Problem 1388: Pizza With 3n Slices

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

There is a pizza with

3n

slices of varying size, you and your friends will take slices of pizza as follows:

You will pick

any

pizza slice.

Your friend Alice will pick the next slice in the anti-clockwise direction of your pick.

Your friend Bob will pick the next slice in the clockwise direction of your pick.

Repeat until there are no more slices of pizzas.

Given an integer array

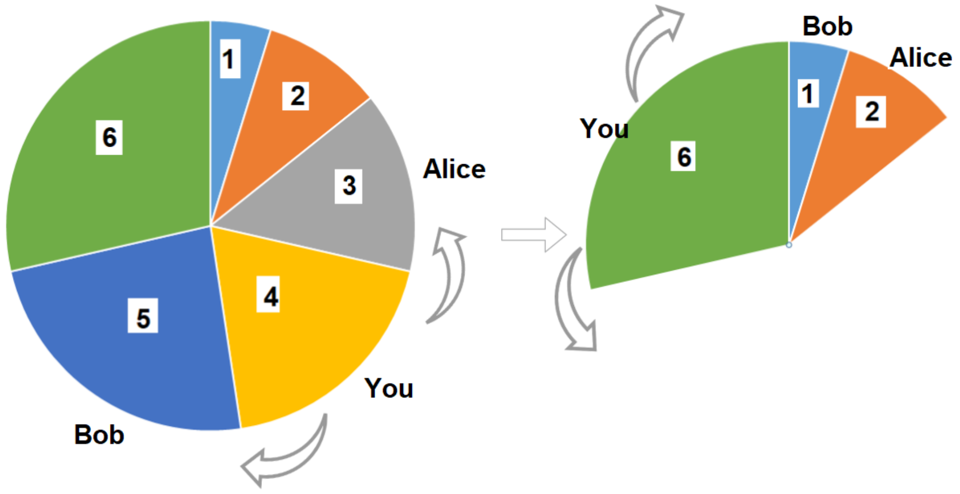
slices

that represent the sizes of the pizza slices in a clockwise direction, return

the maximum possible sum of slice sizes that you can pick

.

Example 1:



Input:

slices = [1,2,3,4,5,6]

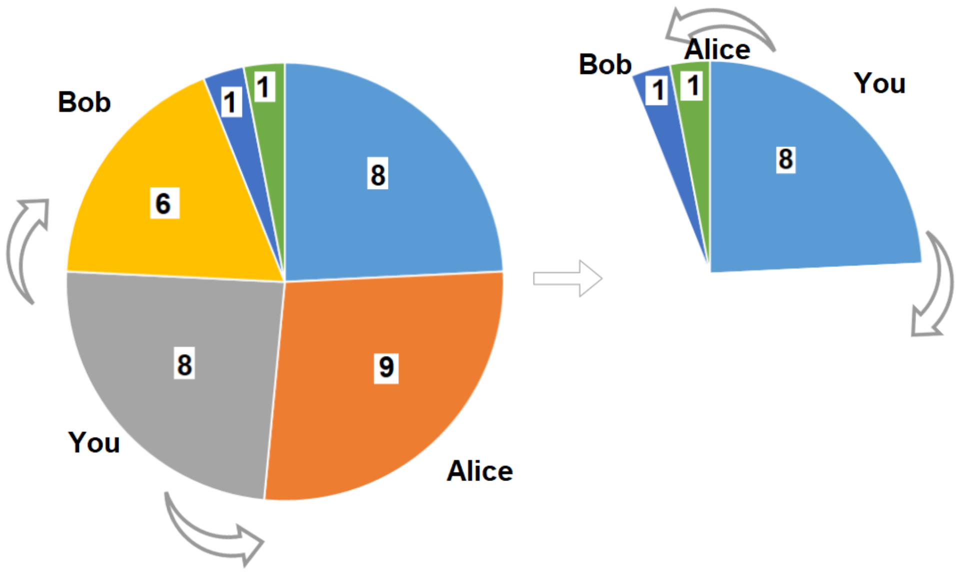
Output:

10

Explanation:

Pick pizza slice of size 4, Alice and Bob will pick slices with size 3 and 5 respectively. Then Pick slices with size 6, finally Alice and Bob will pick slice of size 2 and 1 respectively. Total = 4 + 6.

