




[COMPUTING](#) · [COMPUTER SCIENCE](#) · [ALGORITHMS](#) · [BINARY SEARCH](#)

Implementing binary search of an array

 Google
Classroom

 Facebook

 Twitter

 Email

Let's see how to think about binary search on a sorted array. Yes, JavaScript already provides methods for determining whether a given element is in an array and, if it is, its location (as do many programming languages), but we want to implement it ourselves, to understand how you can implement such methods. Here's a JavaScript array of the first 25 prime numbers, in order:

```
var primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97];
```

Suppose we want to know whether the number 67 is prime. If 67 is in the array, then it's prime.

We might also want to know how many primes are smaller than 67. If we find the position of the number 67 in the array, we can use that to figure out how many smaller primes exist.

The position of an element in an array is known as its index. Array indices start at 0 and count upwards. If an element is at index 0 then it is the first element in the array. If an element is at index 3, then it has 3 elements which come before it in the array.

Looking at the example below, we can read the array of prime numbers from left to right, one at a time, until we find the number 67 (in the pink box) and see that it is at array index 18. Looking through the numbers in order like this is a *linear search*.

Once we know that the prime number 67 is at index 18, we can identify that it is a prime. We can also quickly identify that there are 18 elements which come before 67 in the array, meaning that there are 18 prime numbers smaller than 67.

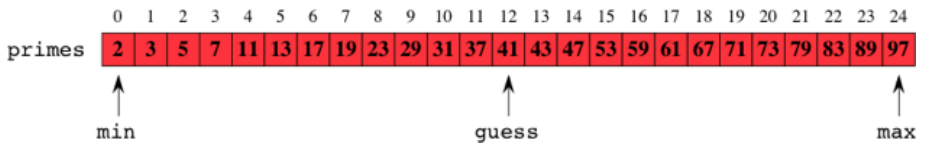
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53

67 = 67

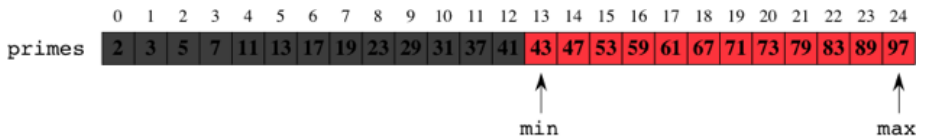


Did you see how many steps that took? A binary search might be more efficient. Because the array `primes` contains 25 numbers, the indices into the array range from 0 to 24. Using our pseudocode from before, we start by letting `min = 0` and `max =`

24. The first guess in the binary search would therefore be at index 12 (which is $(0 + 24) / 2$). Is `primes[12]` equal to 67? No, `primes[12]` is 41.



Is the index we are looking for higher or lower than 12? Since the values in the array are in increasing order, and $41 < 67$, the value 67 should be to the right of index 12. In other words, the index we are trying to guess should be greater than 12. We update the value of `min` to $12 + 1$, or 13, and we leave `max` unchanged at 24.



What's the next index to guess? The average of 13 and 24 is 18.5, which we round down to 18, since an index into an array must be an integer. We find that `primes[18]` is 67.



The binary search algorithm stops at this point, since it has found the answer. It took only two guesses, instead of the 19 guesses that linear search would have taken. You can step through that again in the visualization below:

Here's the pseudocode for binary search, modified for searching in an array. The inputs are the array, which we call `array`; the number `n` of elements in

`array`; and `target`, the number being searched for. The output is the index in `array` of `target`:

1. Let `min = 0` and `max = n-1`.
2. Compute `guess` as the average of `max` and `min`, rounded down (so that it is an integer).
3. If `array[guess]` equals `target`, then stop.
You found it! Return `guess`.
4. If the guess was too low, that is,
`array[guess] < target`, then set `min = guess + 1`.
5. Otherwise, the guess was too high. Set `max = guess - 1`.
6. Go back to step 2.

Implementing pseudocode

We'll alternate between English, pseudocode, and JavaScript in these tutorials, depending on the situation. As a programmer, you should learn to understand pseudocode and be able to turn it into your language of choice - so even though we're using JavaScript here, it should be straightforward for you to implement pseudocode using other languages.

How would we turn that pseudocode into a JavaScript program? We should create a function, because we're writing code that accepts an input and returns an output, and we want that code to be reusable for different inputs. The parameters to the function—let's call it `binarySearch`—will be the

array and target value, and the return value of the function will be the index of the location where the target value was found.

Now let's go into the body of the function, and decide how to implement that. Step 6 says to go back to step 2. That sounds like a loop. Should it be a for-loop or a while-loop? If you really wanted to use a for-loop, you could, but the indices guessed by binary search don't go in the sequential order that a for-loop makes convenient. First we might guess the index 12, and then 18, based on some computations. So a while-loop is the better choice.

There's also an important step missing in that pseudocode that didn't matter for the guessing game, but does matter for the binary search of an array. What should happen if the number you are looking for is *not* in the array? Let's start by figuring out what index the `binarySearch` function should return in this case. It should be a number that cannot be a legal index into the array. We'll use `-1`, since that cannot be a legal index into any array. (Actually, any negative number would do.)

