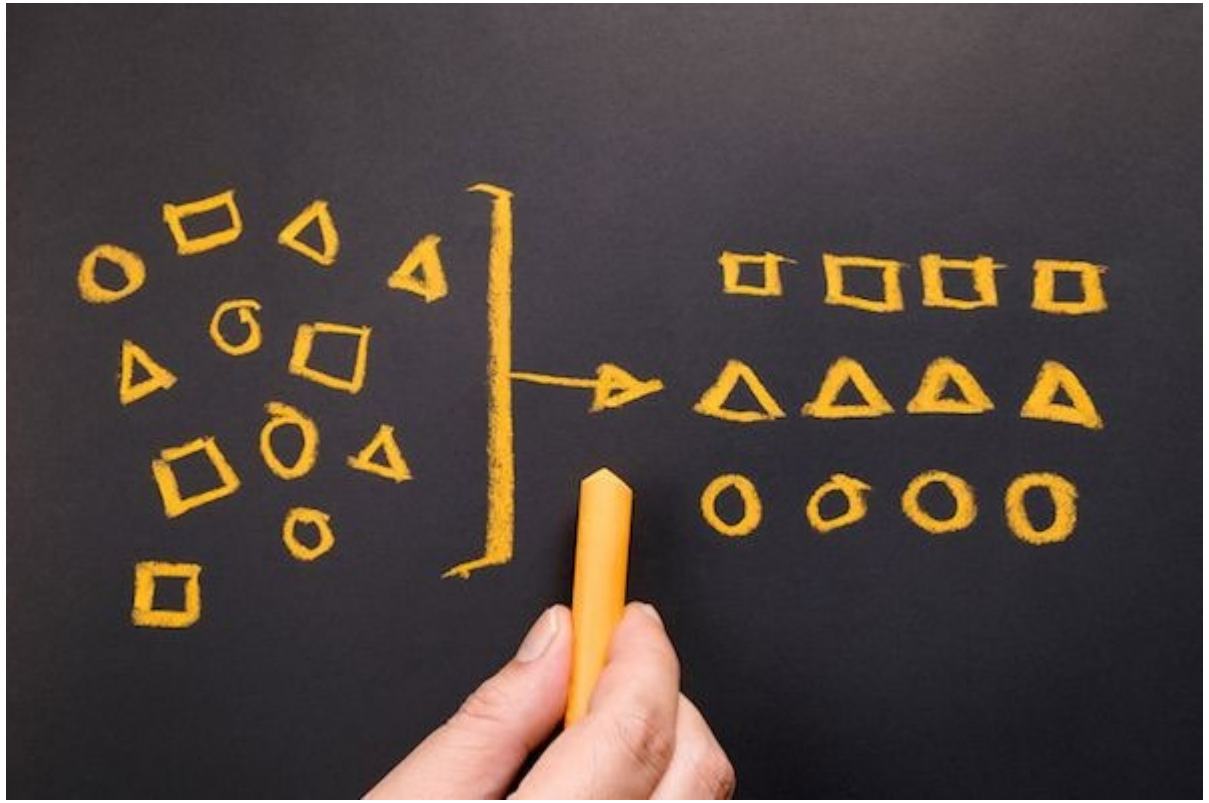
Chapter 5

Searching and Sorting



Contents

- Searching
  - The Sequential Search
  - The Binary Search
- Sorting
  - The Bubble Sort
  - The Selection Sort

Searching

Searching

- **Searching** is the algorithmic process of finding a particular item in a collection of items.
- A search typically answers either **True** or **False** as to whether the item is present.
- It may be modified to return where the item is found.

```
num = [1, 2, 32, 8, 17, 19]
print(32 in num)
print(5 in num)
```

The Sequential Search

The Sequential Search

- When data items are stored in a collection such as a **list** that they have **a linear or sequential relationship**
- Each data item is stored in a **position** relative to the others.

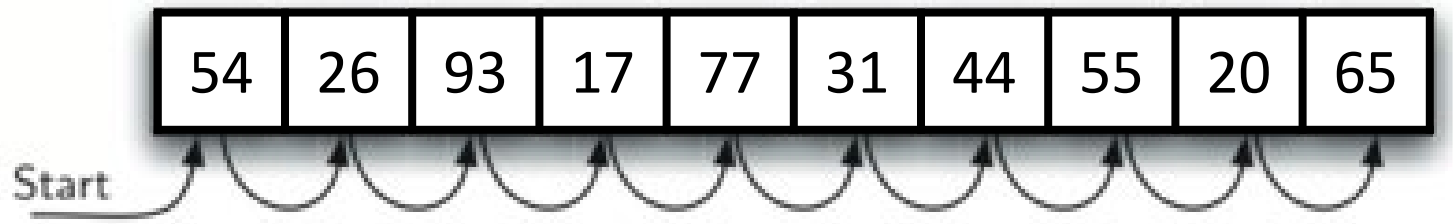
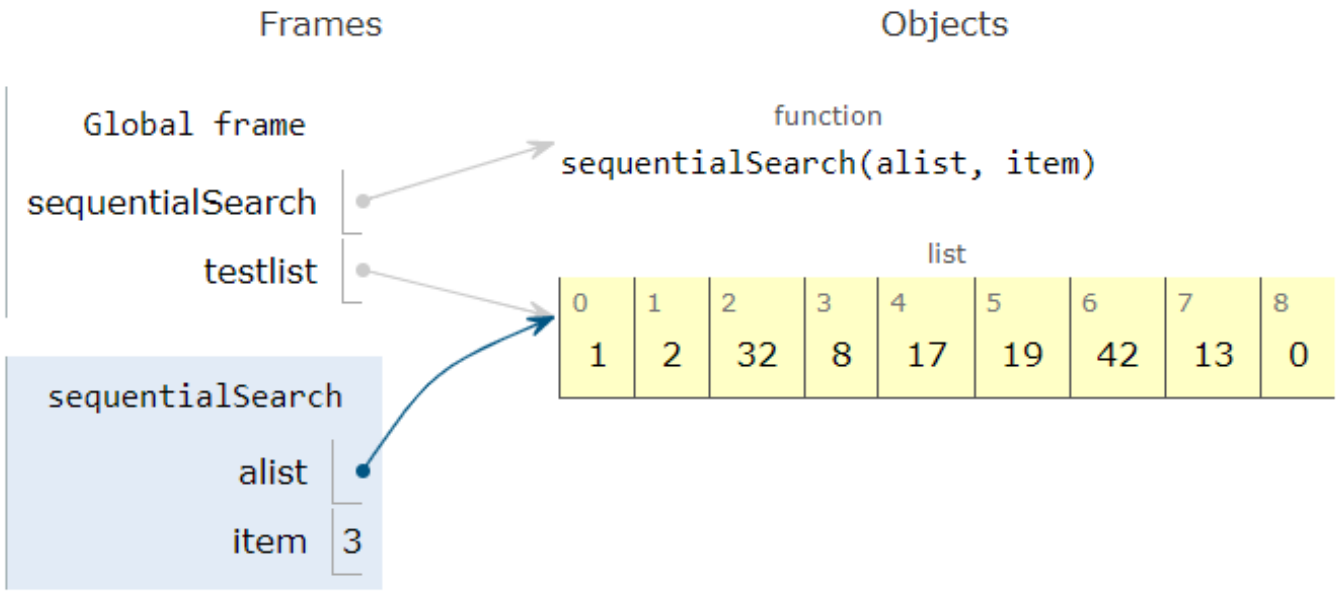


