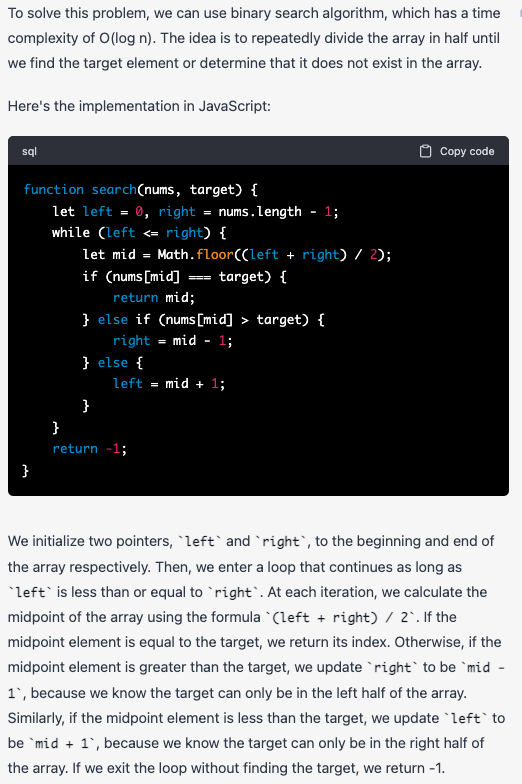
Search- In its simplest form, Binary Search operates on a contiguous sequence with a specified left and right index. This is called the Search Space. Binary Search maintains the left, right, and middle indicies of the search space and compares the search target or applies the search condition to the middle value of the collection; if the condition is unsatisfied or values unequal, the half in which the target cannot lie is eliminated and the search continues on the remaining half until it is successful. If the search ends with an empty half, the condition cannot be fulfilled and target is not found.

TLDR: halves the collection and searches

T{ O(logn):



TLDR= set left and right, while L is <=R, find M, if M=T then return M, if Middle>T, set R to inside first half, if M<T set L to inside latter half.

Binary search should be considered anytime searching for index or element. if unordered, sort first.

**3 Parts of a Successful Binary Search**Binary Search is generally composed of 3 main sections:

1. *Pre-processing* - Sort if collection is unsorted.
2. *Binary Search* - Using a loop or recursion to divide search space in half after each comparison.
3. *Post-processing* - Determine viable candidates in the remaining space.

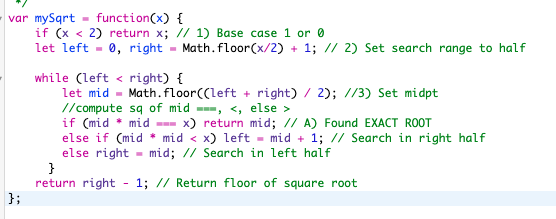
**Template 1:**

Key Attributes:

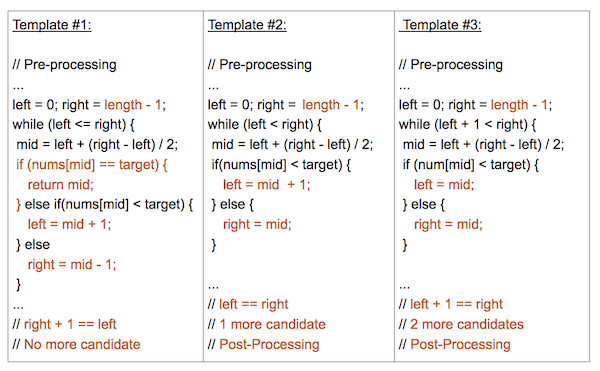
* Most basic and elementary form of Binary Search
* Search Condition can be determined without comparing to the element's neighbors (or use specific elements around it)
* No post-processing required because at each step, you are checking to see if the element has been found. If you reach the end, then you know the element is not found

Distinguishing Syntax:

* Initial Condition: left = 0, right = length-1
* Termination: left > right
* Searching Left: right = mid-1
* Searching Right: left = mid+1



Template analysis:

[https://leetcode.com/explore/learn/card/binary-search/136/template-analysis/935/](https://leetcode.com/explore/learn/card/binary-search/136/template-analysis/935/*)

Template #1 (left <= right):

* Most basic and elementary form of Binary Search
* Search Condition can be determined without comparing to the element's neighbors (or use specific elements around it)
* No post-processing required because at each step, you are checking to see if the element has been found. If you reach the end, then you know the element is not found

Template #2 (left < right):

* An advanced way to implement Binary Search.
* Search Condition needs to access the element's immediate right neighbor
* Use the element's right neighbor to determine if the condition is met and decide whether to go left or right
* Guarantees Search Space is at least 2 in size at each step
* Post-processing required. Loop/Recursion ends when you have 1 element left. Need to assess if the remaining element meets the condition.

Template #3 (left + 1 < right):

* An alternative way to implement Binary Search
* Search Condition needs to access element's immediate left and right neighbors
* Use element's neighbors to determine if the condition is met and decide whether to go left or right
* Guarantees Search Space is at least 3 in size at each step
* Post-processing required. Loop/Recursion ends when you have 2 elements left. Need to assess if the remaining elements meet the condition.