The target number isn't in the array if there are no possible guesses left. In our example, suppose that we're searching for the target number 10 in the `primes` array. If it were there, 10 would be between the values 7 and 11, which are at indices 3 and 4. If you trace out the index values for `min` and `max` as the `binarySearch` function executes, you would

find that they eventually get to the point where `min` equals 3 and `max` equals 4. The guess is then index 3 (since $(3 + 4) / 2$ equals 3.5, and we round down), and `primes[3]` is less than 10, so that `min` becomes 4. With both `min` and `max` equaling 4, the guess must be index 4, and `primes[4]` is greater than 10. Now `max` becomes 3. What does it mean for `min` to equal 4 and `max` to equal 3? It means that the only possible guesses are at least 4 and at most 3. There are no such numbers! At this point, we can conclude that the target number, 10, is not in the `primes` array, and the `binarySearch` function would return `-1`. In general, once `max` becomes strictly less than `min`, we know that the target number is not in the sorted array. Here is modified pseudocode for binary search that handles the case in which the target number is not present:

1. Let `min = 0` and `max = n-1`.
2. If `max < min`, then stop: target is not present in array. Return `-1`.
3. Compute `guess` as the average of `max` and `min`, rounded down (so that it is an integer).
4. If `array[guess]` equals `target`, then stop. You found it! Return `guess`.
5. If the guess was too low, that is, `array[guess] < target`, then set `min = guess + 1`.
6. Otherwise, the guess was too high. Set `max = guess - 1`.

7. Go back to step 2.

Now that we've thought through the pseudocode together, you're going to try implementing binary search yourself. It's fine to look back at the pseudocode - in fact, it's a good thing, because then you'll have a better grasp of what it means to convert pseudocode into a program.

This content is a collaboration of [Dartmouth Computer Science](#) professors [Thomas Cormen](#) and [Devin Balkcom](#), plus the Khan Academy computing curriculum team. The content is licensed [CC-BY-NC-SA](#).

Ask a question...

Questions Tips & Thanks

[Top](#) [Recent](#)

Why did we round down to 18 from 18.5? One can also round up to 19. In case we round it to 19, the efficiency will decrease greatly. Is there any rule to generalize?



• 52 votes ▲ ▼ • 3 comments • Flag

3 years ago by [dhakal.gareema](#)

We have $\text{min}=13$ $\text{max}=24$ $\text{avg}=(\text{min}+\text{max})/2=18.5$

Suppose we round down:

guess=18

If the item we are searching for is $<$ guess we can eliminate $24-18+1=7$ choices

If the item we are searching for is $>$ guess we can eliminate $18-13+1=6$ choices

If the item we are searching for is $=$ guess we eliminate all choices

Suppose we round up:

guess=19

If the item we are searching for is $<$ guess we can eliminate $24-19+1=6$ choices

If the item we are searching for is $>$ guess we can eliminate $19-13+1=7$ choices

If... [\(more\)](#)



• 273 votes ▲ ▼ • 12 comments • Flag

3 years ago by  Cameron

[Show all 9 answers](#) • [Answer this question](#)

Hi, sorry. I'm somehow stuck on the first part of the Binary search challenge. Here is the orange message and my code: "It looks like you almost have the correct condition on the while loop, but something is still wrong with it."

[Edited to only include relevant code]

```
while (max > min)
```

What is wrong with the condition in the while loop?



• 33 votes ▲ ▼ • 6 comments • Flag

3 years ago by  Mickey O

Hi, I think your code is good but you missed out an equal sign from your while condition. It should be : while (max \geq min) {
..
}




• 27 votes ▲ ▼ • 10 comments • Flag

3 years ago by  Lilla Csanaky

[Show all 5 answers](#) • [Answer this question](#)

What if you are searching an unsorted (you cannot sort it, because it's precise order is in some way vital to the program) array for a value? Would you temporarily sort it so you could do a binary search, or is there some special kind of search algorithm for this?

 • 12 votes ▲ ▼ • 1 comment • Flag

3 years ago by  Joshua P Gammage I de los Estados Unidos de America

Linear search is a very good method in this case. When you sort things you almost always have to look at each and every item at least once which will take the same amount of time as linear search. If at all possible you want to store data in some sort of order so you can operate on it more quickly later. If it's random, linear search is about as good as it's going to get, but if you know anything about the "precise order" you can probably come up with a faster way to do it.

 • 21 votes ▲ ▼ • 2 comments • Flag

3 years ago by  iandanforth

[Show all 5 answers](#) • [Answer this question](#)

In the pseudocode for binary search, modified for searching in an array. It says

"Let min = 0 and max = n-1." Can someone explain why the -1 is there? Thanks!

9 votes ▲ ▼ • Comment • Flag 3 years ago by  valeriamettler

The -1 is there because the first array index is 0. So if we have n elements in the array, the last element will be at index n-1.

e.g. the primes array in the article has 25 numbers. The index of the first element is 0, and the index of the last element is 24.

If you didn't subtract the 1 and instead looked at index n, you would be outside of the array.

