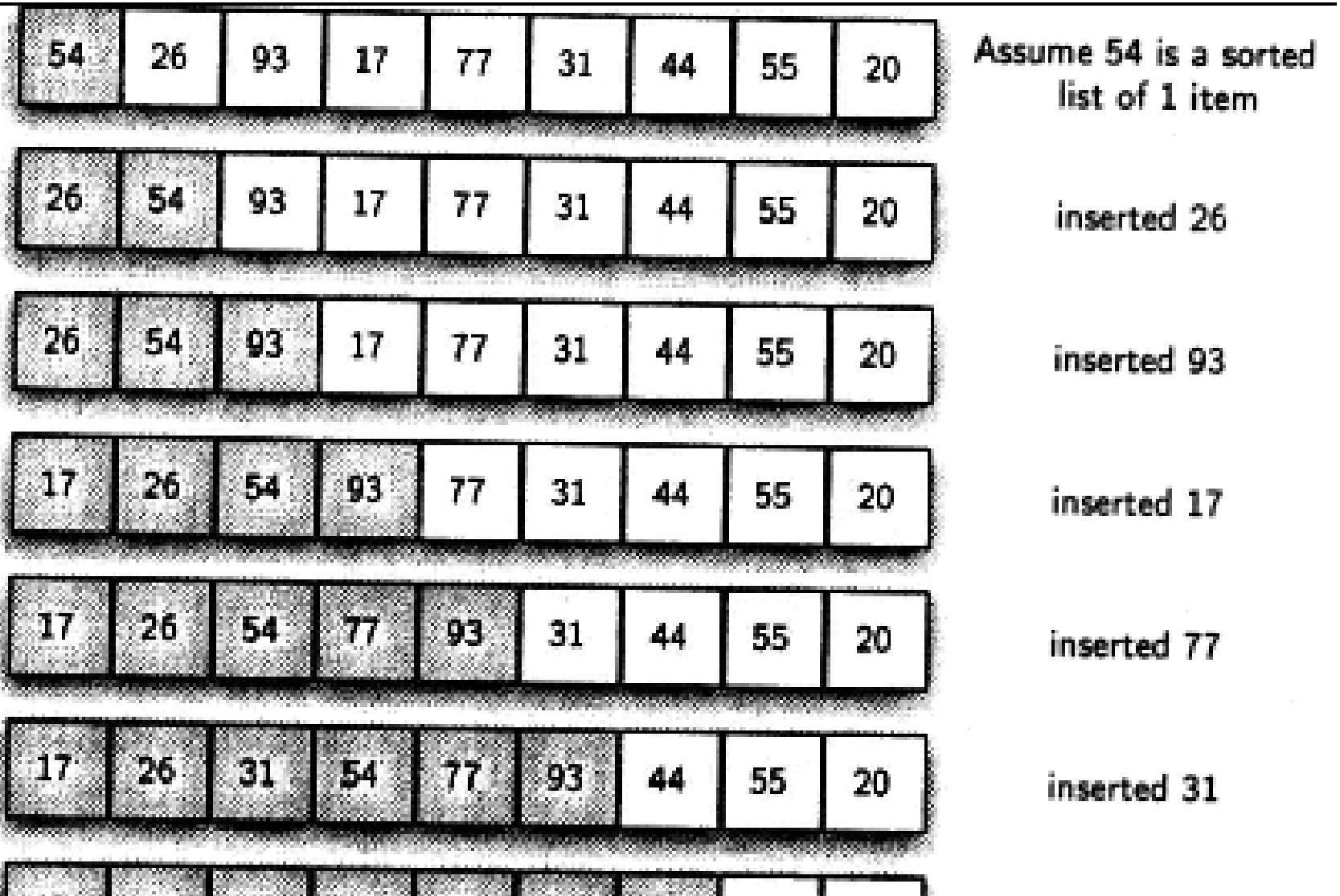
4.4.3 araya girmeli (insertion) sıralama

→ karmaşıklığı O(n^2)

→ listenin alt parçasını sürekli sıralı tut

→ yeni elemanı sıralı altlistede uygun yere koy (**insert**)

