# 4.3 Arama

→ hesaplamada karşılaşılan yaygın problemlerdir: arama, sıralama.

→ dönütü: true veya false

→ Python: in bir çeşit aramadır

```
>>> 15 in [3,5,2,4,1]
False
>>> 5 in [3,5,2,4,1]
True
```

# 4.3.1 Ardışıl (sequential) arama

→ sıralanacak veri listede

→ liste: doğrusal - ardışıl ilişki

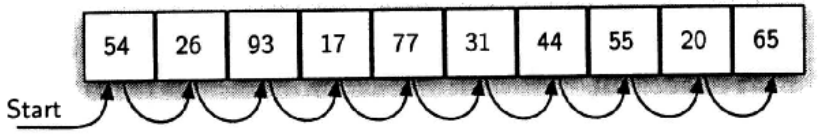


**Figure 4.2:** Sequential Search of a List of Integers

# 4.3.1 Ardışıl (sequential) arama

→ sıralanacak veri listede

→ liste: doğrusal – ardışıl ilişki

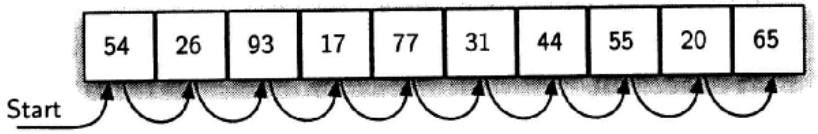


**Figure 4.2:** Sequential Search of a List of Integers

→ amaç,

```
>> 44 in [54, 26, 93, 17, 77, 31, 44, 5
True
>> 99 in [54, 26, 93, 17, 77, 31, 44, 5
False
```

# algoritma

→ i=0'dan başla, len(list)-1'e kadar tara - ara

```
1       def sequentialSearch(alist, item):
2           pos = 0
3           found = False
4           stop = False
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
```

# analiz

$\longrightarrow$ temel hesaplama birimi nedir?

# analiz

→ temel hesaplama birimi nedir?

→ problemi çözerken gerekli ortak adım

→ **karşılaştırma** (s6)

# senaryolar

→ üç farklı senaryo: en iyi, en kötü, ortalama

| | Best Case | Worst Case | Average Case |
|---|---|---|---|
| item is present | 1 | $n$ | $\frac{n}{2}$ |
| item is not present | $n$ | $n$ | $n$ |

**Table 4.2:** Comparisons Used in a Sequential Search of an Unordered List

→ eleman bulundu/bulunamadı

→ ortalama olarak: n/2 $\Longrightarrow$ O(n)

# sıralı listeler

→ ararken listenin sonuna kadar bakmaya gerek kalmaz

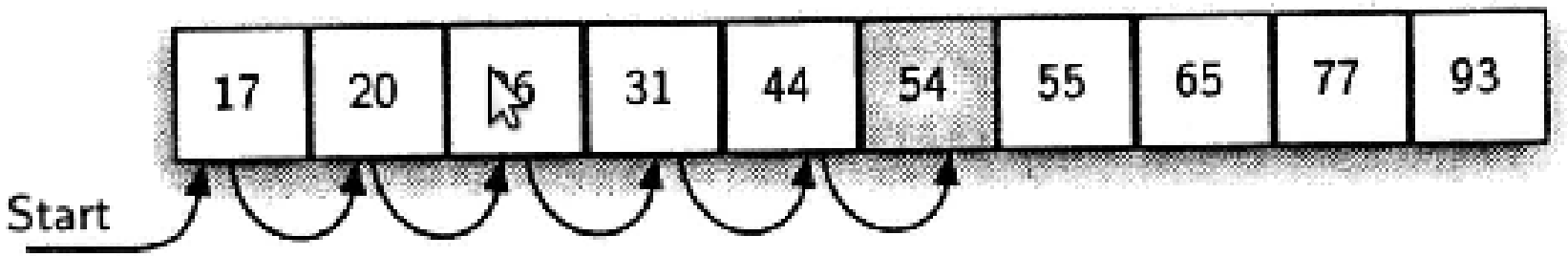


**Figure 4.3:** Sequential Search of an Ordered List of Integers

→ yerinde bulunamadığı ilk yerden çıkılır

```
>> 50 in [17, 20, 26, 31, 44, 54, 55, 65, 77, 93]
False
```

→ aranan 50 değeri 44'den sonra 54'den önce olmalıydı

→ devam etmeye gerek yok!