Figure 1: Sequential Search of a List of Integers

- The complexity of the sequential search is O(n)

The Sequential Search

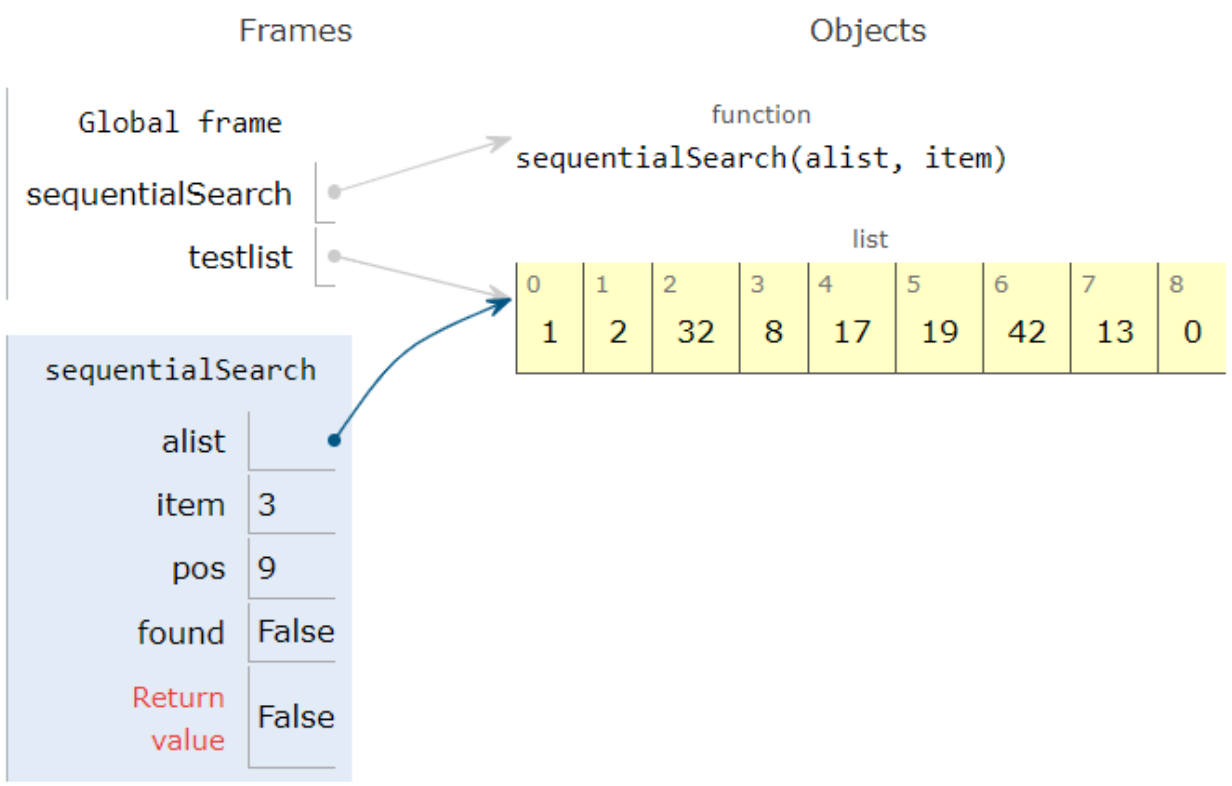
```
1  def sequentialSearch(alist, item):
2      pos = 0
3      found = False
4
5      while pos < len(alist) and not found:
6          if alist[pos] == item:
7              found = True
8          else:
9              pos = pos+1
10
11     return found
12
13 testlist = [1, 2, 32, 8, 17, 19, 42, 13, 0]
14 print(sequentialSearch(testlist, 3))
15 print(sequentialSearch(testlist, 13))
```

