10 Searching

For COMSC 132

Sam Bowne

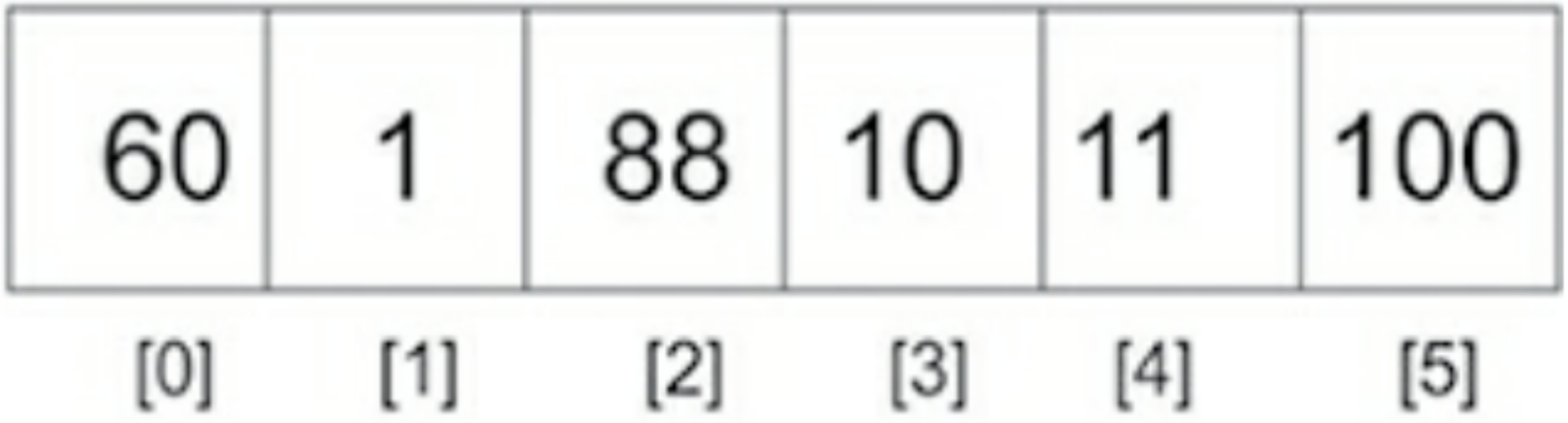
Oct 12, 2024

Topics

- Linear search
- Jump search
- Binary search
- Interpolation search
- Exponential search

Linear search

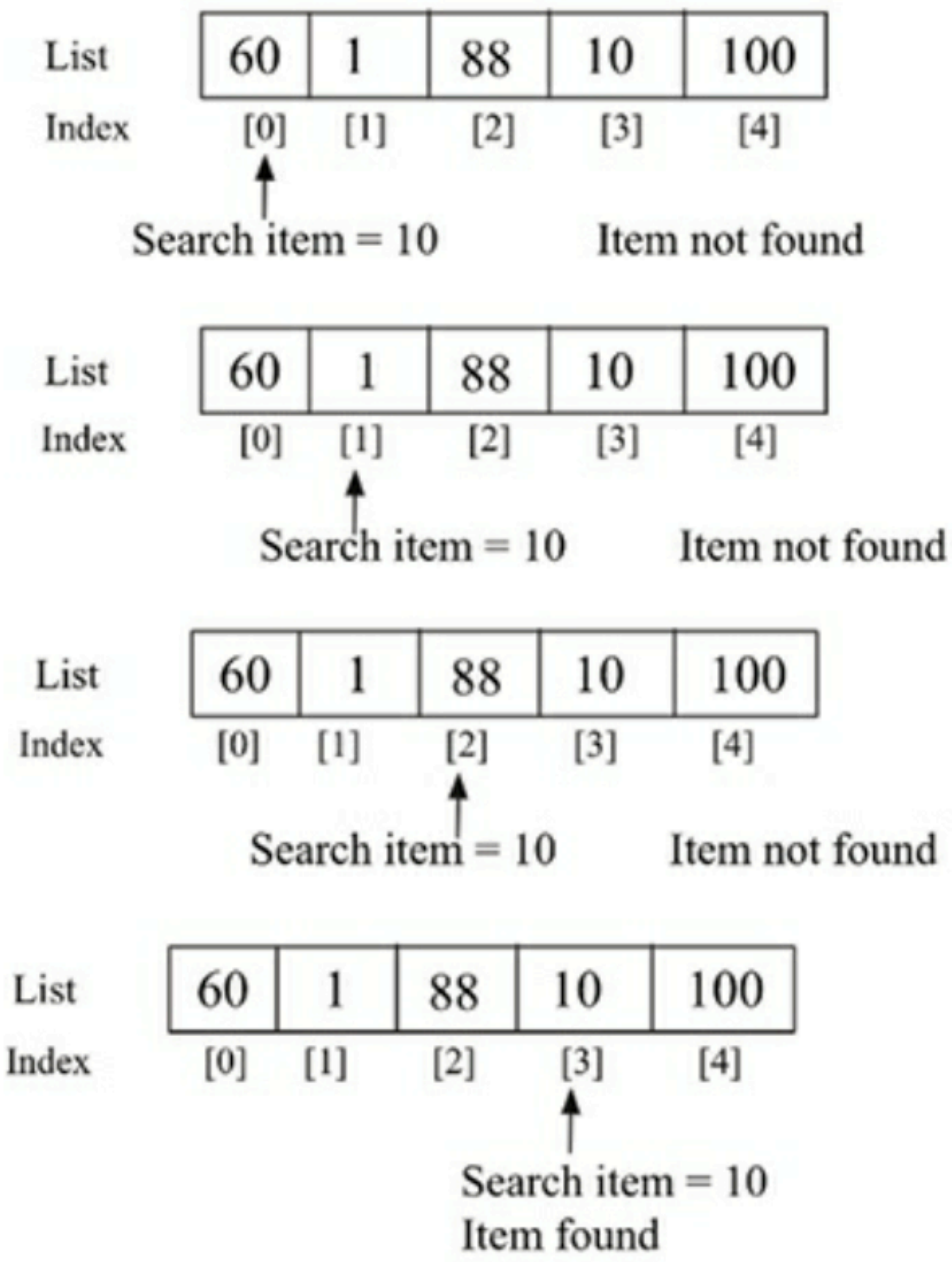
- Compare items one by one to the desired value
- Complexity O(***n***)



60	1	88	10	11	100
[0]	[1]	[2]	[3]	[4]	[5]

