Ketchup

Algoritmia II

February 13, 2023

At Rosie's party, there is a table full of N prizes, all in a row with a number on it that identifies the value of each of them.

Ketchup just won a few games, so he can take as many prizes as he likes with the condition that he can not take two prizes next to each other.

Help Ketchup to know how much total value on prizes he can take home.

Input

The first line has the number N. The following line has N numbers separated by a space representing each prize's values.

Output

An integer representing the maximum total value of prizes Ketchup can take home.

Sample 1

Input	**Output**
5 10 5 20 5 3	33

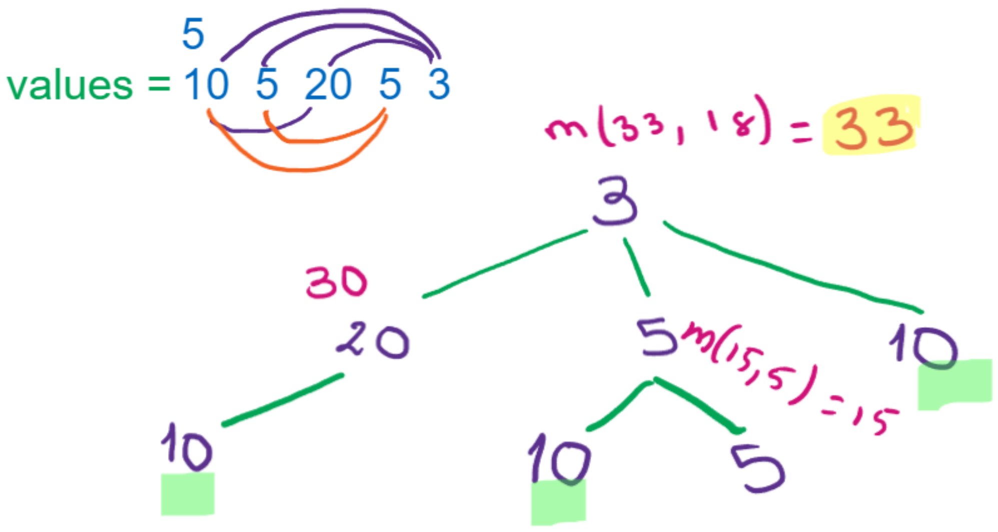
Sample 2

Input	**Output**
4 5 2 1 5	10

Top-down

(5) F(i) = The maximum total value that can be obtained for i prizes. Where 1 < i <= n

(6 - 7) Analysis of sample 1:



F(n) = max(F(n - 2) + values(n), F(n - 1)),Where: n > 2

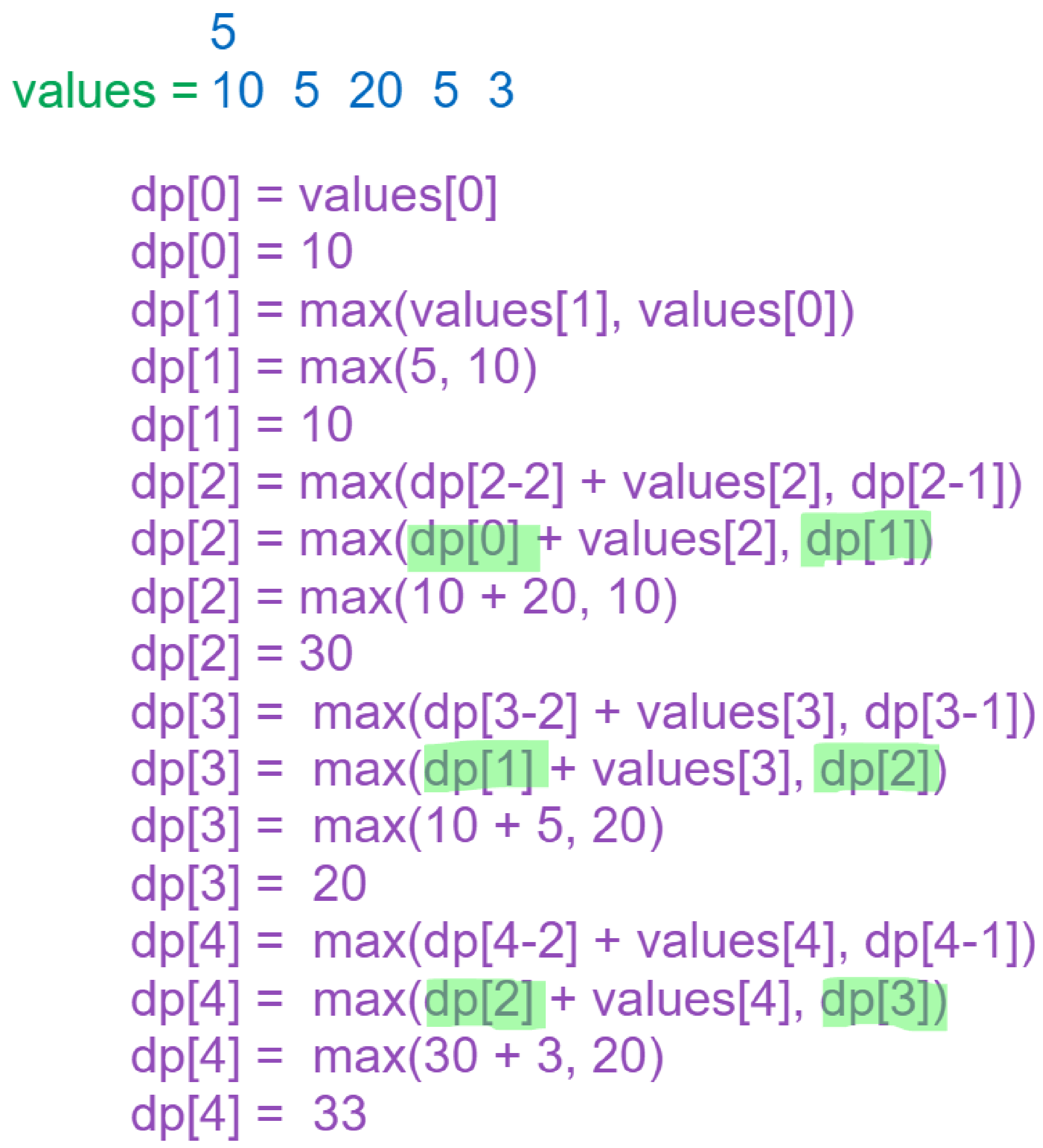
(code)

```
public static int maxPrizesValue(int n, int[] dp, int[] values) {
  if (n == 0) {
    return values[0];
  }

  if (n == 1) {
    dp[1] = Math.max(values[0], values[1]);
  }

  if (dp[n] != -1) {
    return dp[n];
  }

  dp[n] = Math.max(maxPrizesValue(n - 2, dp, values) + values[n],
      maxPrizesValue(n - 1, dp, values));
  return dp[n];

}
```

Bottom-up

(5) F(i) = The maximum total value that can be obtained for i prizes. Where 1 < i <= n

(6 - 7) Analysis of sample 1:



```
          5
values = 10  5  20  5  3

dp[0] = values[0]
dp[0] = 10
dp[1] = max(values[1], values[0])
dp[1] = max(5, 10)
dp[1] = 10
dp[2] = max(dp[2-2] + values[2], dp[2-1])
dp[2] = max(dp[0] + values[2], dp[1])
dp[2] = max(10 + 20, 10)
dp[2] = 30
dp[3] =  max(dp[3-2] + values[3], dp[3-1])
dp[3] =  max(dp[1] + values[3], dp[2])
dp[3] =  max(10 + 5, 20)
dp[3] =  20
dp[4] =  max(dp[4-2] + values[4], dp[4-1])
dp[4] =  max(dp[2] + values[4], dp[3])
dp[4] =  max(30 + 3, 20)
dp[4] =  33
```

F(n) = max(F(n - 2) + values(n), F(n - 1)),Where: n > 2

code

```
public static int maxPrizesValue(int n, int[] values) {
  int[] dp = new int[n];
  dp[0] = values[0];
  if (n >= 2)
    dp[1] = Math.max(values[0], values[1]);

  for (int i = 2; i < n; i++) {
    dp[i] = Math.max(dp[i - 2] + values[i], dp[i - 1]);
  }
  return dp[n - 1];

}
```