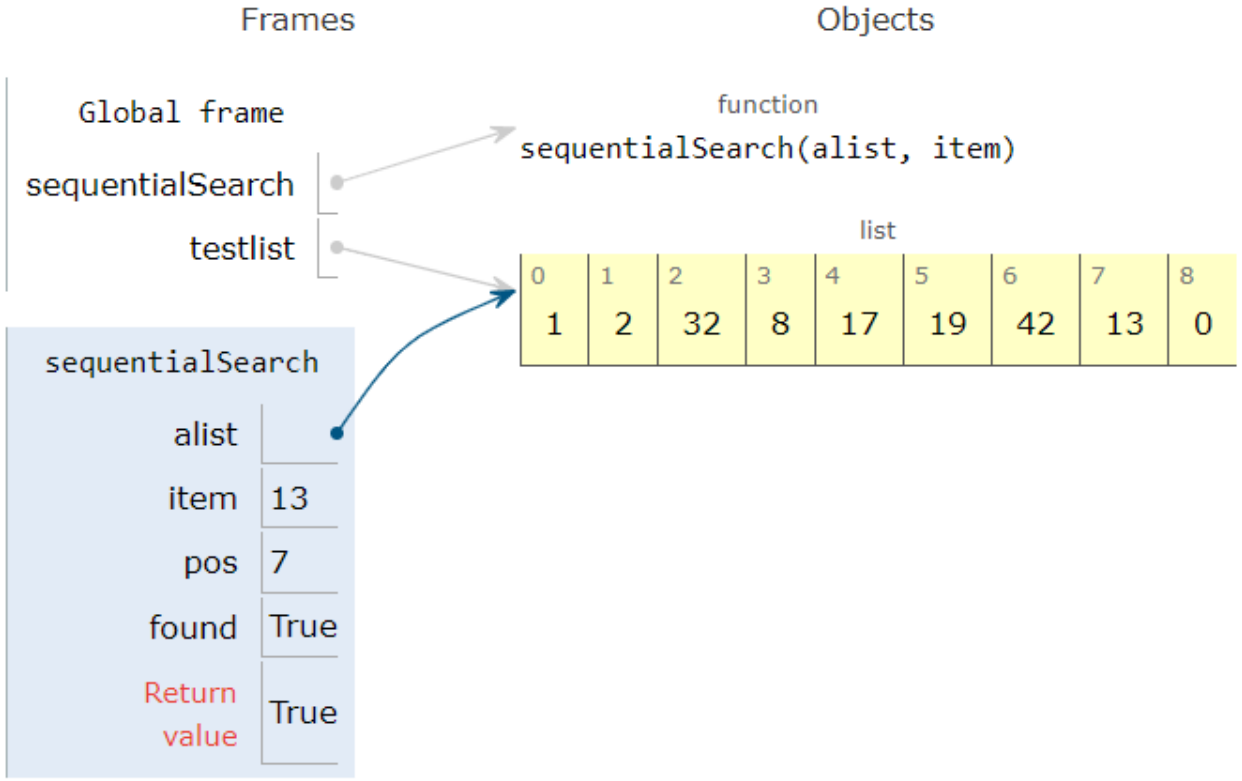


line that just executed

next line to execute

The Sequential Search

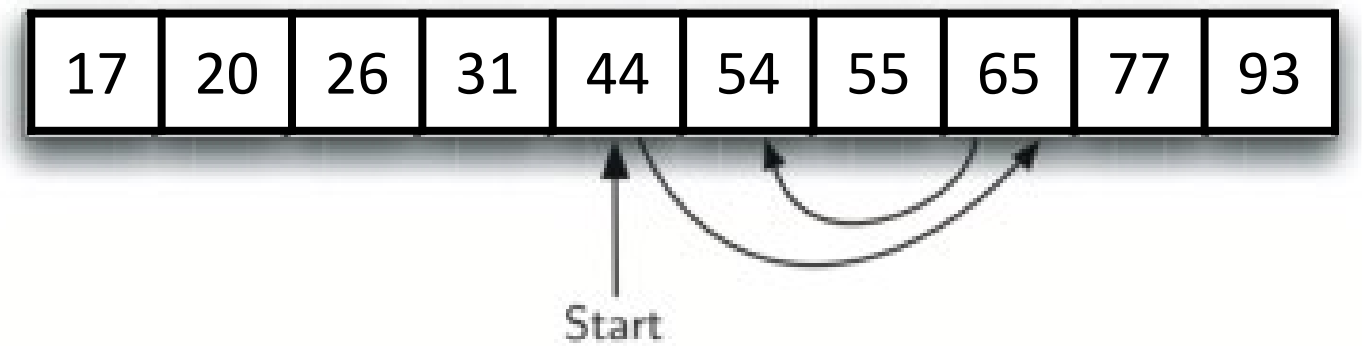




The Binary Search

The Binary Search

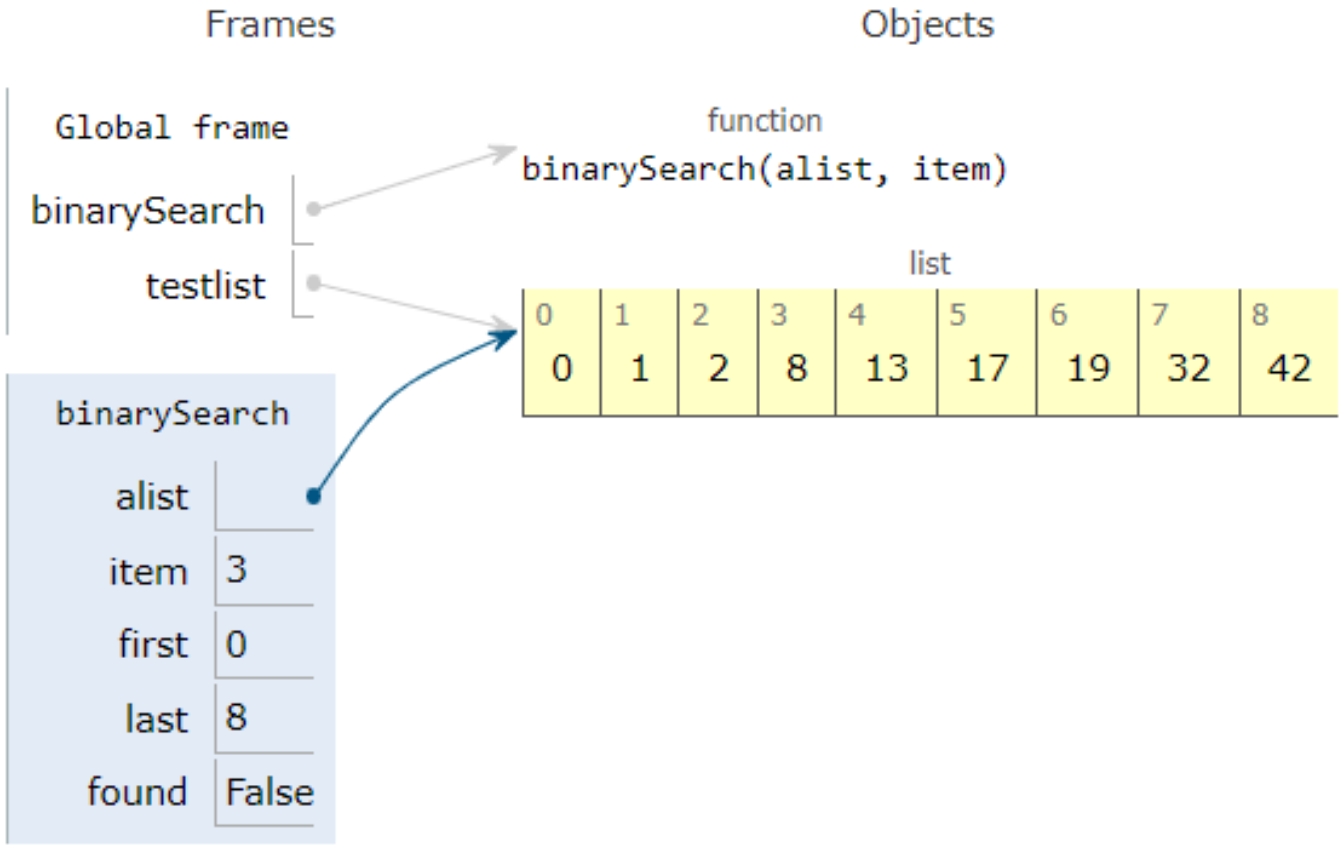
- A **binary search** will **start** by examining the **middle** item.
- If that item is the one we are searching for, we are done.
- If it is not the correct item, we can use the ordered nature of the list to eliminate half of the remaining items.



- The complexity of the sequential search is O(log n)