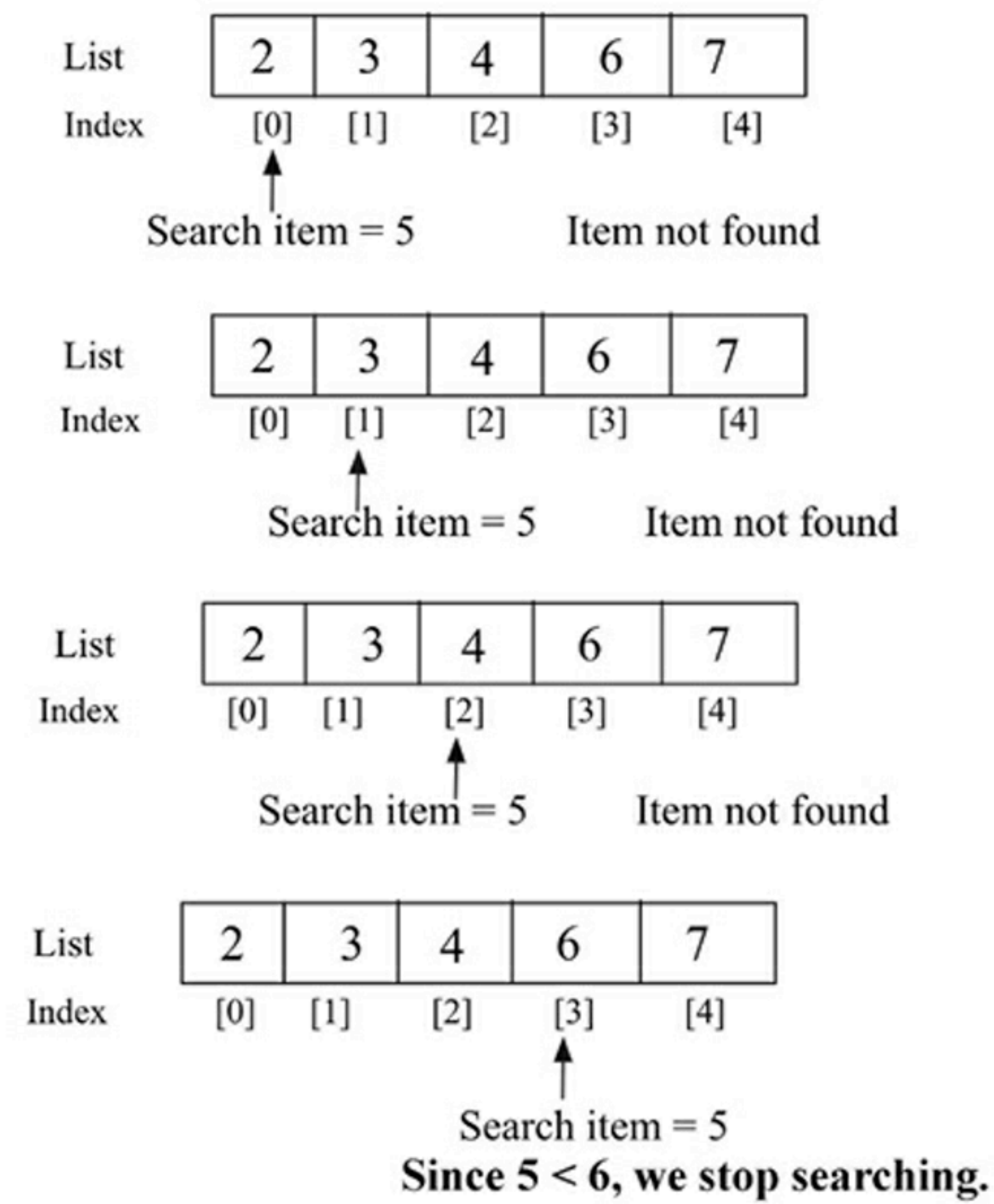
Unordered linear search

- Complexity O(***n***)



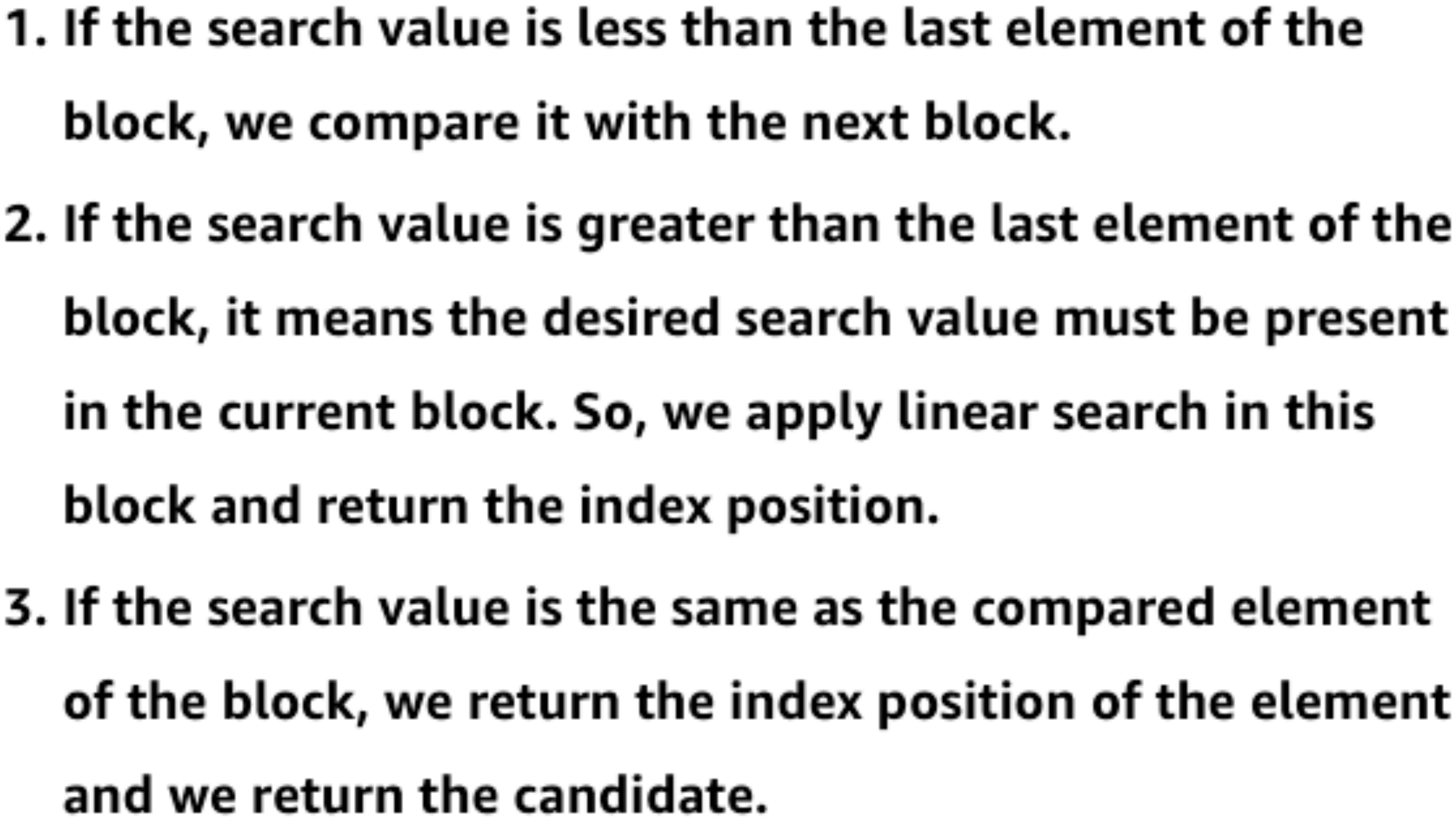
Ordered linear search

- Complexity O(***n***)



