Examples and Applications of Binary Search

Tony Gong

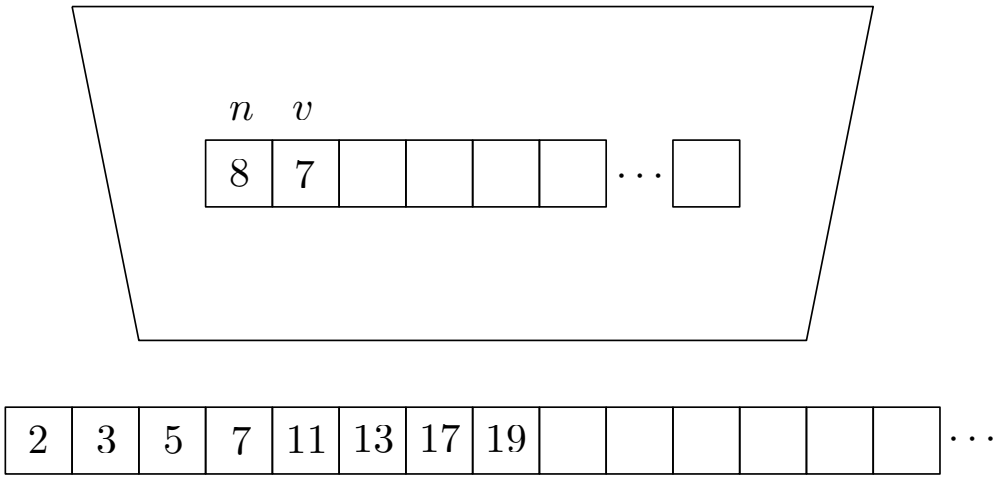
ITEE
University of Queensland

In the second lecture last week we studied the binary search algorithm that solves the problem of determining if a particular value appears in a sorted list of integer or not. We proved that the worst-case running time of the algorithm under the RAM model that we have defined is $2 + 6\log_2 n$, where n (the length of the sorted sequence) is the **problem size**.

Today we will look at some further examples of how the binary search algorithm runs to help us understand its behaviour.

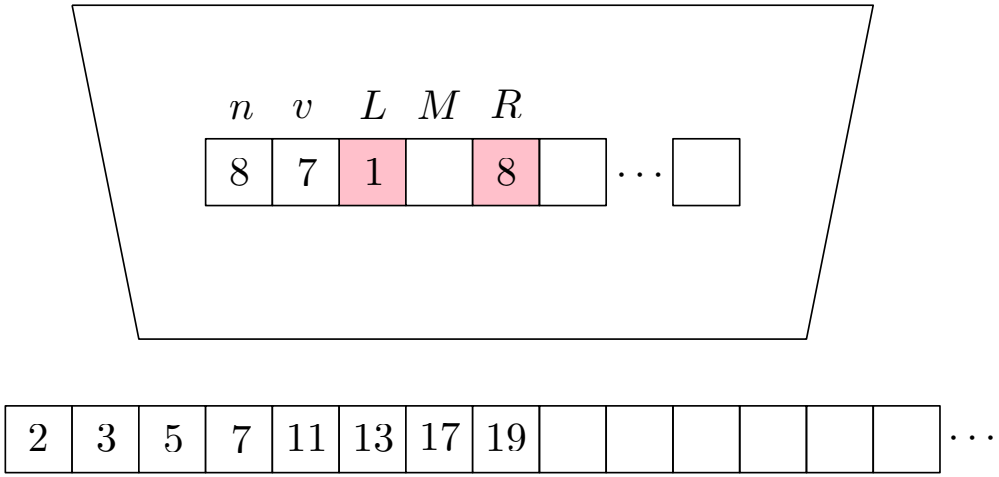
Example 1

Suppose we have the following input set, where $n = 8$ and we are trying to find the value 7.