The Binary Search

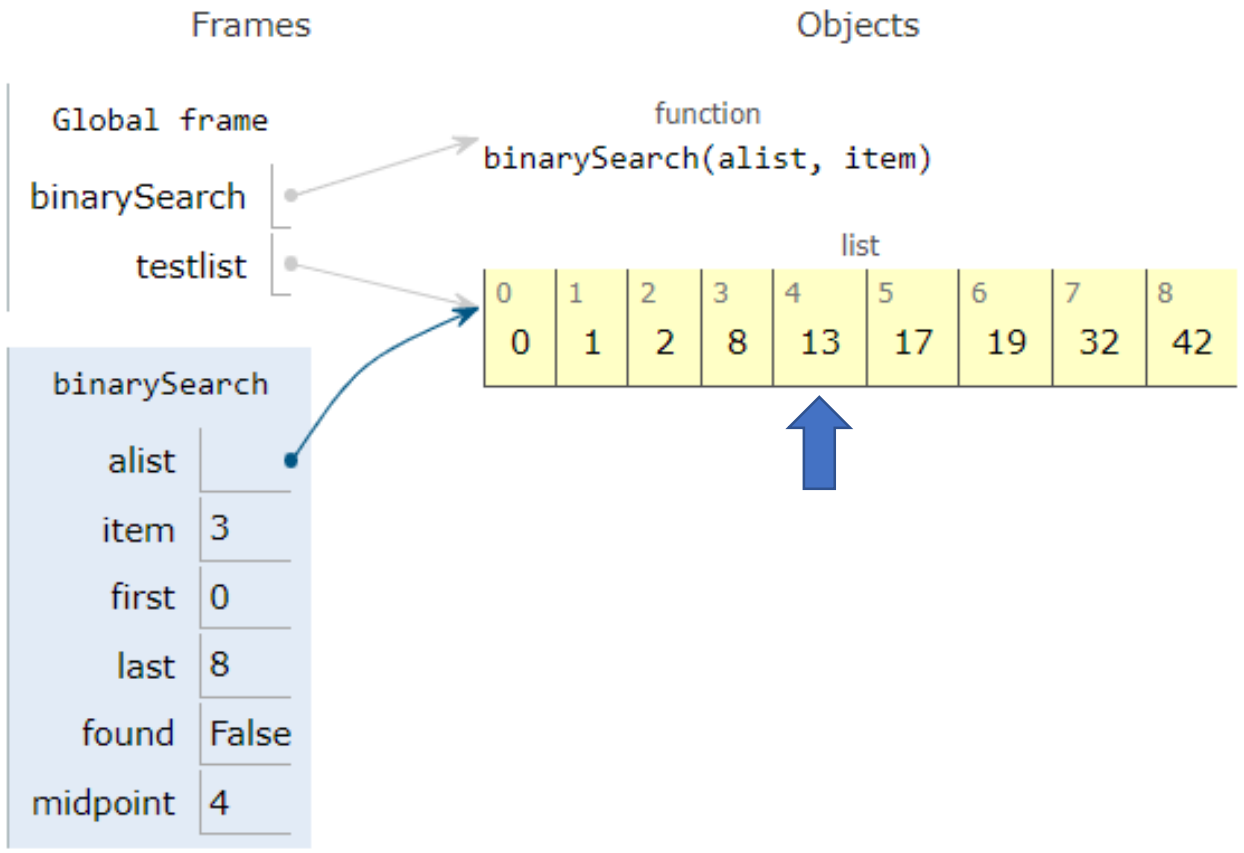
```
1  def binarySearch(alist, item):
2      first = 0
3      last = len(alist)-1
4      found = False
5
6      while first<=last and not found:
7          midpoint = (first + last)//2
8          if alist[midpoint] == item:
9              found = True
10         else:
11             if item < alist[midpoint]:
12                 last = midpoint-1
13             else:
14                 first = midpoint+1
15
16     return found
17
18 testlist = [0, 1, 2, 8, 13, 17, 19, 32, 42,]
19 print(binarySearch(testlist, 3))
20 print(binarySearch(testlist, 13))
```