Jump search

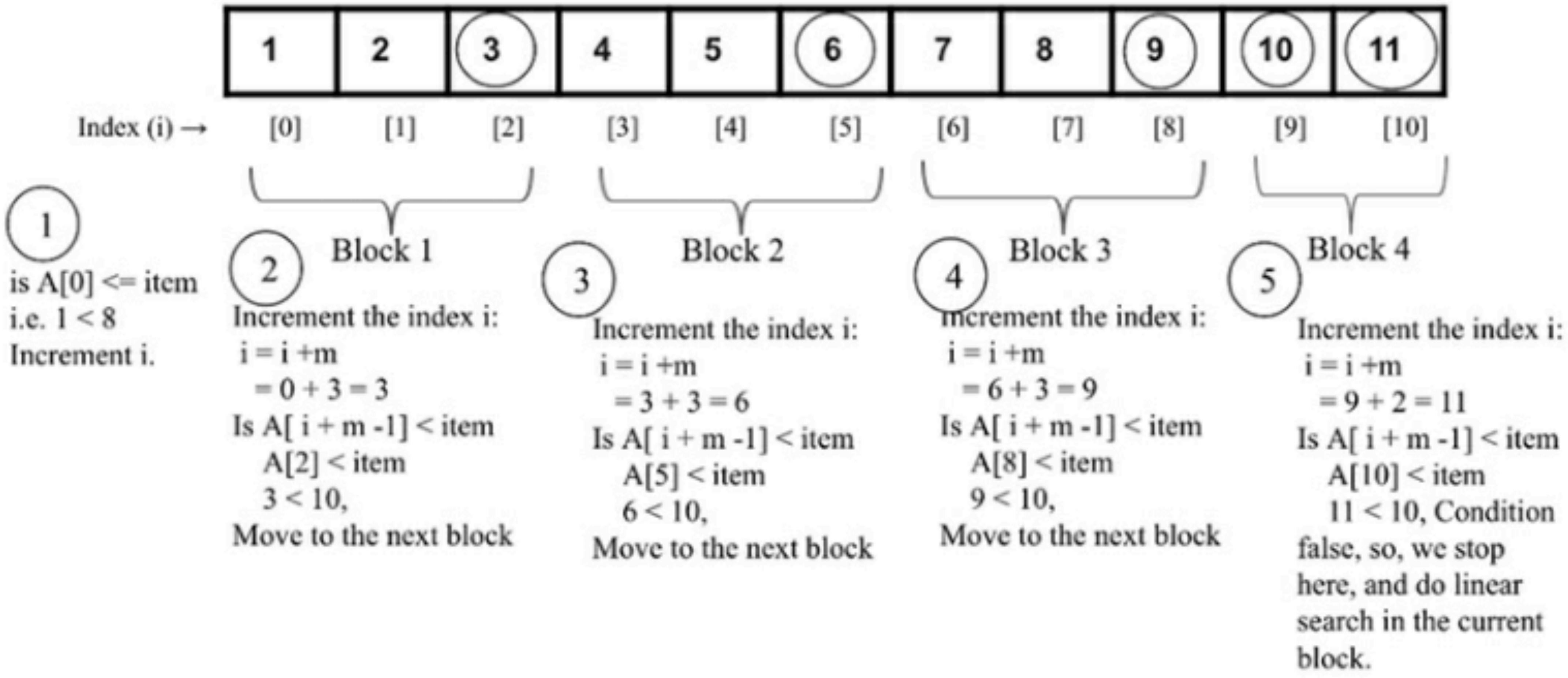
- Requires ordered list
- Divide list into **blocks** of size sqrt(***n***)



1. If the search value is less than the last element of the block, we compare it with the next block.
2. If the search value is greater than the last element of the block, it means the desired search value must be present in the current block. So, we apply linear search in this block and return the index position.
3. If the search value is the same as the compared element of the block, we return the index position of the element and we return the candidate.

Jump search

- Complexity O(sqrt(***n***))



	1	2	3	4	5	6	7	8	9	10	11
Index (i) →	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]

1
is A[0] <= item
i.e. 1 < 8
Increment i.

2
Increment the index i:
i = i +m
= 0 + 3 = 3
Is A[i + m -1] < item
A[2] < item
3 < 10,
Move to the next block

3
Increment the index i:
i = i +m
= 3 + 3 = 6
Is A[i + m -1] < item
A[5] < item
6 < 10,
Move to the next block

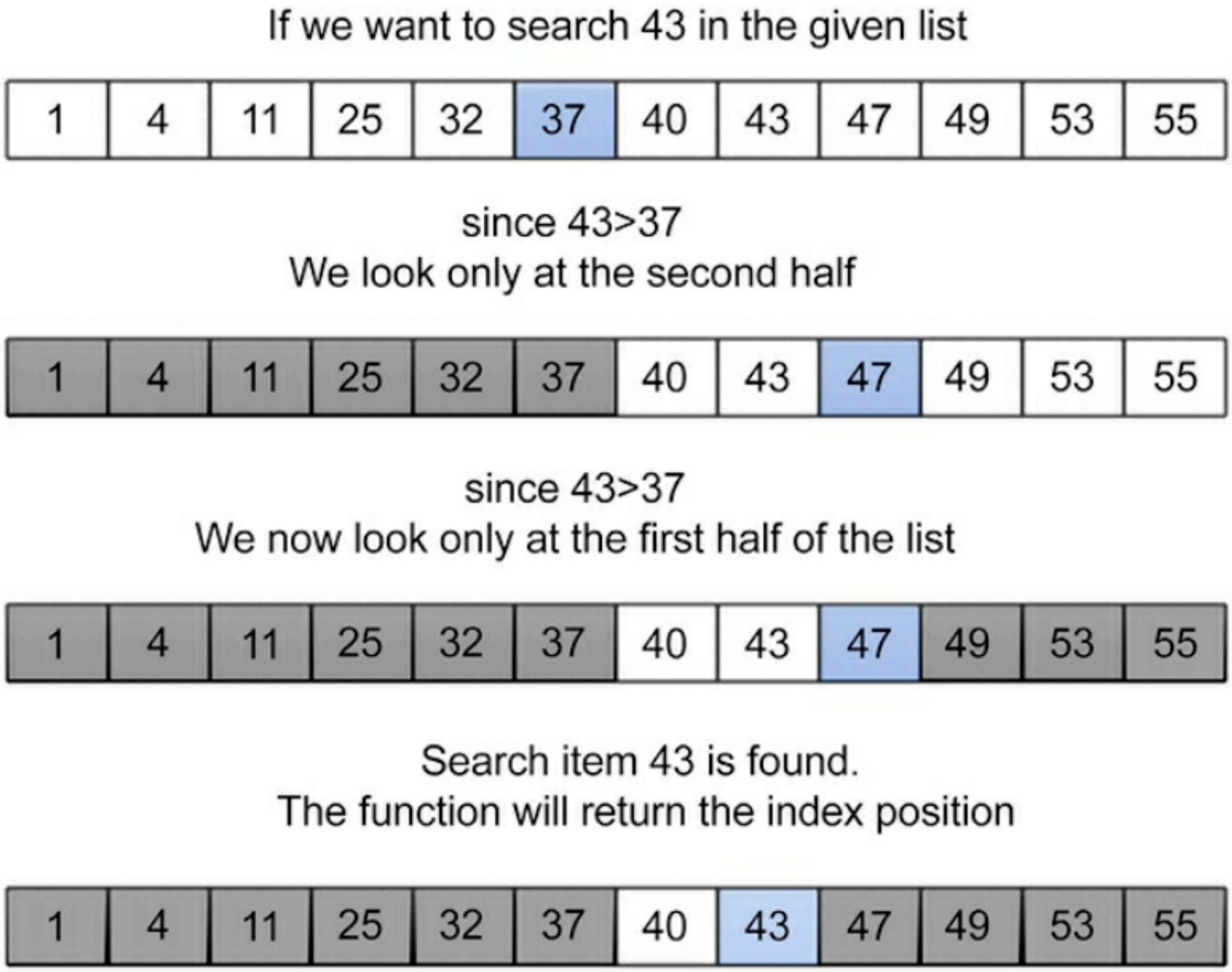
4
Increment the index i:
i = i +m
= 6 + 3 = 9
Is A[i + m -1] < item
A[8] < item
9 < 10,
Move to the next block

5
Increment the index i:
i = i +m
= 9 + 2 = 11
Is A[i + m -1] < item
A[10] < item
11 < 10, Condition false, so, we stop here, and do linear search in the current block.

