Linear search of a collection

This algorithm traverses a collection from its first element (zero index) to its last item. It compares each element in the collection to an input *key*. If the *key* and the collection element are the same, the algorithm stops searching and returns the location (index) of the element within the collection. Otherwise, the algorithm returns -1 if the end of the collection is reached and the key is not found.

In the algorithm:
the method/function returns the index of the key element in the collection, or -1 if not found
COLLECTION = array or list to be searched
LENGTH = number of items (size) of the COLLECTION

Assume that the contents of COLLECTION and LENGTH have already been established.

```
1.    function search(ARRAY, KEY) : integer
2.          set INDEX to 0
3.          loop while INDEX < LENGTH
4.                if ARRAY[INDEX] = KEY then
5.                      return INDEX
6.                end if
7.                set INDEX to INDEX + 1
8.          end loop
9.          return -1
10.   end function
```

```
1.    function search(COLLECTION, KEY) : integer
2.          set INDEX to 0
3.          loop while COLLECTION.hasNext()
4.                if COLLECTION.get(INDEX)[1] = KEY then
5.                      return INDEX
6.                end if
7.                set INDEX to INDEX + 1
8.          end loop
9.          return -1
10.   end function
```

[1] `.get`*`Data`*`()` or similar may be required as well.

Binary search

"A binary search locates the middle element of the array first, and determines if the element being searched for is in the first or second half of the table. The search then points to the middle element of the relevant half table, and the comparison is repeated. This technique of continually halving the number of elements under consideration is continued until the data item being searched for is found, or its absence is detected" (Robertson 95).

In the algorithm:
the method/function returns the index of the key element in the collection, or -1 if not found
COLLECTION = array or list to be searched
LENGTH = number of items (size) of the COLLECTION
LOW = lower index or left index of a sub-array where the key is searched
HIGH = highest index or right index of a sub-array where the key is searched

Assume that the contents of COLLECTION and LENGTH have already been established. COLLECTION ***must*** be sorted in ascending order (small to large).

"The binary search will continue until the data item has been found, or there can be no more halving operations; that is, LOW is not less than HIGH" (Robertson 96).

```
1.      function binarySearch(ARRAY, KEY) : integer
2.             set LOW to 0
3.             set HIGH to LENGTH
4.             loop while LOW < HIGH
5.                  set INDEX to (LOW + HIGH) / 2
6.                  if KEY = ARRAY[INDEX] then
7.                        return INDEX
8.                  else
9.                       if KEY < ARRAY[INDEX] then
10.                     set HIGH to INDEX – 1
11.                else
12.                     set LOW  to INDEX + 1
13.                end if
14.            end if
15.       end loop
16.       return -1
17.   end function
```

```
1     function binarySearch(COLLECTION, KEY) : integer
2           set FOUND to false
3           set LOW to 0
4           set HIGH to LENGTH
5           loop while not FOUND and (LOW < HIGH)
6                 set INDEX to (LOW + HIGH) / 2
7                 if COLLECTION.get(INDEX)² = KEY then
8                       set FOUND to true
9                       set LOCATION to INDEX
10                      exit loop
11                else
12                      if KEY < COLLECTION.get(INDEX) then
13                            set HIGH to INDEX – 1
14                      else
15                            set LOW  to INDEX + 1
16                      end if
17                end if
18          end loop
19          return LOCATION
20    end function
```

`.get`*`Data`*`()` or similar may be required as well.

Recursive Binary Search (HL)

The binary search algorithm may also be implemented recursively due to the way it works, reducing the search scope by half every time. Note the base case (line 2) and recursive calls (lines 9 and 11).

Again, the method/function returns the index of the key element in the collection, or -1 if not found.

```
1     function rBinarySearch(ARRAY, LOW, HIGH, KEY) : integer
2           if LOW > HIGH then
3                 return -1
4           end if
5           INDEX = (LOW + HIGH) div 2
6           if ARRAY[INDEX] == KEY then
7                 return INDEX
8           else if KEY < ARRAY[INDEX] then
9                 return rBinarySearch(ARRAY, LOW, INDEX-1, KEY)
10          else
11                return rBinarySearch(ARRAY, INDEX+1, HIGH, KEY)
12          end if
13    end function
```

From previous examples, it should be straightforward to convert the code above to work with a collection (such as an ArrayList or similar) instead of an array.

Bibliography
(and material adapted from)

GeeksforGeeks. "Binary Search - Data Structure and Algorithm Tutorials." *GeeksforGeeks*, Sanchhaya Education Private Limited, 26 July 2023, www.geeksforgeeks.org/binary-search/.

Rana, Kanak. "Recursive Binary Search." *Coding Ninjas Studio*, 23 Sept. 2023, www.codingninjas.com/studio/library/implementation-of-binary-search-in-recursive-manner.

Robertson, Lesley Anne. *Simple Program Design: A Step-by-step Approach*. 5th ed. Southbank: Thomson, 2007. Print.