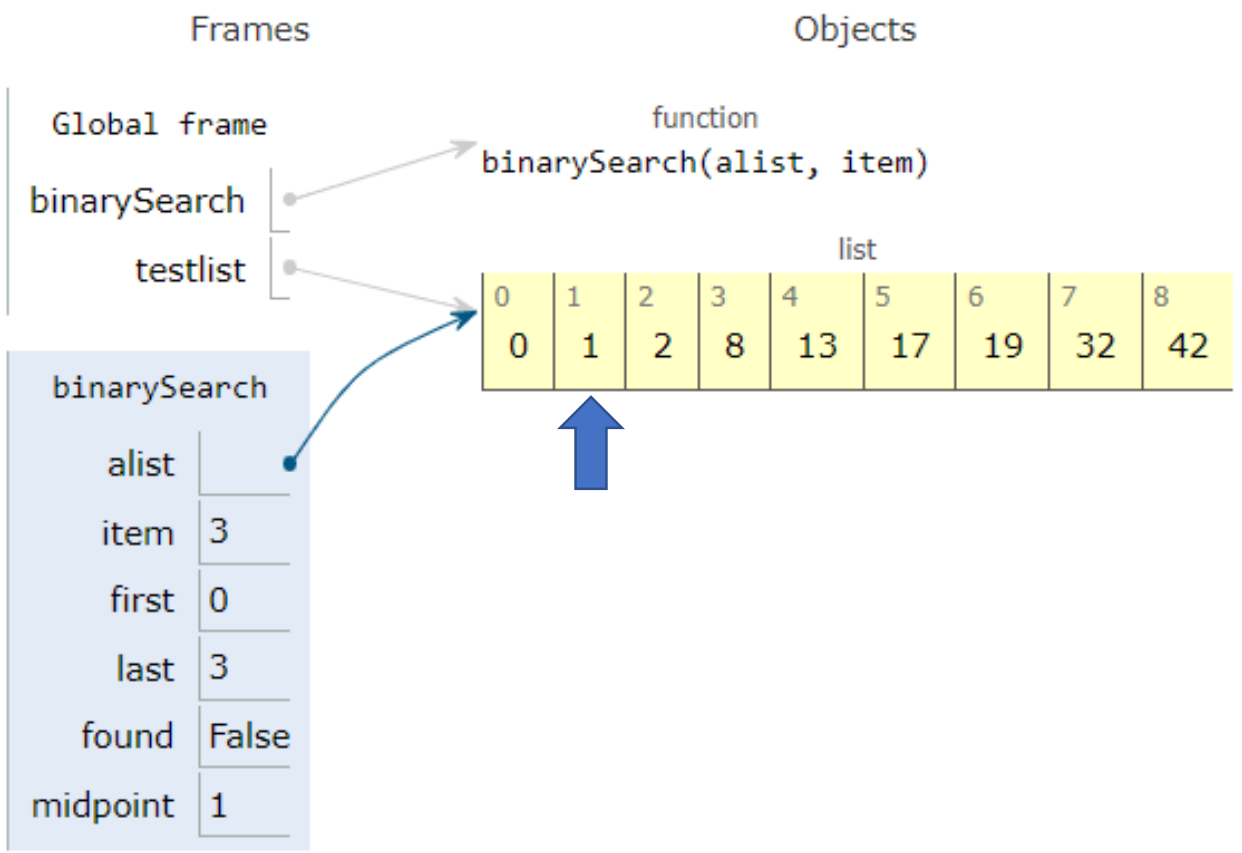


line that just executed

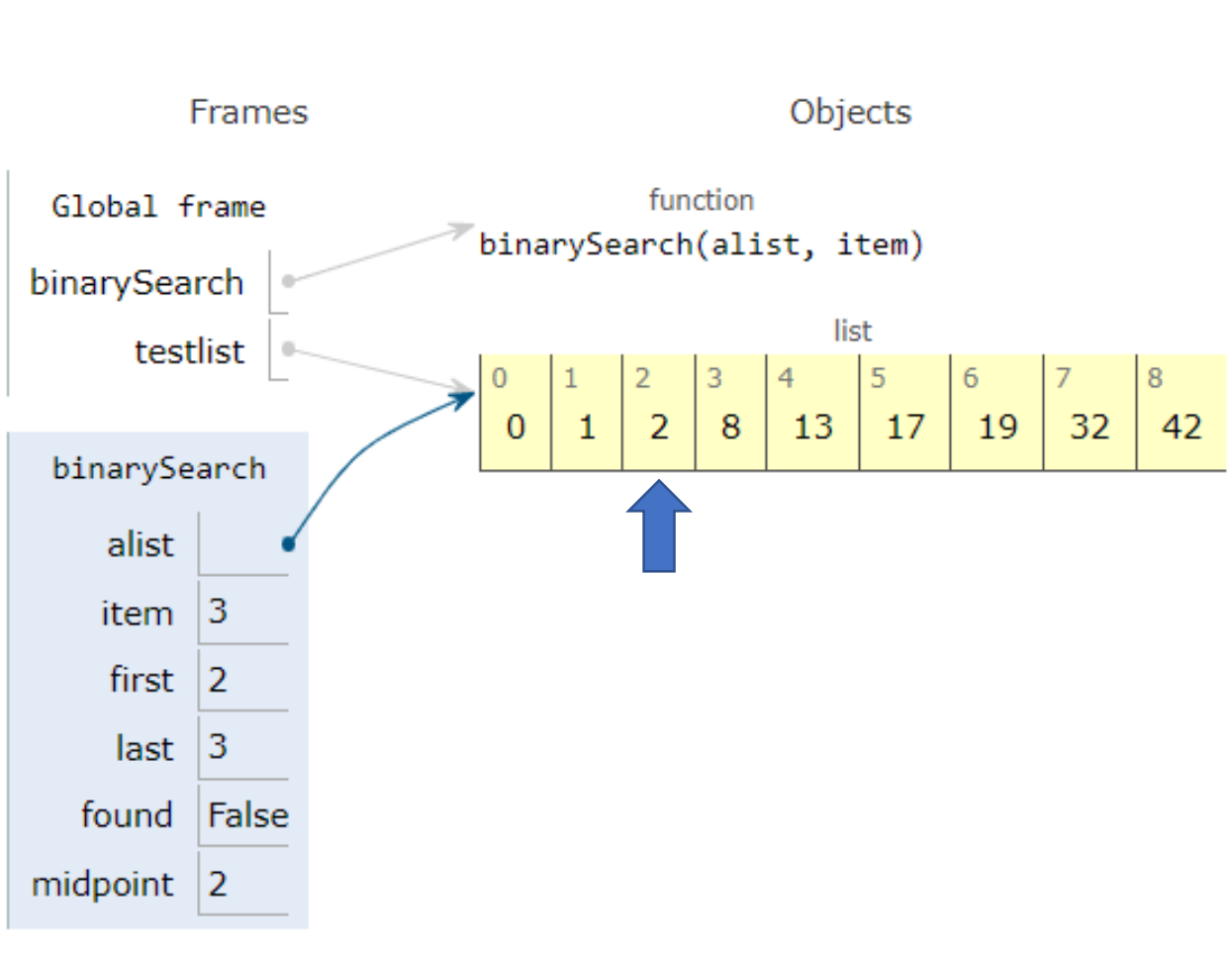
next line to execute

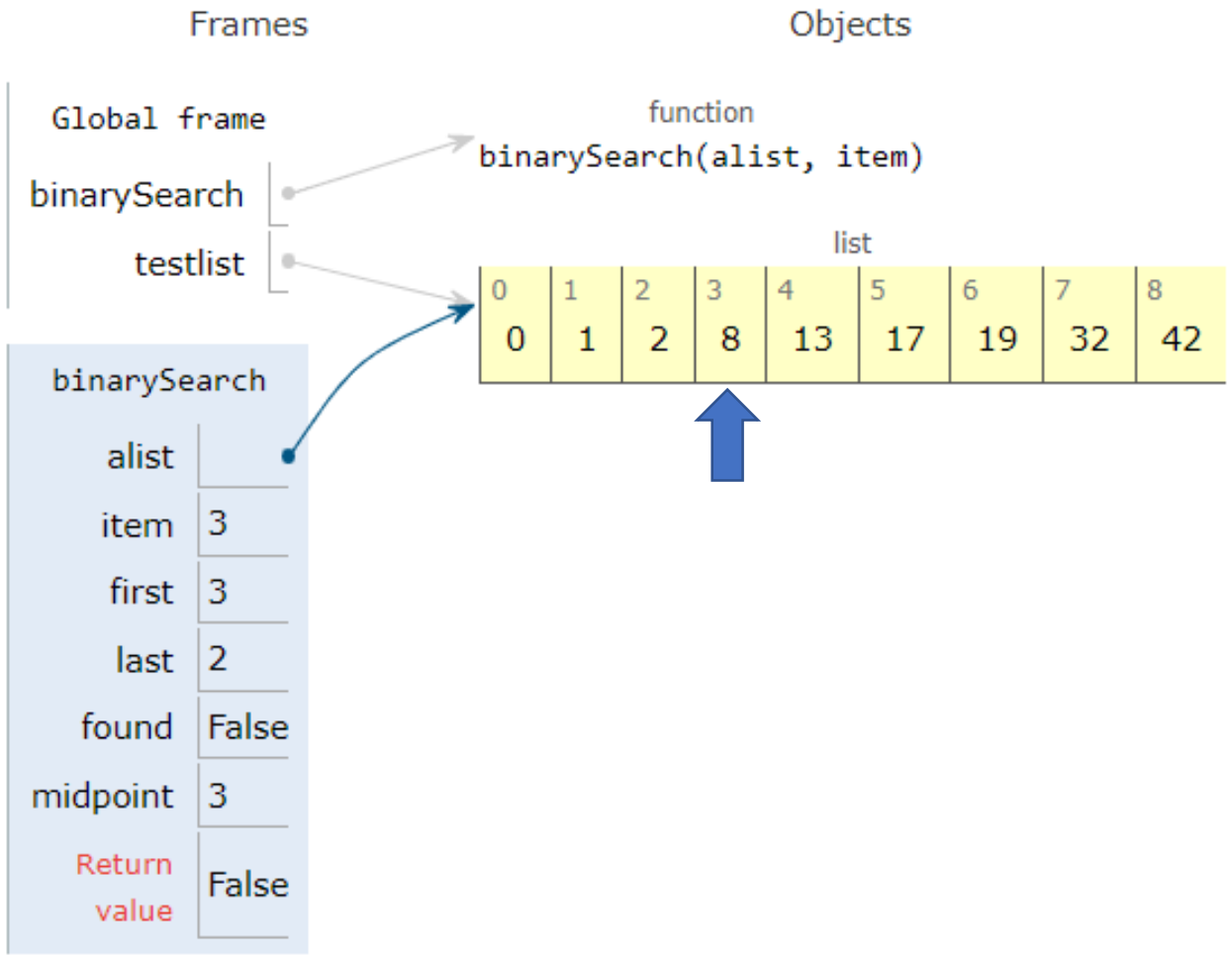
The Binary Search



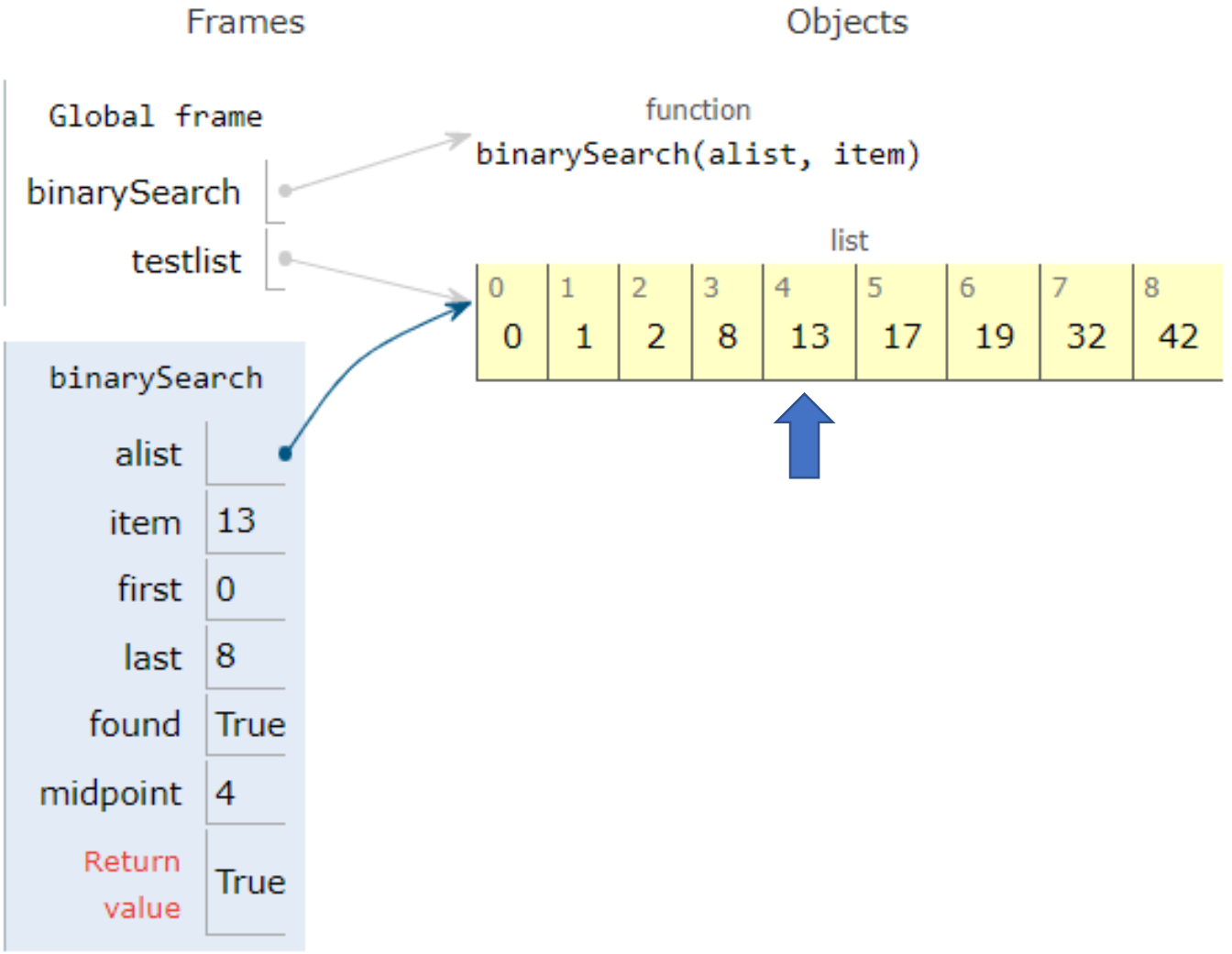


The Binary Search





The Binary Search




```
print(binarySearch(testlist,13))
```

Sorting

Sorting

- **Sorting** is the process of placing elements from a collection in some kind of order.
- For example,
  - A list of **words** could be sorted alphabetically or by length.
  - A list of **cities** could be sorted by population, by area, or by zip code.