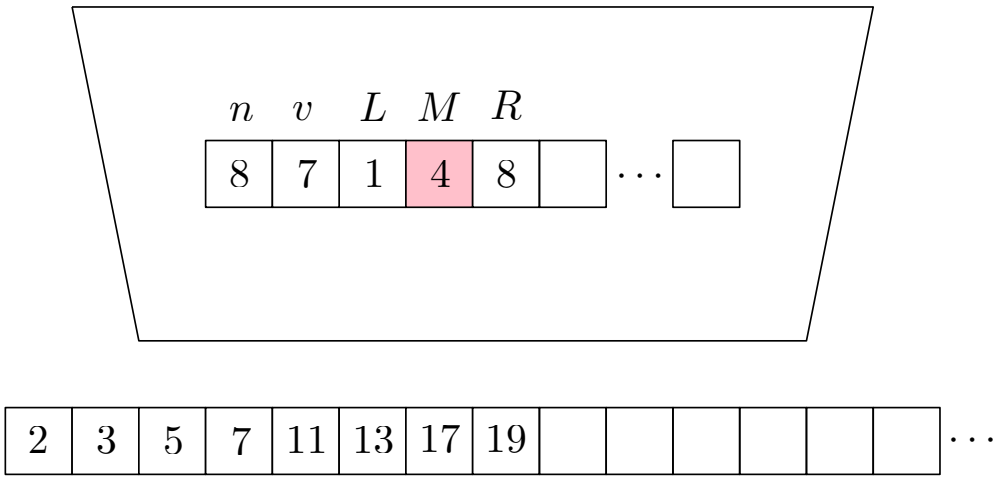
Example 1

We begin by initialising *left* to be 1 and *right* to *n* (in this case 8).



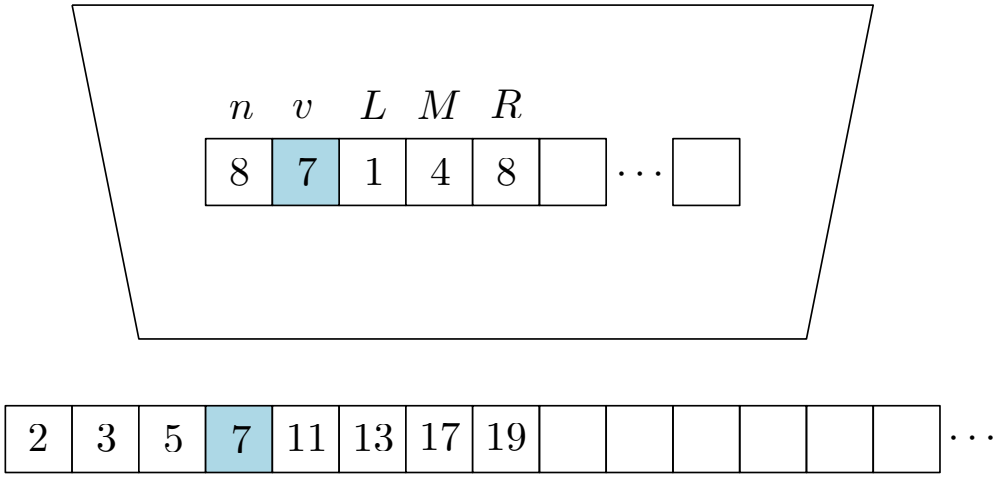
Example 1

Since the value of *right* is strictly greater than the value of *left*, we proceed by computing *mid*.



Example 1

We then do a comparison of the value we're looking for with the value in the sorted sequenced indexed by *mid*. We get lucky in this case and find that they are actually equal, meaning we can stop here and say that the value does appear in the sorted sequence.

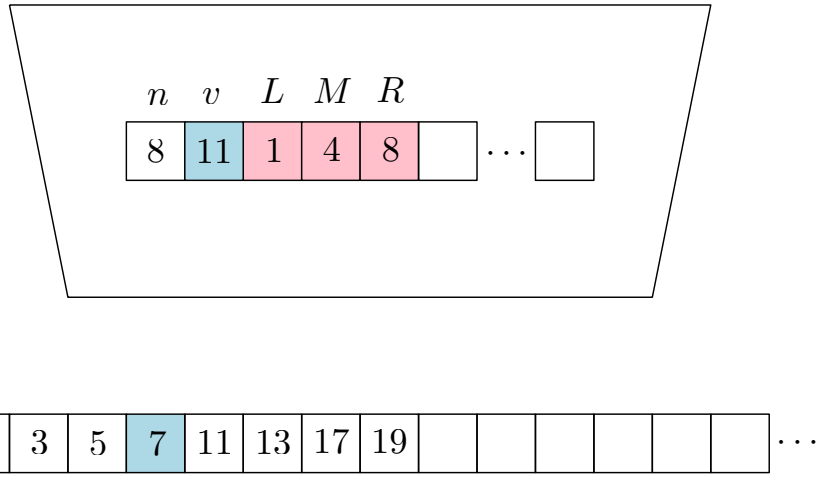


The previous example illustrates that the algorithm may terminate in far less time (if we get lucky) than $2 + 6 \log_2 n$. This worst-case running time that we proved gives us a guarantee on how fast our algorithm runs, and is independent of luck.

