Problem 1099: Two Sum Less Than K

Problem Information

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

Problem Description

Given an array

nums

of integers and integer

k

, return the maximum

sum

such that there exists

i < j

with

nums[i] + nums[j] = sum

and

sum < k

. If no

i

,

j

exist satisfying this equation, return

-1

.

Example 1:

Input:

nums = [34,23,1,24,75,33,54,8], k = 60

Output:

58

Explanation:

We can use 34 and 24 to sum 58 which is less than 60.

Example 2:

Input:

nums = [10,20,30], k = 15

Output:

-1

Explanation:

In this case it is not possible to get a pair sum less that 15.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 1000

1 <= k <= 2000

Code Snippets

C++:

```
class Solution {
public:
int twoSumLessThanK(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public int twoSumLessThanK(int[] nums, int k) {

}
}
```

Python3:

```
class Solution:
def twoSumLessThanK(self, nums: List[int], k: int) -> int:
```

Python:

```
class Solution(object):
def twoSumLessThanK(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: int
```

```
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
var twoSumLessThanK = function(nums, k) {

};
```

TypeScript:

```
function twoSumLessThanK(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int TwoSumLessThanK(int[] nums, int k) {

}
}
```

C:

```
int twoSumLessThanK(int* nums, int numsSize, int k) {

}
```

Go:

```
func twoSumLessThanK(nums []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun twoSumLessThanK(nums: IntArray, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func twoSumLessThanK(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn two_sum_less_than_k(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def two_sum_less_than_k(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function twoSumLessThanK($nums, $k) {

}

}
```

Dart:

```
class Solution {
int twoSumLessThanK(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def twoSumLessThanK(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec two_sum_less_than_k(nums :: [integer], k :: integer) :: integer
def two_sum_less_than_k(nums, k) do

end
end
```

Erlang:

```
-spec two_sum_less_than_k(Nums :: [integer()], K :: integer()) -> integer().
two_sum_less_than_k(Nums, K) ->
.
```

Racket:

```
(define/contract (two-sum-less-than-k nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Two Sum Less Than K
* Difficulty: Easy
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int twoSumLessThanK(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Two Sum Less Than K
* Difficulty: Easy
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int twoSumLessThanK(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Two Sum Less Than K
Difficulty: Easy
Tags: array, sort, search
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

```
class Solution(object):
def twoSumLessThanK(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Two Sum Less Than K
* Difficulty: Easy
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
var twoSumLessThanK = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Two Sum Less Than K
* Difficulty: Easy
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function twoSumLessThanK(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Two Sum Less Than K
* Difficulty: Easy
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int TwoSumLessThanK(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Two Sum Less Than K
* Difficulty: Easy
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int twoSumLessThanK(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Two Sum Less Than K
// Difficulty: Easy
// Tags: array, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func twoSumLessThanK(nums []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun twoSumLessThanK(nums: IntArray, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func twoSumLessThanK(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Two Sum Less Than K
// Difficulty: Easy
// Tags: array, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn two_sum_less_than_k(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def two_sum_less_than_k(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function twoSumLessThanK($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int twoSumLessThanK(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def twoSumLessThanK(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec two_sum_less_than_k(nums :: [integer], k :: integer) :: integer
def two_sum_less_than_k(nums, k) do

end
end
```

Erlang Solution:

```
-spec two_sum_less_than_k(Nums :: [integer()], K :: integer()) -> integer().
two_sum_less_than_k(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (two-sum-less-than-k nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```