In this example we create an algorithm in Python which is simple but ineficcient. Then we try to figure out how we can make it faster.

Problem: *From how many spots a given list of integers can be split in half so that the sums of left and right subsets are equal?*

For example a list $T = [1, -1, 2, -2]$ has only one spot: right in the middle. The solution can be computed by checking the sums of both sides from $n - 1$ spots:

```
def example(T:):
      count = 0
      for i in range(1, len(L)):
          if sum(T[:i]) == sum(T[i:]):
              count += 1
      return count
```

This algorithm gives us a right solution but how fast it is when it comes to largers lists? Let's create a small test program:

```
if __name__ == "__main__":
      N = 1
      T = [1, -1, 2, -2] * N
      print(example(T))
```

Below we see the results when we run the test program with `time` command:

$N = 100$	$N = 1000$	$N = 10000$

`$ time python3 example.py`
`199`
 	
`real 0m0.016s`
`user 0m0.016s`
`sys 0m0.000s`
	`$ time python3 example.py`
	`1999`
	
	`real 0m0.118s`
	`user 0m0.111s`
	`sys 0m0.007s`
		`$ time python3 example.py`
		`19999`
		
		`real 0m13.678s`
		`user 0m13.579s`
		`sys 0m0.036s`

Analyzing the function `example` we can see that its time complexity is $\Theta(n^2)$. Inside the `for`-loop we have two `sum` functions which are together $\Theta(n)$ making it $\Theta(n^2)$ algorithm. Is it possible to create a faster solution?

Counting sums over and over again from each position is unnecessary. Instead of that we can count cumulative sums from left to right and from right to left in two new lists L (Left) and R (Right). After that we count how many times $L_i = R_{i+1}$.

```
def example_better(T):
    count = 0
    length = len(T)
    L = [T[0]] * length     # cumulative sums from left
    R = [T[-1]] * length    # cumulative sums from right

    for i in range(1, length):
        L[i] = L[i - 1] + T[i]
        R[-(i + 1)] = R[-i] + T[-(i + 1)]

    for i in range(1, length):
        if L[i-1] == R[i]:
            count += 1

    return count
```

Since the new algorithm has only inpidual `for`-loops depended by the length of the list n the algorithm performs in $\Theta(n)$ time!

Now let's run the test program again with the new algorithm:

$N = 100$	$N = 1000$	$N = 10000$	$N = 100000$

`$ time python3 example.py`
`199`
 	
`real 0m0.017s`
`user 0m0.007s`
`sys 0m0.010s`
	`$ time python3 example.py`
	`1999`
	
	`real 0m0.015s`
	`user 0m0.015s`
	`sys 0m0.000s`
		`$ time python3 example.py`
		`19999`
		
		`real 0m0.027s`
		`user 0m0.027s`
		`sys 0m0.000s`
			`$ time python3 example.py`
			`199999`
			
			`real 0m0.127s`
			`user 0m0.127s`
			`sys 0m0.000s`

Although the new algorithm is faster its space complexity is three times larger. With modern computers this is not a problem but with limited resources and small input size choosing the first algorithm can be some times more reasonable.

Last modified: Monday, 18 September 2023, 11:53 AM