Let's look at another example.

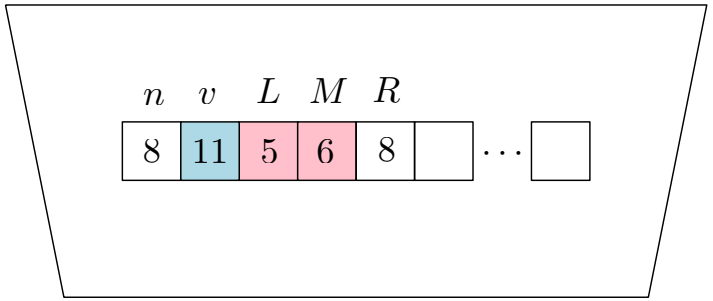
Example 2

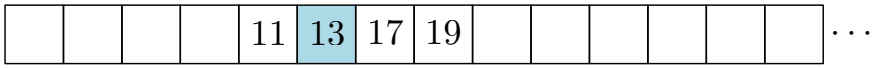
In this example, we keep the same sorted sequence but look for the value 11 instead. L and R get initialised as before, M gets computed and we perform the comparison to find that v is greater than the Mth term in the sequence.



Example 2

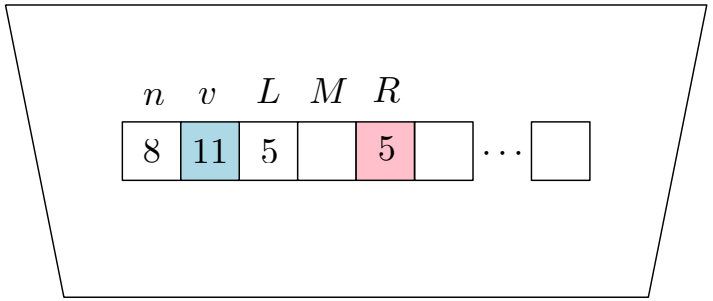
The means that what we are looking for is in the right half of the sorted sequence, hence we set L to be $M + 1$ (and conceptually discard the left half of the sequence). We then recompute M and do another comparison, this time finding that v is less than the value at M.

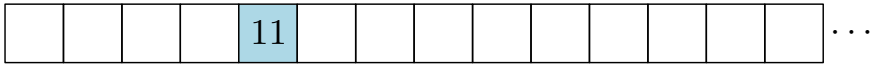




Example 2

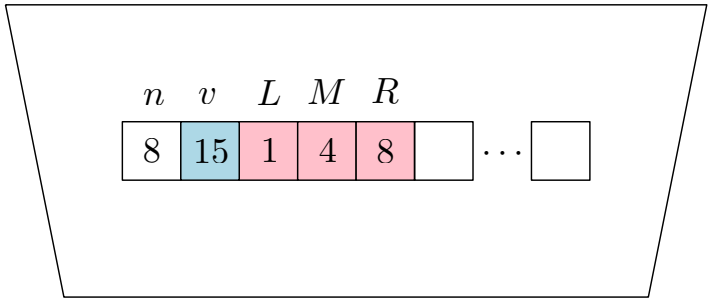
So now we adjust R (and throw away the right half) to be $M - 1$, and find that L and R have converged, meaning there's only a single value for us to check. Indeed we find that it is the value we are looking for, therefore the output in this case is "yes".