- We have already seen a number of algorithms that were able to benefit from having a sorted list.

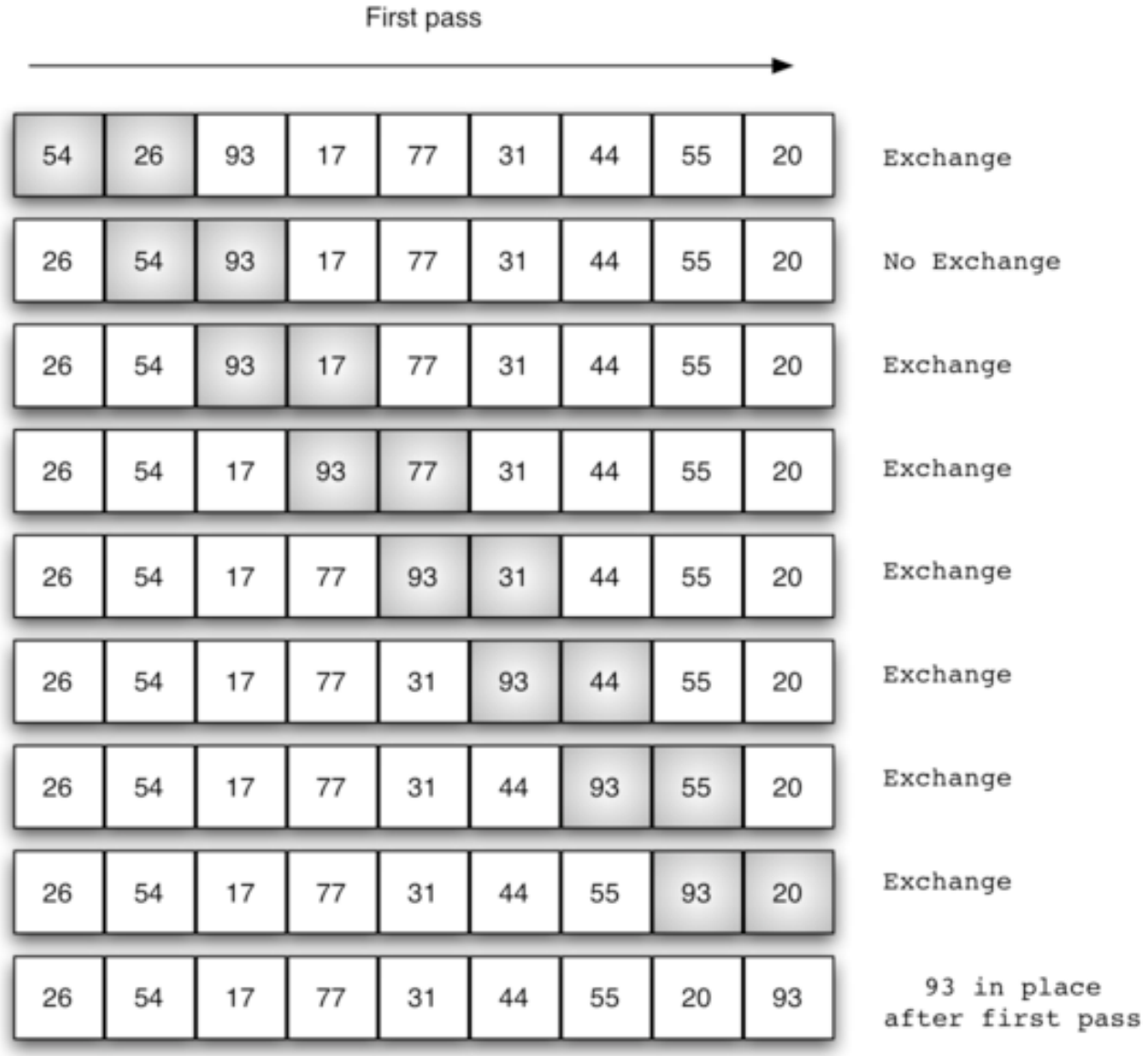
Sorting

- Many sorting algorithms that have been developed and analyzed.
- Sorting is an important area of study in computer science.
- Sorting a large number of items can take a substantial amount of computing resources.
- The efficiency of a sorting algorithm is related to the number of items being processed.
  - For small collections, a complex sorting method may be more trouble than it is worth. The overhead may be too high.
  - For larger collections, we want to take advantage of as many improvements as possible.