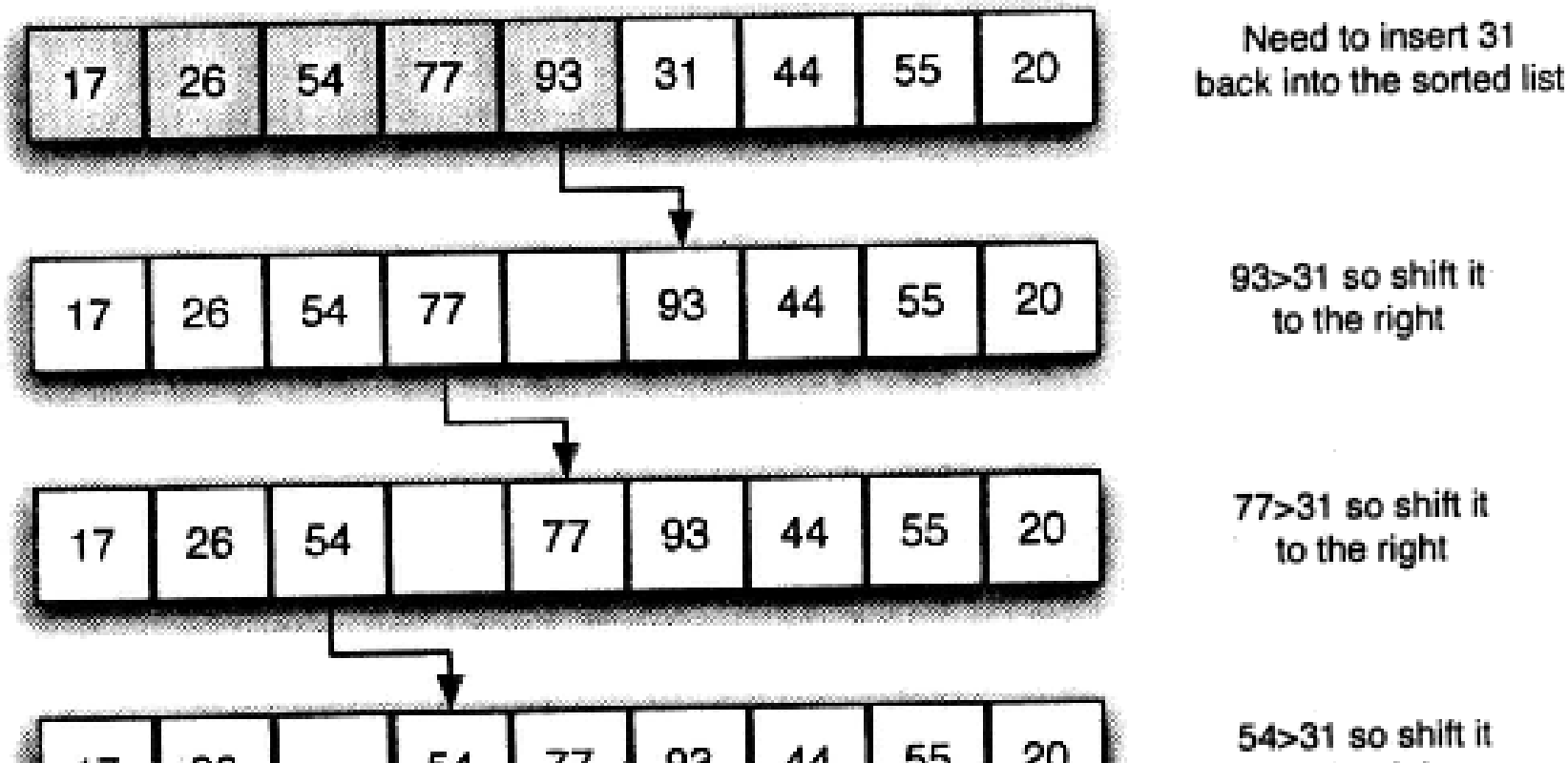


insertion sort

→ altliste bir büyüdü, ve hala sıralı

→ 5. geçişin ayrıntıları

1. şimdiki değeri (31) nereye koyacağız?
2. sıralı altlistedeki sağdan (en büyükten, 93) başlayarak
3. sola doğru ilerle (sırayla 93, 77, 54, 26 ve 17'yi ziyaret et)
4. ilk küçüğün ardına (sağına, 26) yerleştir



gerçekleme

→ gerçekleme

→ pozisyon 1'den, (n-1)'e doğru ilerle

```
1       def insertionSort(alist):
2           for index in range(1,len(alist)):
3
4               currentvalue = alist[index]
5               position = index
6
7               while position>0 and alist[position-1]>currentvalue:
8                   alist[position]=alist[position-1]
9                   position = position-1
10
11              alist[position]=currentvalue
```

analiz

en kötü durum

- → maksimum karşılaştırma: ilk (n-1) tamsayının toplamı
- → yani O(n^2)

en iyi durum

- → zaten sıralı olan liste
- → her geçişte yalnızca bir karşılaştırma
- → yani O(n)

takas mı X kaydırma mı?