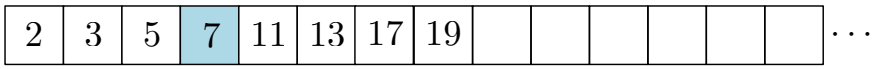




Example 3

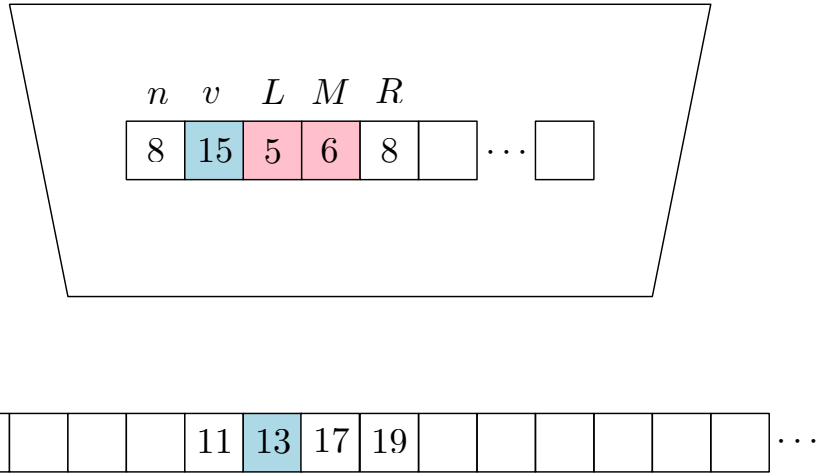
For this last example, we will look at a case where the value we are looking for does not appear in the sorted sequence. Here the first comparison tells us that v would have to be in the right half of the sequence (if it was to appear at all).





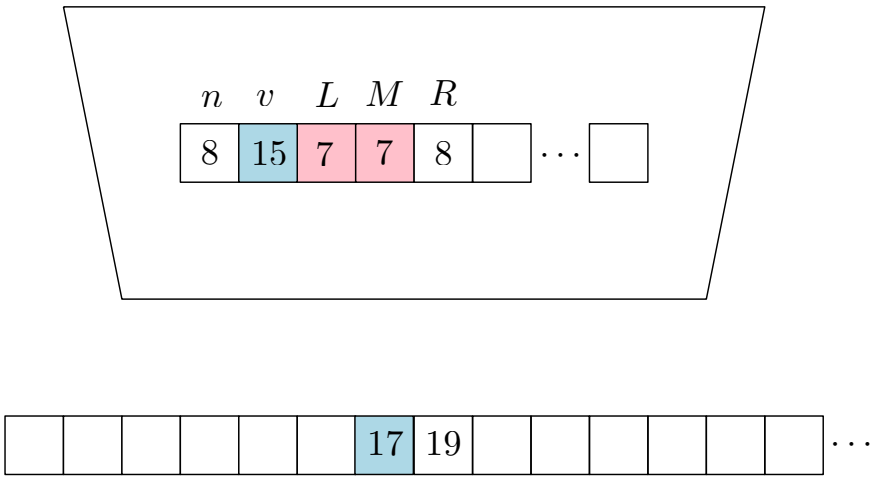
Example 3

The second comparison agains says that v must in the right half.



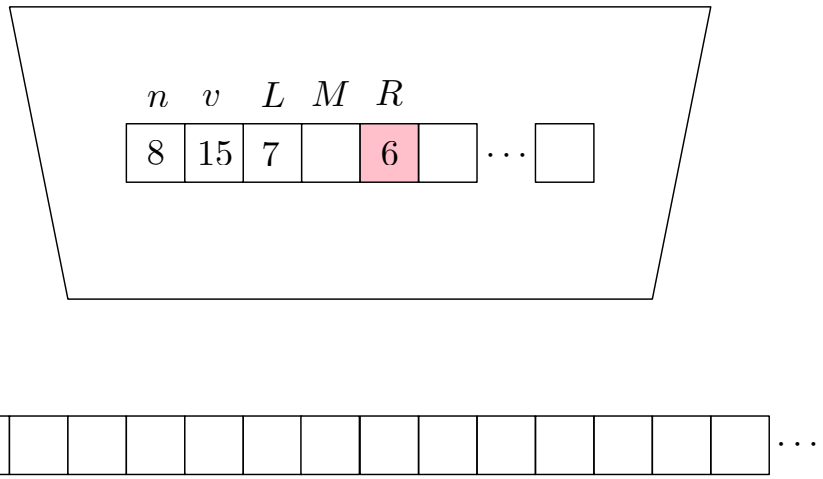
Example 3

The third comparison tells us that v must be to the left of the highlighted value of 17, but we see that there aren't any values to the left of 17...



Example 3

And so we arrive at the base case where we no longer have $L \leq R$, which tells us that what we are looking for does not appear in the sequence.