Binary search

- List must be sorted
- Compare middle value in list to target value
- This limits the search to one half of the list
- Each comparison halves the list size
- Complexity O(log(***n***))

Binary search



If we want to search 43 in the given list

1	4	11	25	32	37	40	43	47	49	53	55

since 43>37
We look only at the second half

1	4	11	25	32	37	40	43	47	49	53	55

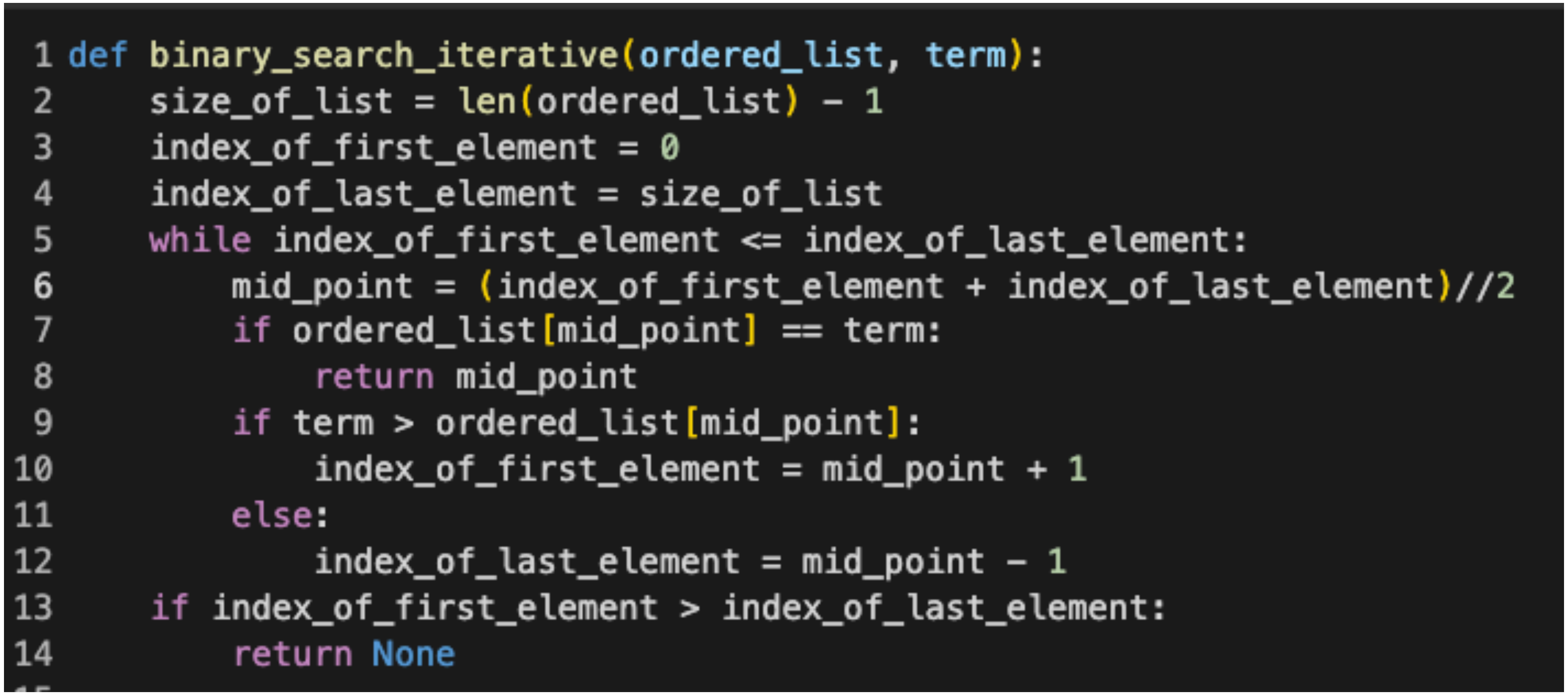
since 43>37
We now look only at the first half of the list