Example 2:



Input:

slices = [8,9,8,6,1,1]

Output:

16

Explanation:

Pick pizza slice of size 8 in each turn. If you pick slice with size 9 your partners will pick slices of size 8.

Constraints:

3 * n == slices.length

1 <= slices.length <= 500

1 <= slices[i] <= 1000

Code Snippets

C++:

```
class Solution {
public:
int maxSizeSlices(vector<int>& slices) {

}
};
```

Java:

```
class Solution {
public int maxSizeSlices(int[] slices) {

}
}
```

Python3:

```
class Solution:
def maxSizeSlices(self, slices: List[int]) -> int:
```

Python:

```
class Solution(object):
def maxSizeSlices(self, slices):
"""
:type slices: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} slices
* @return {number}
*/
var maxSizeSlices = function(slices) {

};
```

TypeScript:

```
function maxSizeSlices(slices: number[]): number {
};
```

C#:

```
public class Solution {
public int MaxSizeSlices(int[] slices) {

}
}
```

C:

```
int maxSizeSlices(int* slices, int slicesSize) {

}
```

Go:

```
func maxSizeSlices(slices []int) int {

}
```

Kotlin:

```
class Solution {
fun maxSizeSlices(slices: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxSizeSlices(_ slices: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_size_slices(slices: Vec<i32>) -> i32 {
}
}
```

Ruby:

```
# @param {Integer[]} slices
# @return {Integer}
def max_size_slices(slices)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $slices
* @return Integer
*/
function maxSizeSlices($slices) {

}
}
```

Dart:

```
class Solution {
int maxSizeSlices(List<int> slices) {

}
}
```

Scala:

```
object Solution {
def maxSizeSlices(slices: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_size_slices(slices :: [integer]) :: integer
def max_size_slices(slices) do

end
end
```

Erlang:

```
-spec max_size_slices(Slices :: [integer()]) -> integer().
max_size_slices(Slices) ->
.
```

Racket:

```
(define/contract (max-size-slices slices)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Pizza With 3n Slices
* Difficulty: Hard
* Tags: array, dp, greedy, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maxSizeSlices(vector<int>& slices) {

}
};
```

Java Solution:

```
/**
* Problem: Pizza With 3n Slices
* Difficulty: Hard
* Tags: array, dp, greedy, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maxSizeSlices(int[] slices) {

}
}
```

Python3 Solution:

```
"""
Problem: Pizza With 3n Slices
Difficulty: Hard
Tags: array, dp, greedy, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maxSizeSlices(self, slices: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxSizeSlices(self, slices):
"""
:type slices: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Pizza With 3n Slices
* Difficulty: Hard
* Tags: array, dp, greedy, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} slices
* @return {number}
*/
var maxSizeSlices = function(slices) {

};
```

TypeScript Solution:

```
/**
* Problem: Pizza With 3n Slices
* Difficulty: Hard
* Tags: array, dp, greedy, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxSizeSlices(slices: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Pizza With 3n Slices
* Difficulty: Hard
* Tags: array, dp, greedy, queue, heap
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaxSizeSlices(int[] slices) {

}
}
```

C Solution:

```
/*
* Problem: Pizza With 3n Slices
* Difficulty: Hard
* Tags: array, dp, greedy, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maxSizeSlices(int* slices, int slicesSize) {

}
```

Go Solution:

```
// Problem: Pizza With 3n Slices
// Difficulty: Hard
// Tags: array, dp, greedy, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxSizeSlices(slices []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxSizeSlices(slices: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxSizeSlices(_ slices: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Pizza With 3n Slices
// Difficulty: Hard
// Tags: array, dp, greedy, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_size_slices(slices: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} slices
# @return {Integer}
def max_size_slices(slices)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $slices
* @return Integer
*/
function maxSizeSlices($slices) {

}
}
```

Dart Solution:

```
class Solution {
int maxSizeSlices(List<int> slices) {

}
}
```

Scala Solution:

```
object Solution {
def maxSizeSlices(slices: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_size_slices(slices :: [integer]) :: integer
def max_size_slices(slices) do

end
end
```

Erlang Solution:

```
-spec max_size_slices(Slices :: [integer()]) -> integer().
max_size_slices(Slices) ->
.
```

Racket Solution:

```
(define/contract (max-size-slices slices)
(-> (listof exact-integer?) exact-integer?)
)
```