The Bubble Sort

The Bubble Sort

- The **bubble sort** makes multiple passes through a list.
- It compares adjacent items and exchanges those that are out of order.
- Each pass through the list places the next largest value in its proper place.



The Bubble Sort

- The exchange operation, sometimes called a "**swap**"
- Typically, swapping two elements in a list requires a temporary storage location
- A code fragment such as:

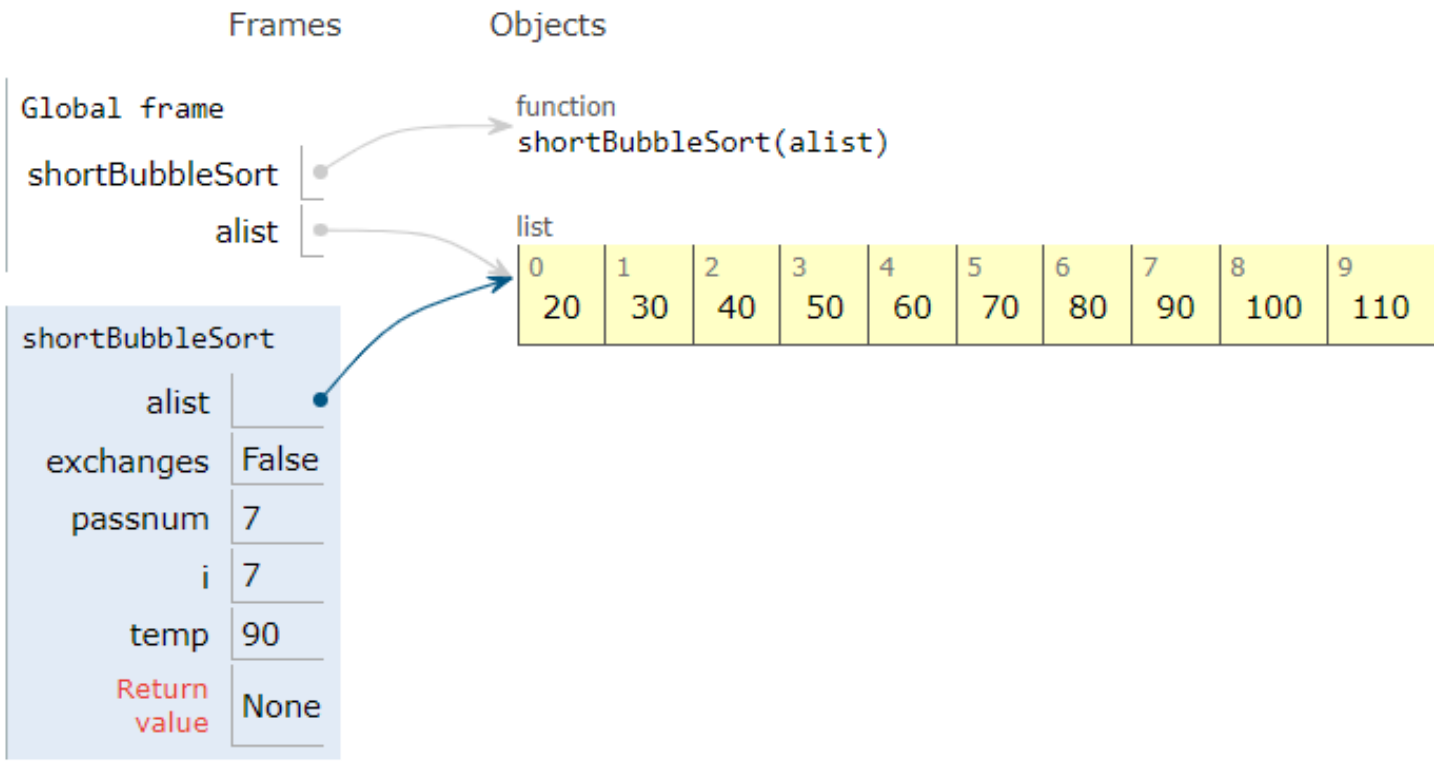
```
temp = alist[i]
alist[i] = alist[j]
alist[j] = temp
```

- This is $O(n^2)$ comparisons.

```
def bubbleSort(alist):
    for passnum in range(len(alist)-1,0,-1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp

alist = [54,26,93,17,77,31,44,55,20]
bubbleSort(alist)
print(alist)
```

The Bubble Sort

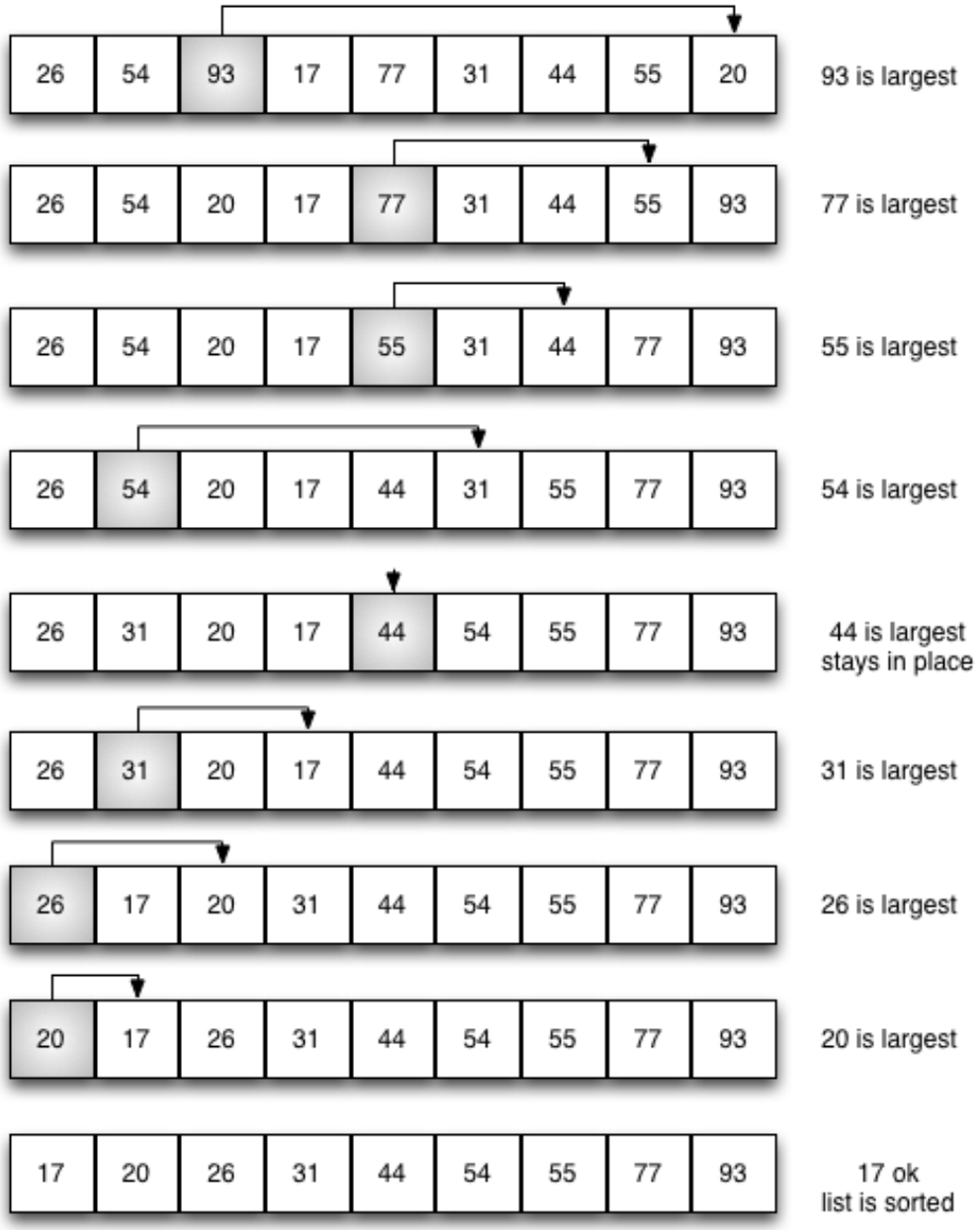
```
def shortBubbleSort(alist):
    exchanges = True
    passnum = len(alist)-1
    while passnum > 0 and exchanges:
        exchanges = False
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                exchanges = True
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp
        passnum = passnum-1

alist=[20,30,40,90,50,60,70,80,100,110]
shortBubbleSort(alist)
print(alist)
```