→ kaydırma işlem süresi = (takas işlem süresi) / 3

→ benchmarkler bu yüzden daha iyidir

4.4.4 kabuk sıralama

→ araya eklemeli sıralamanın iyileştirilmişi

a) listeyi altlistelere böl

b) her birini insertion sort ile sırala

c) nihayi listelerin bir araya getirilmişini insertion sort ile sırala

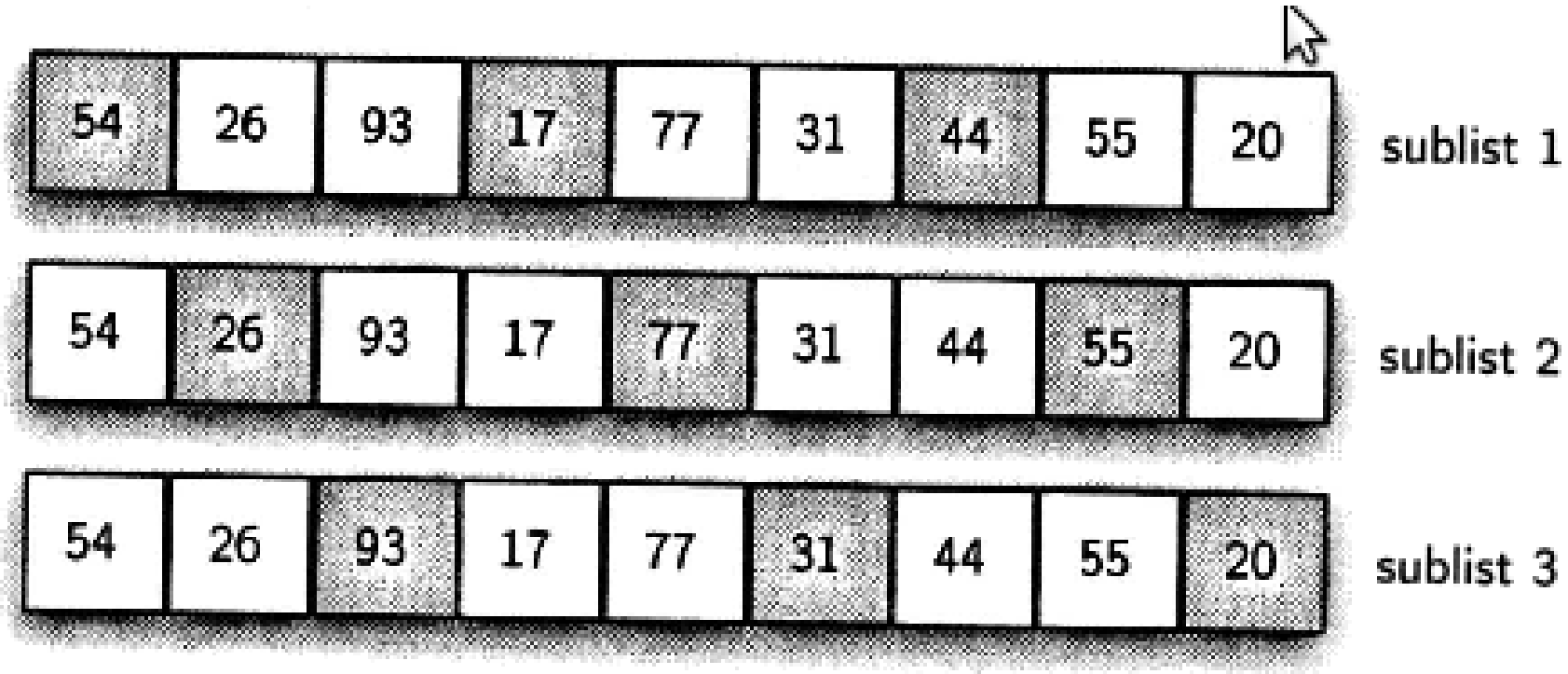
altlisteler

- → altlisteler nasıl oluşturulacak?
- → kişiye/tecrübesel
- → ardışıl seçmek yerine, atlayarak seçme
- → insertion sort nihayi listeyi sıralarken zaten komşulara bakacak
- → işini kolaylaştırmak için atlamayı kullan