The Sum of Two Integers Problem

Now let's look at how binary search can be applied. Suppose we have the following problem:

Problem Input:

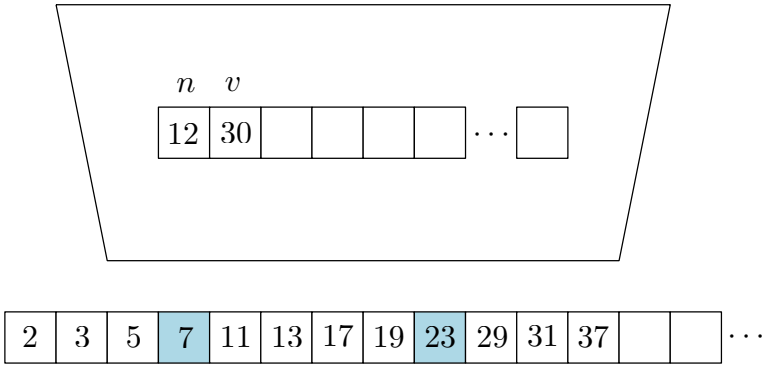
There is a sequence of n positive integers in strictly increasing order in memory at the cells numbered from 1 up to n. The value n has been placed in Register 1, and the a positive integer v has been placed in Register 2.

Goal:

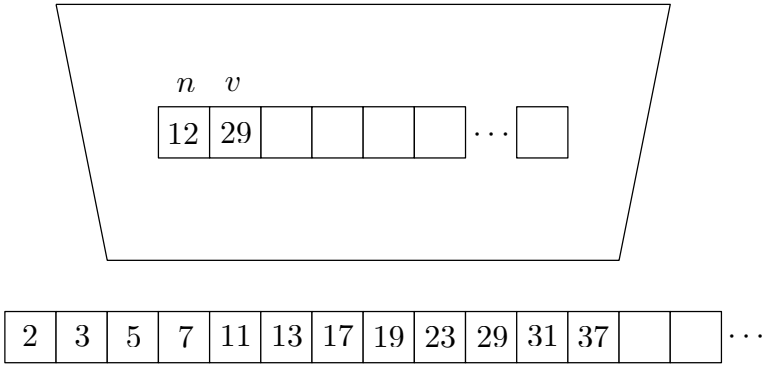
Determine whether if there exist two integers x and y (not necessarily distinct) in the sorted sequence such that $x + y = v$.

Example

A "yes"-input with $n = 12$



A "no"-input with $n = 12$



A First Attempt

A naïve algorithm to solve this problem is to enumerate all possible pairs in the sorted sequence and check if they sum to v. Given a sequence of length n, there are:

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

such pairs. This is on the order of n^2, but can we do better than this?

Hint: Take advantage of the fact that the given sequence is sorted!

Binary Search The Answer

If we rearranged the equation and put it in the form $y = v - x$, we can rephrase the problem in terms of whether if such a y exists in the sequence for every x in the sequence.

The idea is then to let x run over the sequence, compute y as $v - x$ and use binary search to see if y exists in the sequence.

The Repeated Binary Search Algorithm in Pseudocode

```
Let n be register 1 and v be register 2
register i ← 1, register one ← 1
while i ≤ n
    read into register x the memory cell at address i
    y ← v − x
    if BinarySearch(y) = "yes"
        return "yes"
    i ← i + one (effectively increasing i by 1)
return "no"
```

Running Time

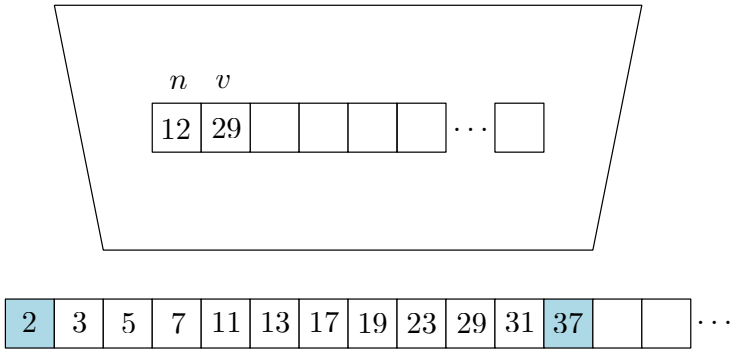
In the worst case (this happens when the output is "no"), the algorithm needs to run binary search n times. A precise count yields a time of $f(n) = 3 + 7n + 6n \log_2 n$. This is an improvement over the naïve algorithm we considered previously.

The story doesn't end there however... because we can do even better than this!