1	4	11	25	32	37	40	43	47	49	53	55

Search item 43 is found.
The function will return the index position

1	4	11	25	32	37	40	43	47	49	53	55

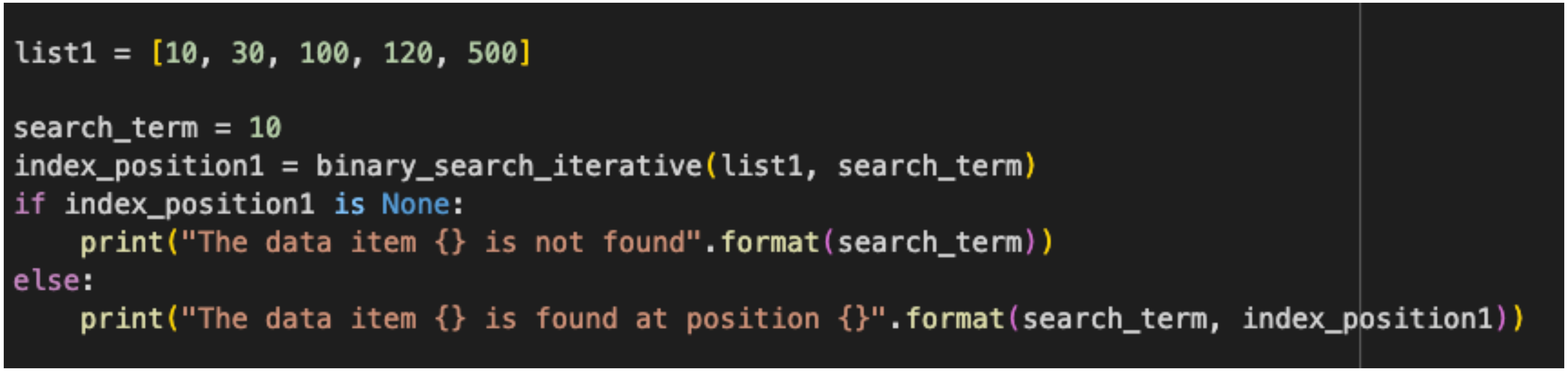
Binary search - iterative



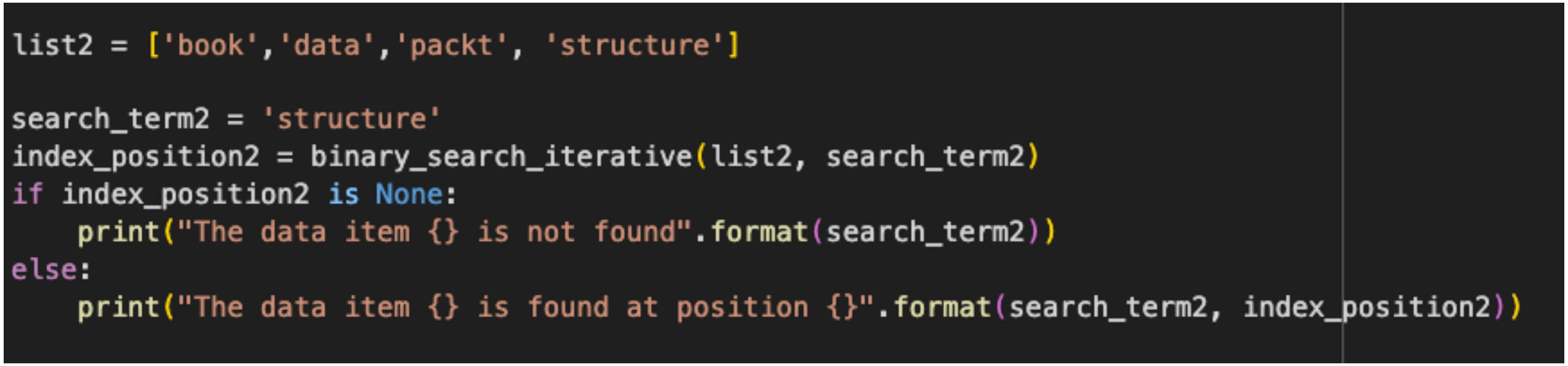
```python
def binary_search_iterative(ordered_list, term):
    size_of_list = len(ordered_list) - 1
    index_of_first_element = 0
    index_of_last_element = size_of_list
    while index_of_first_element <= index_of_last_element:
        mid_point = (index_of_first_element + index_of_last_element)//2
        if ordered_list[mid_point] == term:
            return mid_point
        if term > ordered_list[mid_point]:
            index_of_first_element = mid_point + 1
        else:
            index_of_last_element = mid_point - 1
    if index_of_first_element > index_of_last_element:
        return None
```

- Correction in book, division must be //

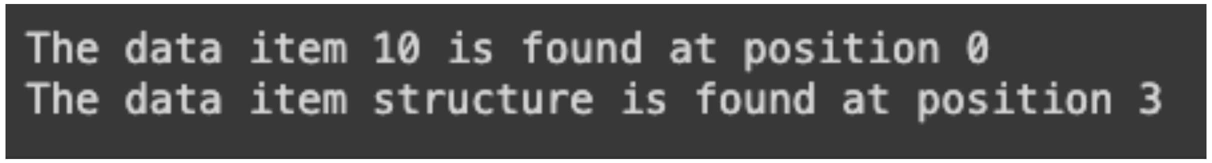
Binary search



```python
list1 = [10, 30, 100, 120, 500]

search_term = 10
index_position1 = binary_search_iterative(list1, search_term)
if index_position1 is None:
    print("The data item {} is not found".format(search_term))
else:
    print("The data item {} is found at position {}".format(search_term, index_position1))
```



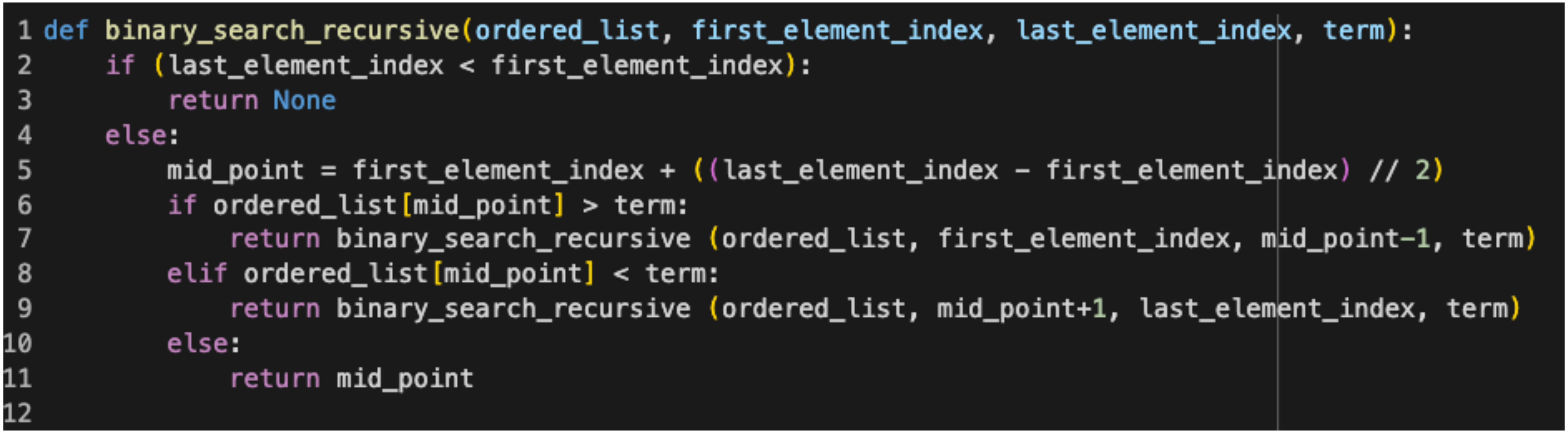
```python
list2 = ['book','data','packt', 'structure']

search_term2 = 'structure'
index_position2 = binary_search_iterative(list2, search_term2)
if index_position2 is None:
    print("The data item {} is not found".format(search_term2))
else:
    print("The data item {} is found at position {}".format(search_term2, index_position2))
```



```
The data item 10 is found at position 0
The data item structure is found at position 3
```