# algoritma

## → gerçekleme

```
1       def orderedSequentialSearch(alist, item):
2           pos = 0
3           found = False
4           stop = False
5           while pos < len(alist) and not found and not stop:
6               if alist[pos] == item:
7                   found = True
8               else:
9                   if alist[pos] > item:
10                      stop = True
11                  else:
12                      pos = pos+1
13
14          return found
```

# senaryolar

| | Best Case | Worst Case | Average Case |
|---|---|---|---|
| item is present | 1 | $n$ | $\frac{n}{2}$ |
| item is not present | 1 | $n$ | $\frac{n}{2}$ |

**Table 4.3:** Comparisons Used in Sequential Search of an Ordered List

→ Hala O(n)

→ sıralı olmasının üstünlüğünü göremedik

# 4.3.2 İkil Arama

→ sıralı listenin üstünlüğünden yararlanabilmek için

→ karşılaştırmanın daha zekice yapılması

→ sizce?

# 4.3.2 İkil Arama

→ sıralı listenin üstünlüğünden yararlanabilmek için

→ karşılaştırmanın daha zekice yapılması

→ sizce?

→ ardışıl olanda i=0'dan ... i=len-1'e gidiliyordu

→ ikil aramada ortadan başlanır

# algoritma

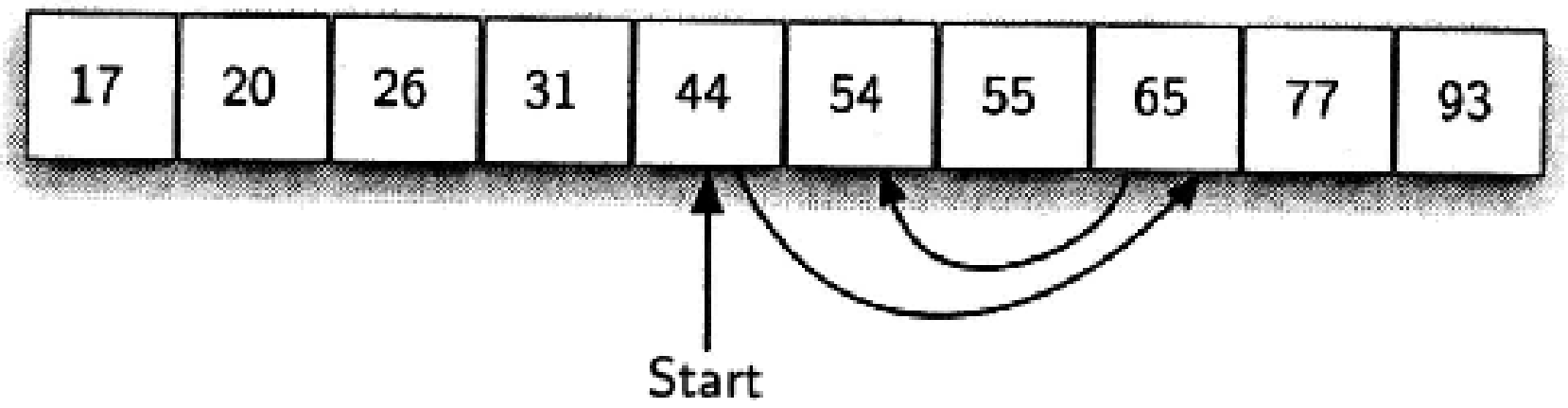


**Figure 4.4:** Binary search of an ordered list of integers

→ ya soldadır ya da sağdadır

# gerçekleme - v1

→ ya soldadır ya da sağdadır

→ seçilen tarafın ortasını belirle

→ yine ya soldadır ya da sağdadır

```
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
```

# gerçekleme - v2

→ ya soldadır ya da sağdadır

→ seçilen tarafta özyineli olarak işleme devam et

```
1       def binarySearch(alist, item):
2           if len(alist) == 0:
3               return False
4           else:
5               midpoint = len(alist)/2
6               if alist[midpoint]==item:
7                   return True
8               else:
9                   if item<alist[midpoint]:
10                    return binarySearch(alist[:midpoint],item)
11                  else:
12                    return binarySearch(alist[midpoint+1:],item)
```

# analiz

| Comparisons | Approximate Number of Items Left |
|---|---|
| 1 | $\frac{n}{2}$ |
| 2 | $\frac{n}{4}$ |
| 3 | $\frac{n}{8}$ |
| ... | |
| i | $\frac{n}{2^i}$ |

**Table 4.4:** Tabular Analysis for a Binary Search

→ n elemanla başla

→ ilk karşılaştırmanın ardından n/2 eleman,

→ sonra n/4, ...

# analiz

| Comparisons | Approximate Number of Items Left |
|---|---|
| 1 | $\frac{n}{2}$ |
| 2 | $\frac{n}{4}$ |
| 3 | $\frac{n}{8}$ |
| ... | |
| i | $\frac{n}{2^i}$ |

**Table 4.4:** Tabular Analysis for a Binary Search

→ n elemanla başla

→ ilk karşılaştırmanın ardından n/2 eleman,

→ sonra n/4, ...

→ log n kez karşılaştırma $\Longrightarrow$ O(log n)

# özet

→ bir kere sırala, çok kez ikil ara (n küçükse)

→ n büyükse sıralama maliyeti büyüyeceğinden ardışıl ara