The Selection Sort

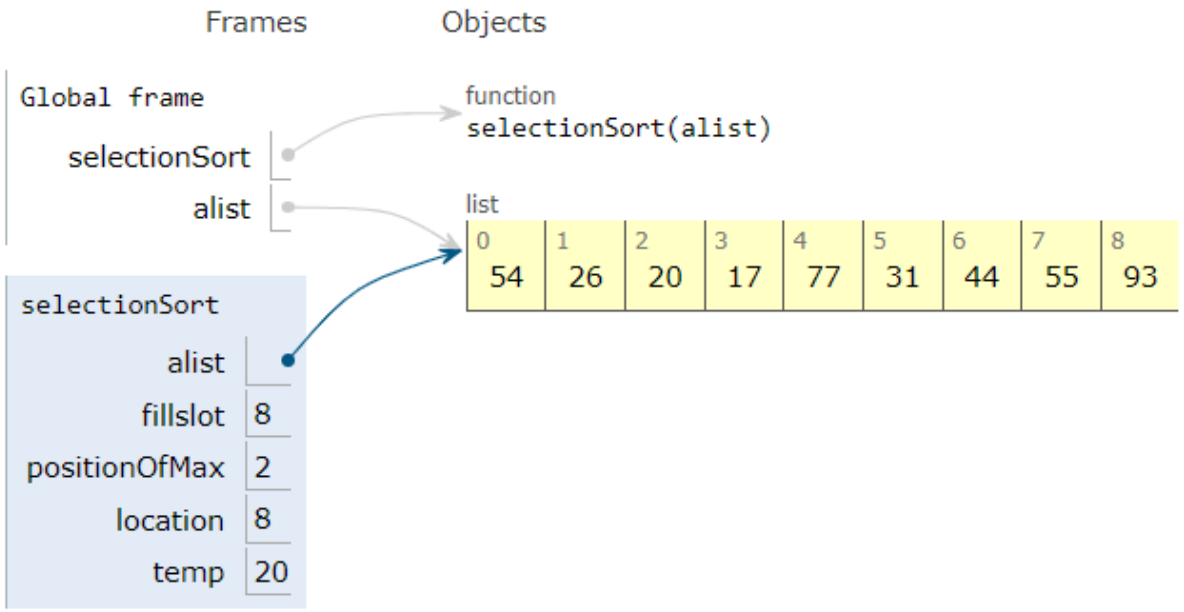
The Selection Sort

- The **selection sort** improves on the bubble sort by making only one exchange for every pass through the list.
- A selection sort looks for the largest value as it makes a pass and, after completing the pass, places it in the proper location.



The Selection Sort

```
def selectionSort(alist):
    for fillslot in range(len(alist)-1,0,-1):
        positionOfMax=0
        for location in range(1,fillslot+1):
            if alist[location]>alist[positionOfMax]:
                positionOfMax = location

        temp = alist[fillslot]
        alist[fillslot] = alist[positionOfMax]
        alist[positionOfMax] = temp

alist = [54,26,93,17,77,31,44,55,20]
selectionSort(alist)
print(alist)
```



- This is still $O(n^2)$ comparisons.

Big O Summary

Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity	Worst Space Complexity
Linear Search	O(1)	O(n)	O(n)	O(1)
Binary Search	O(1)	O(log n)	O(log n)	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(log n)
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)
Tim Sort	O(n)	O(nlogn)	O(nlogn)	O(n)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)

Work5-1: จับกลุ่ม ทำ Presentation นำเสนอเรื่องต่างๆ ดังนี้

	Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity	Worst Space Complexity
	Linear Search	O(1)	O(n)	O(n)	O(1)
	Binary Search	O(1)	O(log n)	O(log n)	O(1)
	Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
	Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
1	Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
2	Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
3	Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(log n)
4	Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
5	Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
6	Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)
7	Tim Sort	O(n)	O(nlogn)	O(nlogn)	O(n)
8	Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)

กลุ่มละไม่เกิน 3 คน

นำเสนอไม่เกิน 10 นาที

Work5-2 งานเดี่ยว

- จงเขียนโปรแกรมเพื่อเก็บคะแนนวิชา Problem Solving โดยโปรแกรมมีความสามารถดังนี้
  - สามารถระบุจำนวนนักศึกษาได้
  - สามารถกรอกชื่อและคะแนนของนักศึกษาแต่ละคนได้
  - สามารถแสดงคะแนนที่ sort แล้วโดยแสดงพร้อมชื่อได้
    - แสดงคะแนนทั้งหมดจากมากไปน้อย
    - แสดงชื่อและคะแนนของ Top 3 ของคนที่ได้คะแนนเยอะสุด
    - แสดงชื่อและคะแนนของ Top 3 ของคนที่ได้คะแนนน้อยสุด
  - สามารถค้นหาคะแนนได้
    - ถ้าค้นหาเจอให้บอกว่ามีจำนวนเท่าใด เช่น ให้ค้นหาคะแนน 30 และมีคนที่ได้คะแนน 30 จำนวน 2 คน ก็ให้แสดงออกมาด้วยว่าเป็นใครบ้าง

```
Enter the number of students: 5
Enter student name: AAA
Enter student score: 78
Enter student name: BBB
Enter student score: 35.5
Enter student name: CCC
Enter student score: 63
Enter student name: DDD
Enter student score: 24
Enter student name: MMM
Enter student score: 89

--Unsorted Scores--
AAA: 78.0
BBB: 35.5
CCC: 63.0
DDD: 24.0
MMM: 89.0

--Sorted Scores (Bubble Sort)--
MMM: 89.0
AAA: 78.0
CCC: 63.0
BBB: 35.5
DDD: 24.0

--Top 3 Highest Scores--
MMM: 89.0
AAA: 78.0
CCC: 63.0

--Top 3 Lowest Scores--
DDD: 24.0
BBB: 35.5
CCC: 63.0

Enter the score to search: 39
No students found with score 39.0

Enter the score to search: 63
Found student with score 63.0
```