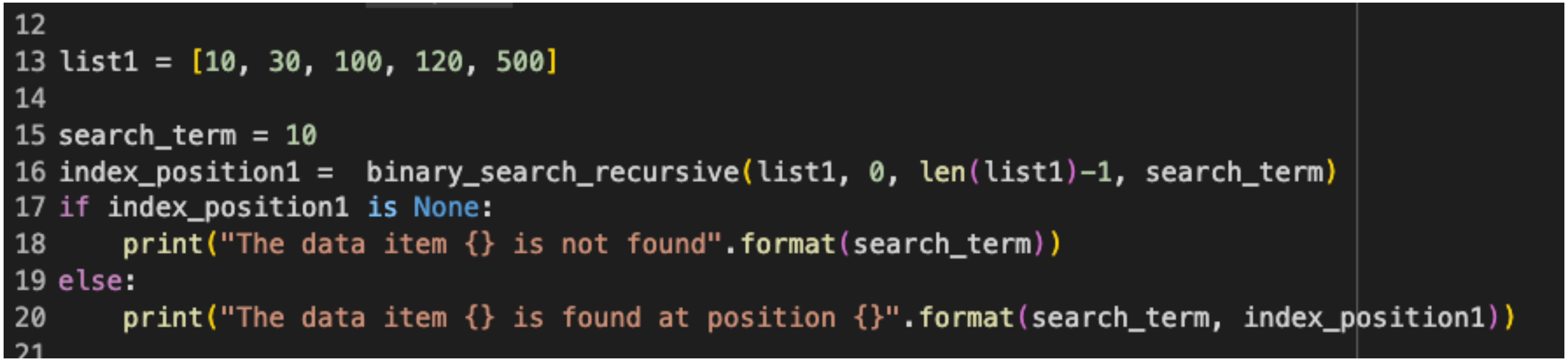
Binary search - recursive

- Function calls itself

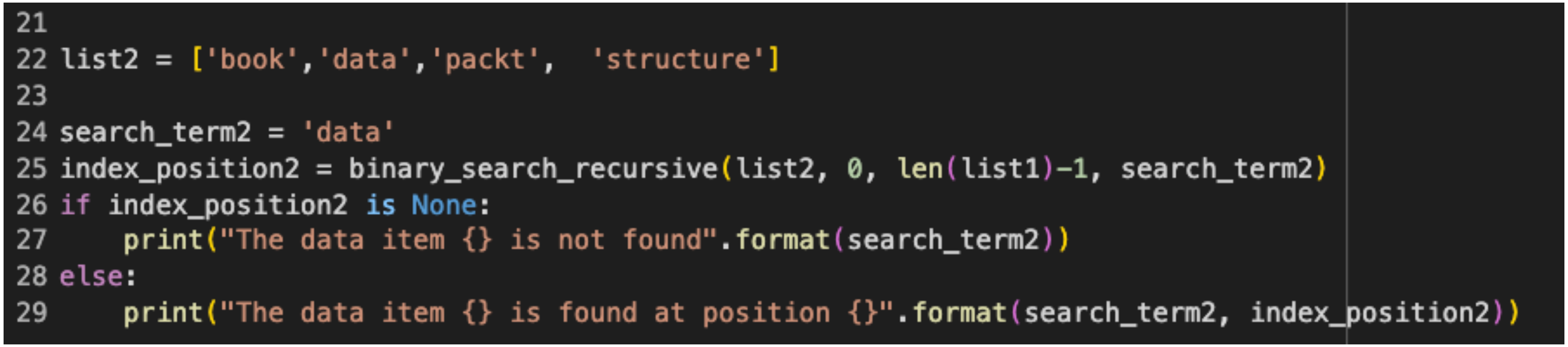


```
def binary_search_recursive(ordered_list, first_element_index, last_element_index, term):
    if (last_element_index < first_element_index):
        return None
    else:
        mid_point = first_element_index + ((last_element_index - first_element_index) // 2)
        if ordered_list[mid_point] > term:
            return binary_search_recursive (ordered_list, first_element_index, mid_point-1, term)
        elif ordered_list[mid_point] < term:
            return binary_search_recursive (ordered_list, mid_point+1, last_element_index, term)
        else:
            return mid_point

```

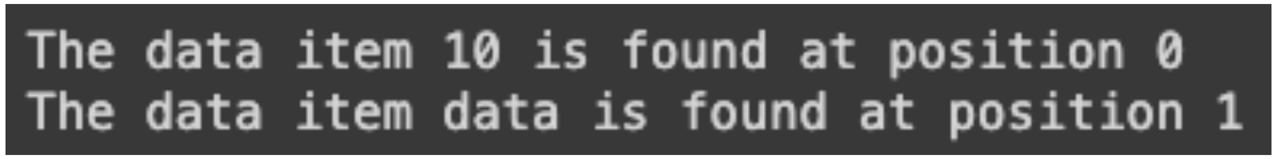
Binary search - recursive



```python

list1 = [10, 30, 100, 120, 500]

search_term = 10
index_position1 =  binary_search_recursive(list1, 0, len(list1)-1, search_term)
if index_position1 is None:
    print("The data item {} is not found".format(search_term))
else:
    print("The data item {} is found at position {}".format(search_term, index_position1))
```



```python

list2 = ['book','data','packt',  'structure']

search_term2 = 'data'
index_position2 = binary_search_recursive(list2, 0, len(list1)-1, search_term2)
if index_position2 is None:
    print("The data item {} is not found".format(search_term2))
else:
    print("The data item {} is found at position {}".format(search_term2, index_position2))
```



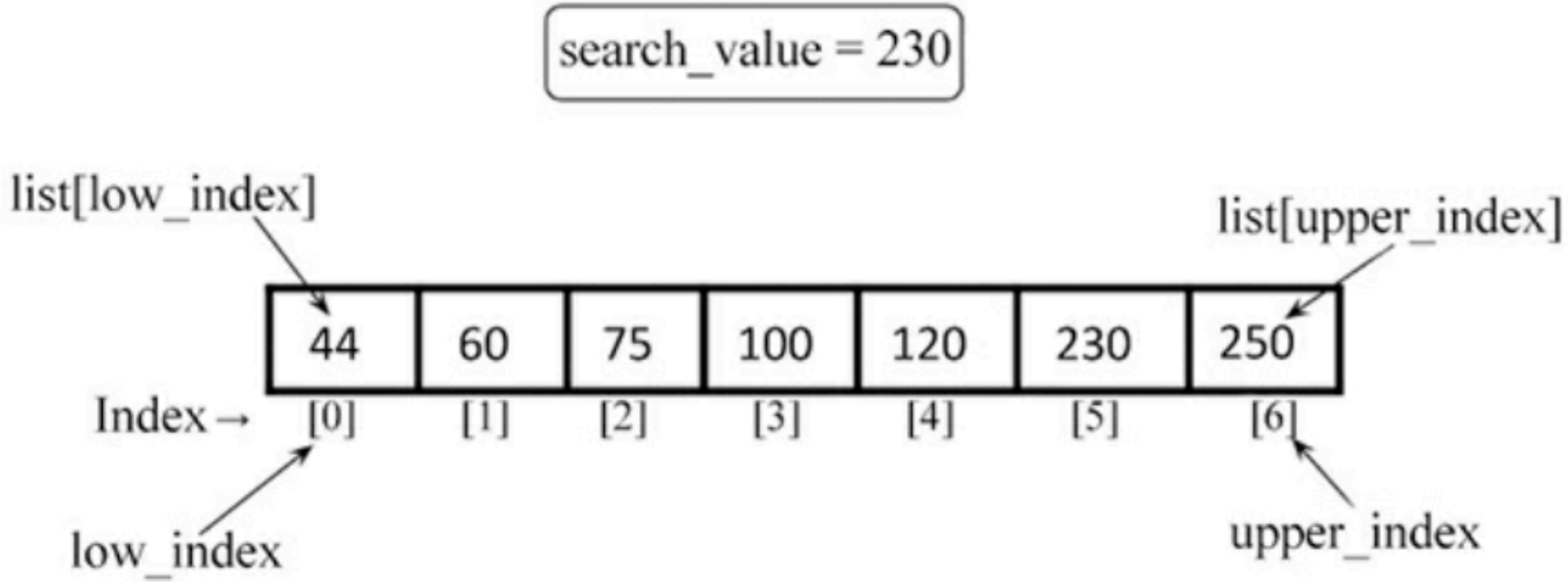
```
The data item 10 is found at position 0
The data item data is found at position 1
```

Interpolation search

- Try to predict the location of the target
  - Assuming that the data is uniformly distributed
- If target is near the start or end of the list
  - This reduces the list by more than half

$$mid = low\_index + \frac{(upper\_index - low\_index)}{(list[upper\_index] - list[low\_index])} * (search\_term - list[low\_index])$$

Interpolation search



search_value = 230

list[low_index]

list[upper_index]