An Even Better Algorithm

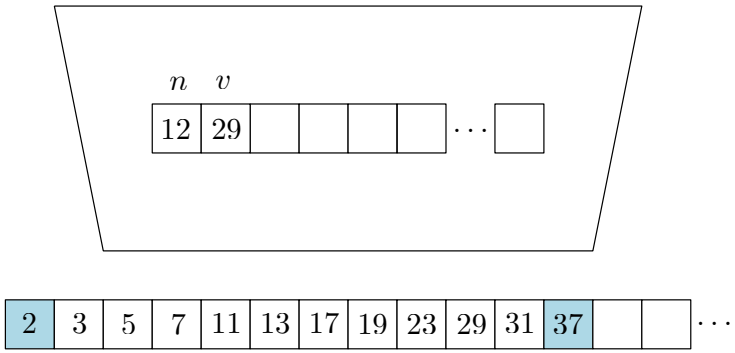
In fact the sortedness of our sequence of integers means that we can find the solution by considering each term only once!

The idea is as follows:



Conceptually we will have two pointers (as highlighed in blue), and they will begin by pointing at the start and the end respectively. If we summed the two numbers being pointed to, we get 39, which is greater than the desired value of 29.

An Even Better Algorithm

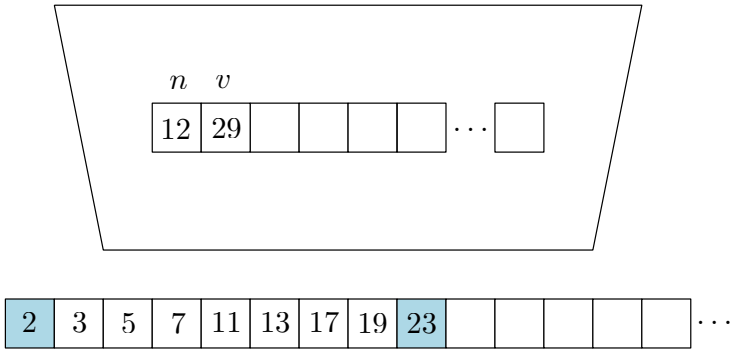


This tells us that:

- 37 will never appear in a valid solution, because 2 is the smallest term in the sequence and the sum is already greater than the desired value; and
- in order to get a solution, we should move the right pointer towards the left, since this will decrease the overall value of the sum.

An Even Better Algorithm

After finding that 31 and 29 are also too large, we arrive at the following:

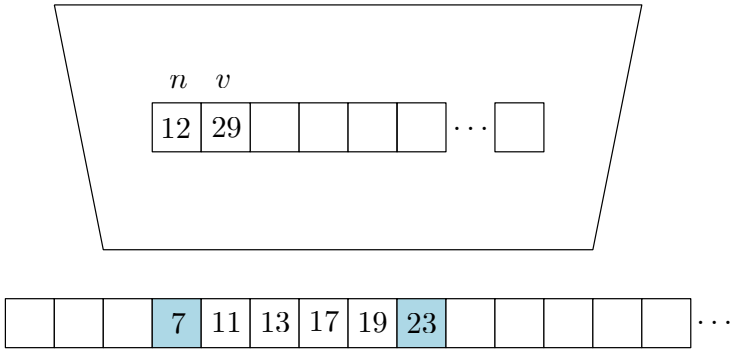


The sum is now too small, and what this tells us is that 2 never appears in a solution because 23 is the largest value that we have, and that the solution is still too small means that 2 plus any other number in the sequence would also be too small.

This means that we should move the left pointer towards the right to increase our "estimate".

An Even Better Algorithm

One shall find after moving the left pointer that 3 and 5 are also too small, arriving at:



And now it's too big and we need to move the right pointer, etc.

The Main Idea

Essentially, if we think of x as the element being pointed to by the left pointer and y as the element being pointed to by the right pointer:

- if $x + y = v$, we're done;
- if $x + y > v$, we need to make the sum smaller, so move y towards the left; and
- if $x + y < v$, we need to make the sum bigger, so move x towards the right.

Pseudocode

```
let n be register 1, and v be register 2
register left ← 1, right ← n
while left ≤ right
    read into register x the memory cell at address left
    read into register y the memory cell at address right
    if x + y = v then
        return "yes"
    else if x + y > v then
        right ← right − 1
    else
        left ← left + 1
return "no"
```

This algorithm has a worst-case running time of $5n + 4$.

Recap

In this week's tutorial, we reviewed the binary search algorithm and looked at a problem that can be solved by repeated application of binary search (although the best algorithm turned out to be even cleverer).

You are encouraged to run this algorithm on some input sets and convince yourself that it actually gives the correct result.