altlisteler

→ atlamalı altliste seçme

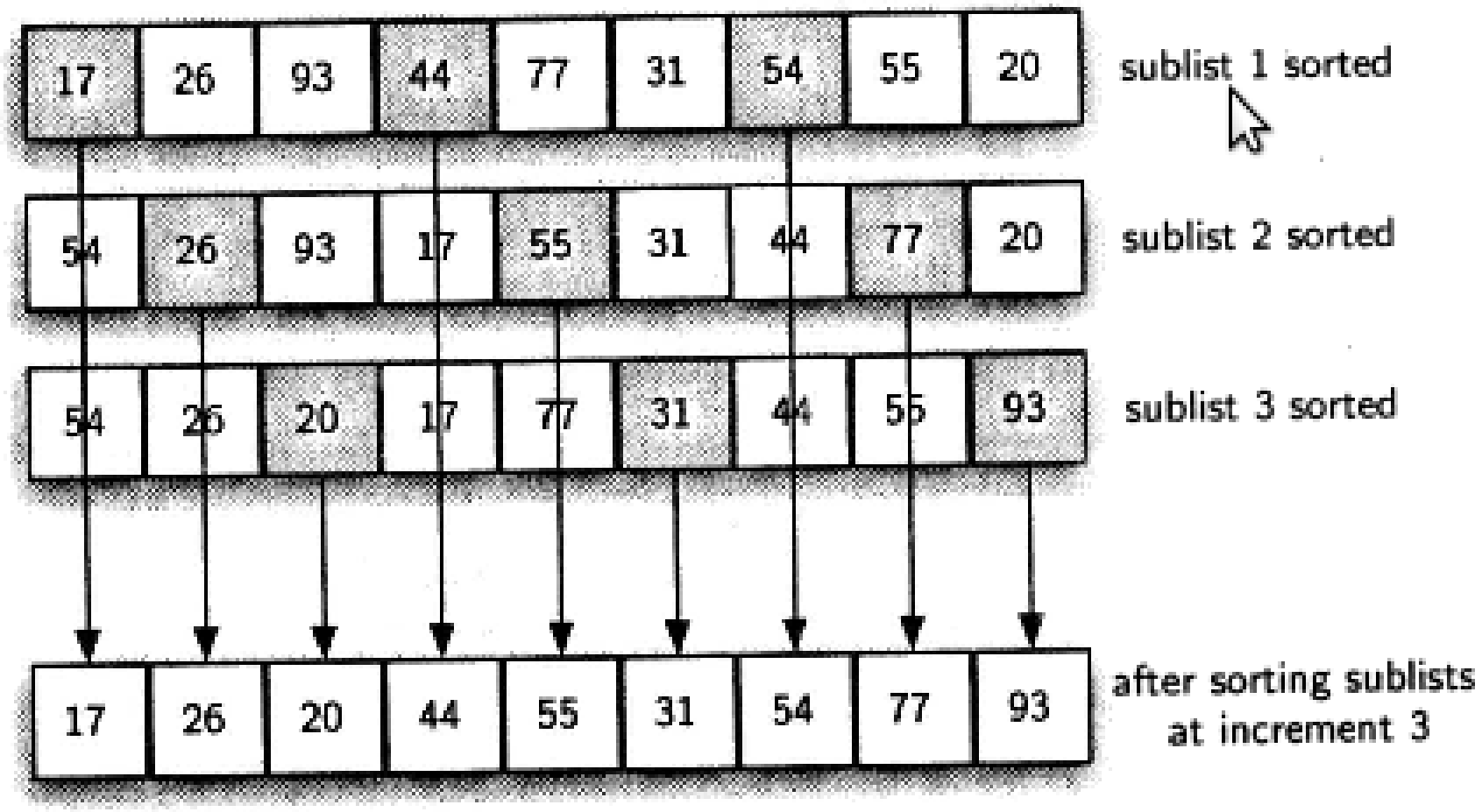
→ Python'cası nasıl?