Hope this makes sense

 • 21 votes ▲ ▼ • 2 comments • Flag

3 years ago by  Cameron

[Show all 2 answers](#) • [Answer this question](#)

What does it mean when it asks if all my assertions passed?

 • 10 votes ▲ ▼ • 1 comment • Flag

3 years ago by  Adam Henson

It means there is something wrong with your code.

Here's what worked for me:

```
var doSearch = function(array, targetValue) {  
  var min = 0;  
  var max = array.length - 1;  
  var guess;  
  while(max >= min){  
    guess = floor((max + min)/2);  
    if(array[guess] === targetValue){return guess;}  
    else if(array[guess]< targetValue){min = guess +1;}  
    else {max = guess - 1;}  
  }  
  
  return -1;  
};
```

```
var primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,  
41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97];
```

```
var result = doSearch(... \(more\))
```

6 votes ▲ ▼ • 1 comment • Flag

6 months ago by  Alexis Jensen

[Show all 2 answers](#) • [Answer this question](#)

I passed all of the steps, but number of guesses shows 0.
Why is this?

[Edited to show only code necessary for question]

Code used to increment counter for number of guesses:

```
gcnt = gcnt++;
```

6 votes ▲ ▼ • Comment • Flag

3 years ago by  D.j. Hensley

Short Answer :

The problem is:

```
gcnt= gcnt++;
```

to achieve the desired effect of incrementing gcnt by 1
you want:

```
gcnt++;
```

Long Answer :

A glance behind the curtain as to what is happening:

In Javascript and Java if we write `i++` it gets translated
into something like:

```
postincrement(i)
```

where the postcriment function is:

```
var postincrement=function(x){  
    var temp=x;  
    x=x+1; //but this writes over  
the original variable, not just the  
local copy  
    return temp;  
};
```

Suppose `gcnt=0` and we execute:

```
gcnt=gcnt++
```

... [\(more\)](#)

 • 16 votes ▲ ▼ • 2 comments • Flag

3 years ago by  Cameron

[Show all 2 answers](#) • [Answer this question](#)

I am on the final step, and I actually passed it, but I don't understand the `Program.assertEqual(doSearch(primes, 34) 20);` part. I wrote the same thing underneath with only changing the numbers, but I get a message that says

"assertion error: 14 is not equal to n" where n is the number that I chose as the final parameter in that Program.assert...(doSearch...) command.

How do I figure out that it wanted 20, then 14, then 15 without it having to tell me?

6 votes ▲ ▼ • 1 comment • Flag

2 years ago by  Mr. Morse

I had the same problem : from what I understood the assertEquals function will just see if the number returned by your function, which is put as the first parameter in Program.assertEquals here, [doSearch(primes,73)] and the second parameter [20] are the same. It will return an error if they aren't in this exercise ! So just count in the array where your prime number is indexed, and put that as your second parameter. :) If it's not a prime, the second parameter should be -1.

6 votes ▲ ▼ • Comment • Flag

2 years ago by  Tristan Duquesne

"Let min = 0 and max = n-1."

Why do you set max to n-1? Why not n?

4 votes ▲ ▼ • Comment • Flag

2 years ago by  Rideron

If max was set to n it would point to a location outside of the array. In this instance a zero indexed array is being used, meaning numbering starts at zero instead of 1 (which would be a one indexed array and also what most people are used to) and has values at indexes 0 to n-1.

8 votes ▲ ▼ • Comment • Flag

2 years ago by  Lyra Wolvespaw

I have a problem in step 3 of the challenge: Binary search, my code already prints the total number of guesses after the target is found, but the editor doesn't allow me to pass to the next step and they don't say the error or if it's right here's the code

[Edited to only show relevant code]

```
var doSearch = function(array, targetValue)
{
    ...
    if (guess >= min || guess <=
max){
        nGuesses ++;
    }
    ...
};
```

7 votes ▲ ▼ • 1 comment • Flag

3 years ago by  Miguel C. Martins

Your if statement is confusing the grader. The guess calculation will always give you a guess that is between min and max (inclusive), so the if statement is unnecessary (it will always be true). Take `nGuesses++` out of the if statement and it should work fine.

4 votes ▲ ▼ • 1 comment • Flag

3 years ago by  Cameron

Should it be a for-loop or a while-loop?

what is a conceptual difference between these two *loops*?
Please explain in a very simple way :D

4 votes ▲ ▼ • Comment • Flag


A for loop and a while loop can do the same things, but traditionally we use for loops when we want to use a counter which has a fixed starting value, ending value and step size. This comes from languages similar to BASIC where a for loop would be structured like this:

```
FOR J = 1 TO 10 STEP 1
  REM do some stuff
NEXT J
```

Its equivalent in JavaScript would be:

```
for ( j = 1; j <= 10; j += 1) {
  //do some stuff
}
```

When we reserve for loops for these types of situations it makes it clearer what the... [\(more\)](#)

2 votes ▲ ▼ • 4 comments • Flag
about a year ago by  Cameron

[Show all 2 answers](#) • [Answer this question](#)

Show more comments