First, we'll tackle another problem that's related to the given problem.

Assuming you're given an **increasing** sequence of positive integers $(a_n)_{n\in\mathbb{N}}$
$(\forall i \forall j(1 \le i < j \le n \to a_i < a_j))$

and some integer s.

$(*)$ You need to find a pair a_i, a_j s.t $a_i + a_j = s$.

$$(a_n) = (a_1, a_2, \dots, a_n)$$

start from a_1 and a_n.

if $a_1 + a_n < s$ then we must go right from a_1 and check $a_2 + a_n$

else if $s < a_1 + a_n$ then we must go left from a_n and check $a_1 + a_{n-1}$

we do this until we get $(*)$

Now we'll solve the given problem.

The naive solution will be to go through all subsequences of size 3 of a_n - there're

$$\lambda_1 = \binom{n}{3} = \frac{n!}{3!*(n-3)!} = \frac{n(n-1)(n-2)}{6}$$ possibilities for that.

For each triplet we can choose two elements out of three and check if their sum is equal to the third element – there're

$$\lambda_2 = \binom{3}{2} = 3$$ possibilities for that.

So, we can solve this by using no less than $\lambda_1 * \lambda_2$ steps.

If we consider the additional basic operations that takes $O(1)$ steps (e.g. addition, assignment, equality check…) than the algorithm will run at $O(n^3)$ steps.

We can do better using the above algorithm for finding s.

In this case, s will be an element in (a_n).

We assume that (a_n) is an increasing subsequence.

We start from the last element a_n and check if there's a pair of elements in the sequence that sums to a_n

Time complexity:

there're n elements.

For each element we use the first algorithm which takes $O(n)$ steps, therefore, we can solve this problem with an algorithm that takes $O(n^2)$ steps (assuming that the sorting algorithm you choose has time complexity $O(n * \log(n))$ and space complexity $O(1)$).

Author: Idan Damri