**](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/RMk1D1DY1PL)

Longest Palindromic Subsequence

Stuck? Get help on

[**DISCUSS**](https://discuss.educative.io/c/grokking-dynamic-programming-patterns-for-coding-interviews-design-gurus/pattern-4-fibonacci-numbers-house-thief)

Send feedback

9 Recommendations