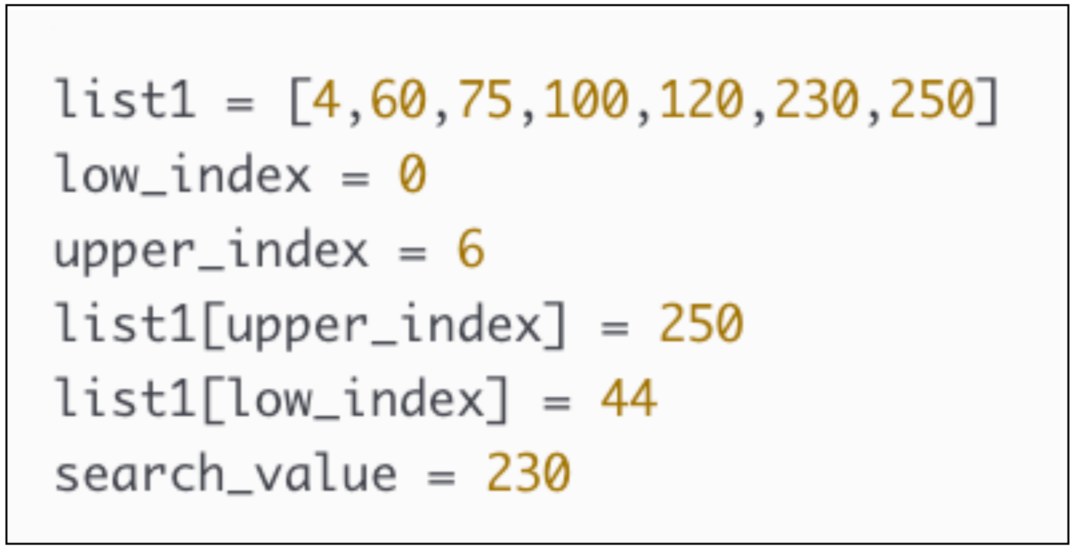
Index →	[0]	[1]	[2]	[3]	[4]	[5]	[6]
	44	60	75	100	120	230	250

low_index

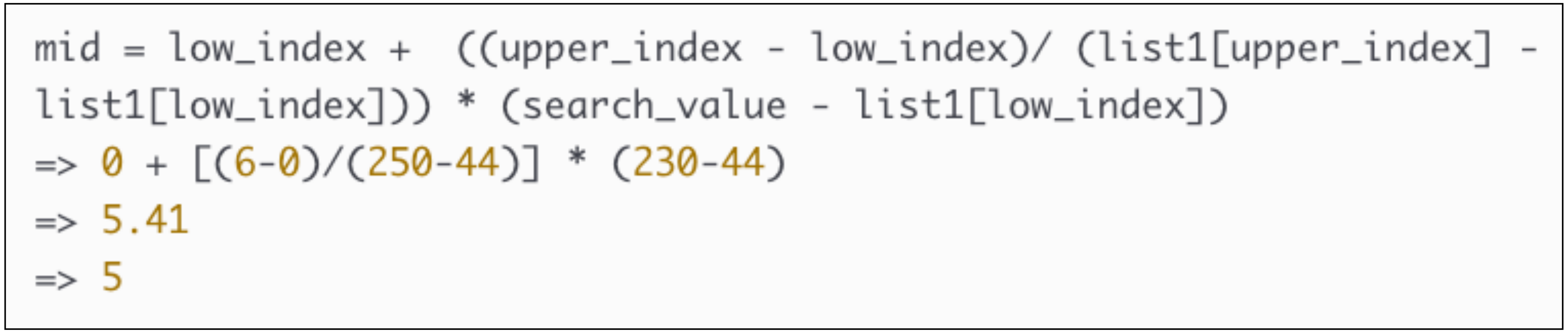
upper_index

Interpolation search

- Converges rapidly if data is uniformly distributed
- Complexity O(log(log(***n***))



```
list1 = [4,60,75,100,120,230,250]
low_index = 0
upper_index = 6
list1[upper_index] = 250
list1[low_index] = 44
search_value = 230
```



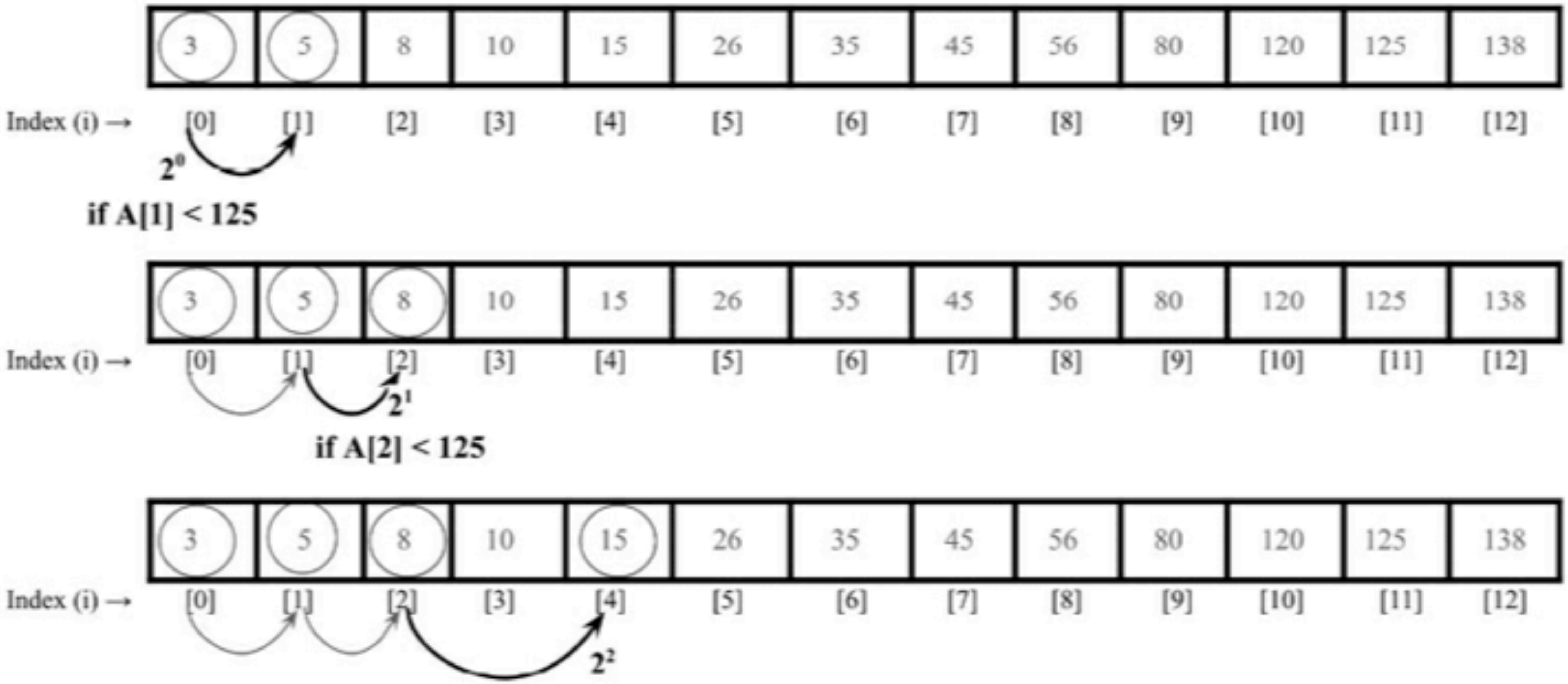
```
mid = low_index +  ((upper_index - low_index)/ (list1[upper_index] -
list1[low_index])) * (search_value - list1[low_index])
=> 0 + [(6-0)/(250-44)] * (230-44)
=> 5.41
=> 5
```

Exponential search

- Mostly used for large datasets
- List must be sorted
- Compare target to items at these index values:
    - 1, 2, 4, 8, 16, ... 2^i
- This locates a sublist to search in
    - Use a binary search in the sublist

Exponential search

Assume the item to be searched is 125



	3	5	8	10	15	26	35	45	56	80	120	125	138
Index (i) →	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]

2^0

if A[1] < 125

	3	5	8	10	15	26	35	45	56	80	120	125	138
Index (i) →	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]

