SEARCH ALGORITHMS

Searching algorithms are used to identify the location of a specified item within an array (list) of items.

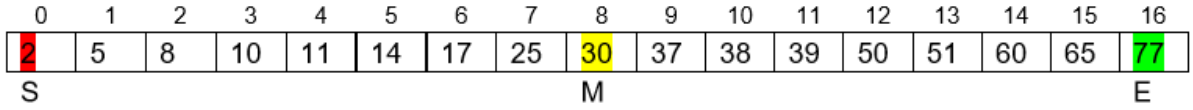
BINARY SEARCH

A binary search finds a value by continually halving a **sorted list** until a target is, or is not, found.

The code begins by designating a start (S) point and an end (E) point in the list. These are initially the first and last elements of the array.

From these, the target value positioned in the middle of the sorted list is identified (M=(E-S)/2).

Example: **Target = 8**

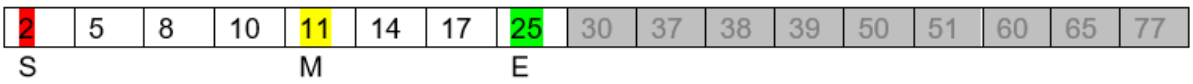


The algorithm compares the target to the value stored at M and makes one of three decisions:

1. If the middle value is larger than the target, then the target must be in the half of the list that contains smaller values.
2. If the middle value is smaller, the target must be in the larger half of the list.
3. If the middle value is equal to the target, then the position of the target has been identified and the search ends.

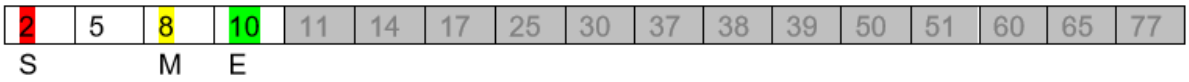
If either bullet points 1 or 2 are true, then the start or end are reassigned as required. The middle point is then calculated for the remaining list and the same decision is made again.

Target = 8



This is carried out again, until a match is found at M.

Target = 8



Binary Search – The Algorithm

Procedure:

```
PROCEDURE binary_search(list,target)
     DECLARE low INITIALLY 0
     DECLARE high INITIALLY length(list)-1
     DECLARE mid INITIALLY 0
     DECLARE found INITIALLY FALSE
     WHILE NOT found AND low <= high
          SET mid TO (low+high)/2
          IF target = list[mid] THEN
               SEND "Found at position"&mid TO DISPLAY
               SET found TO TRUE
          ELSE IF target > list[mid] THEN
               SET low TO mid+1
          ELSE
               SET high TO mid-1
          END IF
     END WHILE
     IF found = FALSE THEN
          SEND "Target not found" TO DISPLAY
     END IF
END PROCEDURE
```

Function:

```
FUNCTION binary_search(list,target) RETURNS INTEGER
     DECLARE low INITIALLY 0
     DECLARE high INITIALLY length(list)-1
     DECLARE mid INITIALLY 0
     DECLARE found INITIALLY FALSE

WHILE NOT found AND low <= high
     SET mid TO (low+high)/2
     IF target = list[mid] THEN
          SET position TO mid
          SET found TO TRUE
     ELSE IF target > list[mid] THEN
          SET low TO mid+1
     ELSE
          SET high TO mid-1
     END IF
END WHILE
RETURN position END FUNCTION
```

Binary Search – Pseudocode and Description

Design	Commentary
`low = 0`	The lowest index point (S) is stored
`high = length(list)-1`	The highest index point (E) is stored
`found = false`	Set a flag variable to show that a match has not yet been found
`while not found and low <= high`	Conditional loop until the target is found or there are no elements left to examine
`  set mid = (low + high) / 2`	Find the midpoint (M) as halfway between the lowest and highest index
`  if target = list[mid] then`	If a match with the target is found…
`    set position = mid`	…store the position of the match…
`    set found to true`	...and end the conditional loop using the flag variable

`  else if target > list[mid]`
`    set low = mid + 1`
`  else`
`    set high = mid - 1`
`  end if`
	Reset the lowest or highest index depending on whether the target is greater or smaller than the value in the middle index

`end while`	

`If found = false then`
`  Display "not found"`
`end if`
	An optional 'not found' may be added to the end of the algorithm, if required