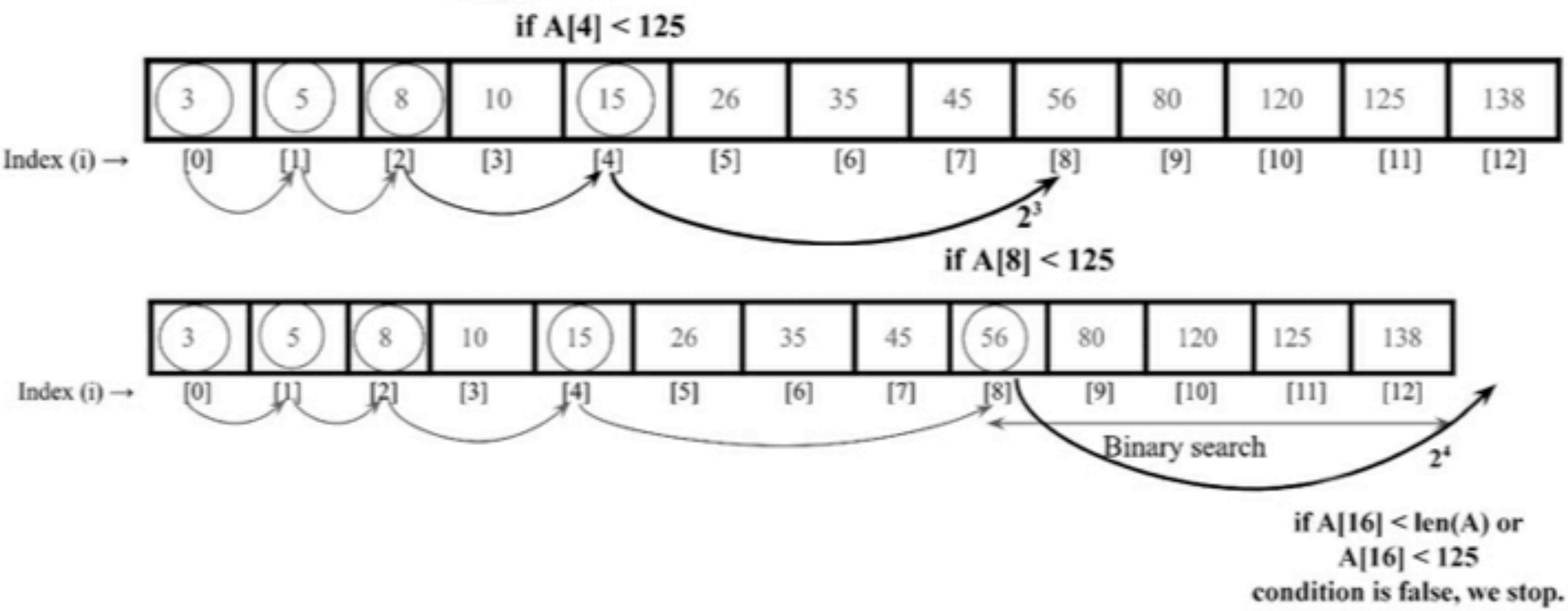
2^1

if A[2] < 125

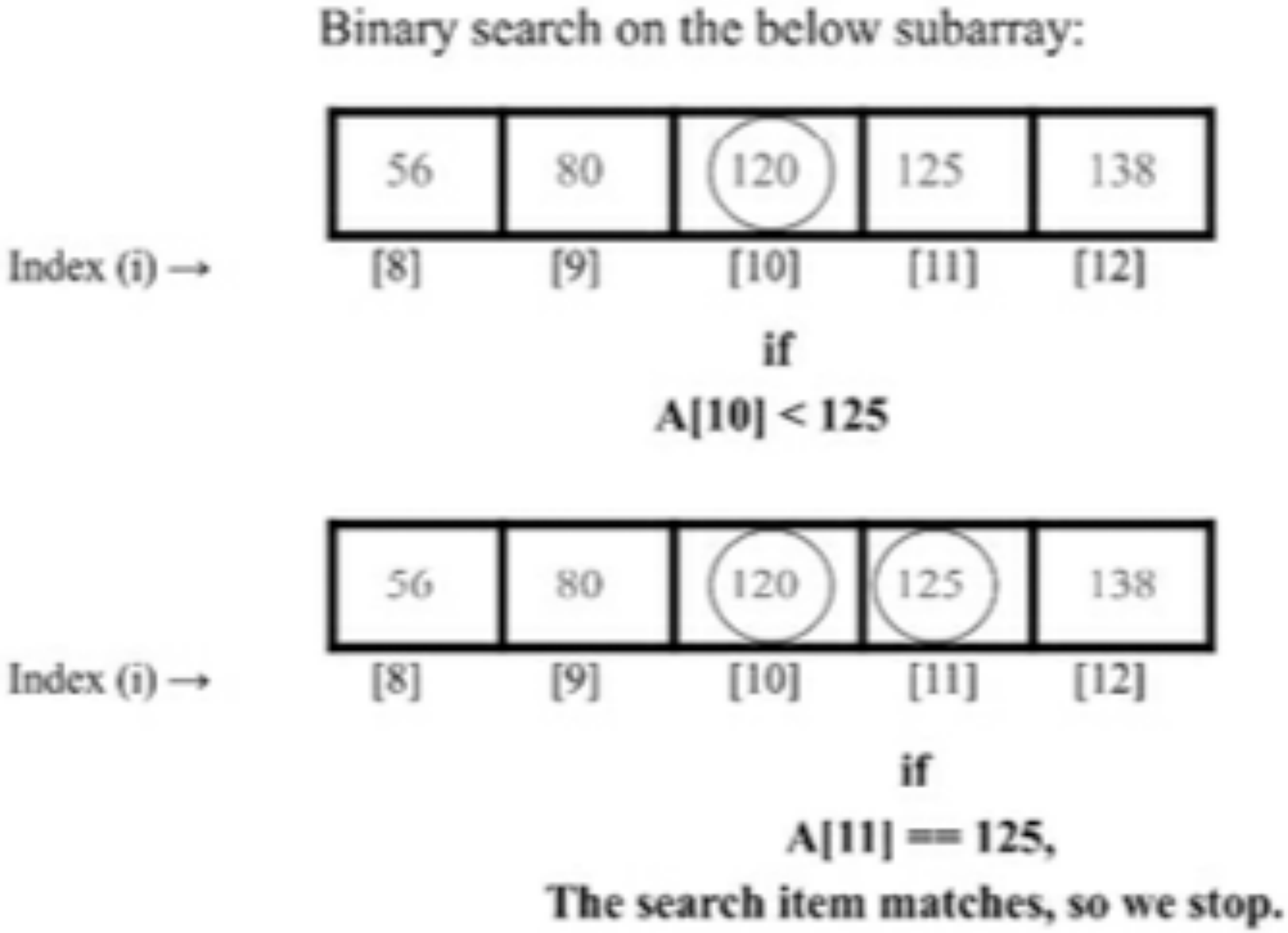
	3	5	8	10	15	26	35	45	56	80	120	125	138
Index (i) →	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]

2^2

Exponential search



Exponential search



Binary search on the below subarray:

	56	80	120	125	138
Index (i) →	[8]	[9]	[10]	[11]	[12]

if

A[10] < 125

	56	80	120	125	138
Index (i) →	[8]	[9]	[10]	[11]	[12]

if

A[11] == 125,

The search item matches, so we stop.

Choosing a search algorithm

- Linear search
  - Best for short, unsorted lists
  - Slow for large lists -- O(***n***)
- Binary search
  - Best for sorted lists that are not very big
- Interpolation sort
  - Good if data are uniformly distributed
- Exponential and Jump search
  - Best for very large lists

Ch 10