altlisteler sıralanınca

→ altlisteler sırala

→ atlama = 3

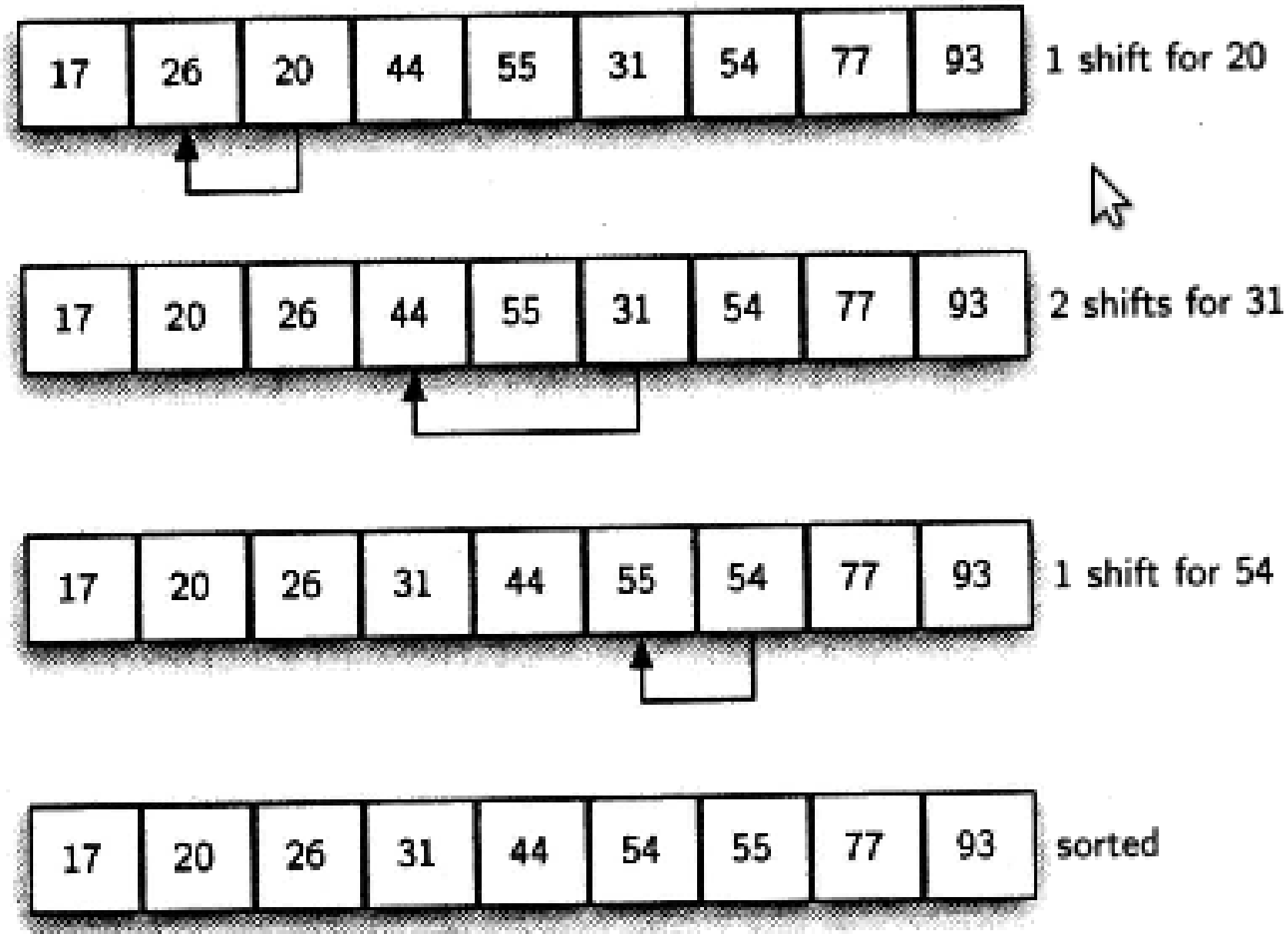


→ elemanlar hemen hemen doğru yerdeler

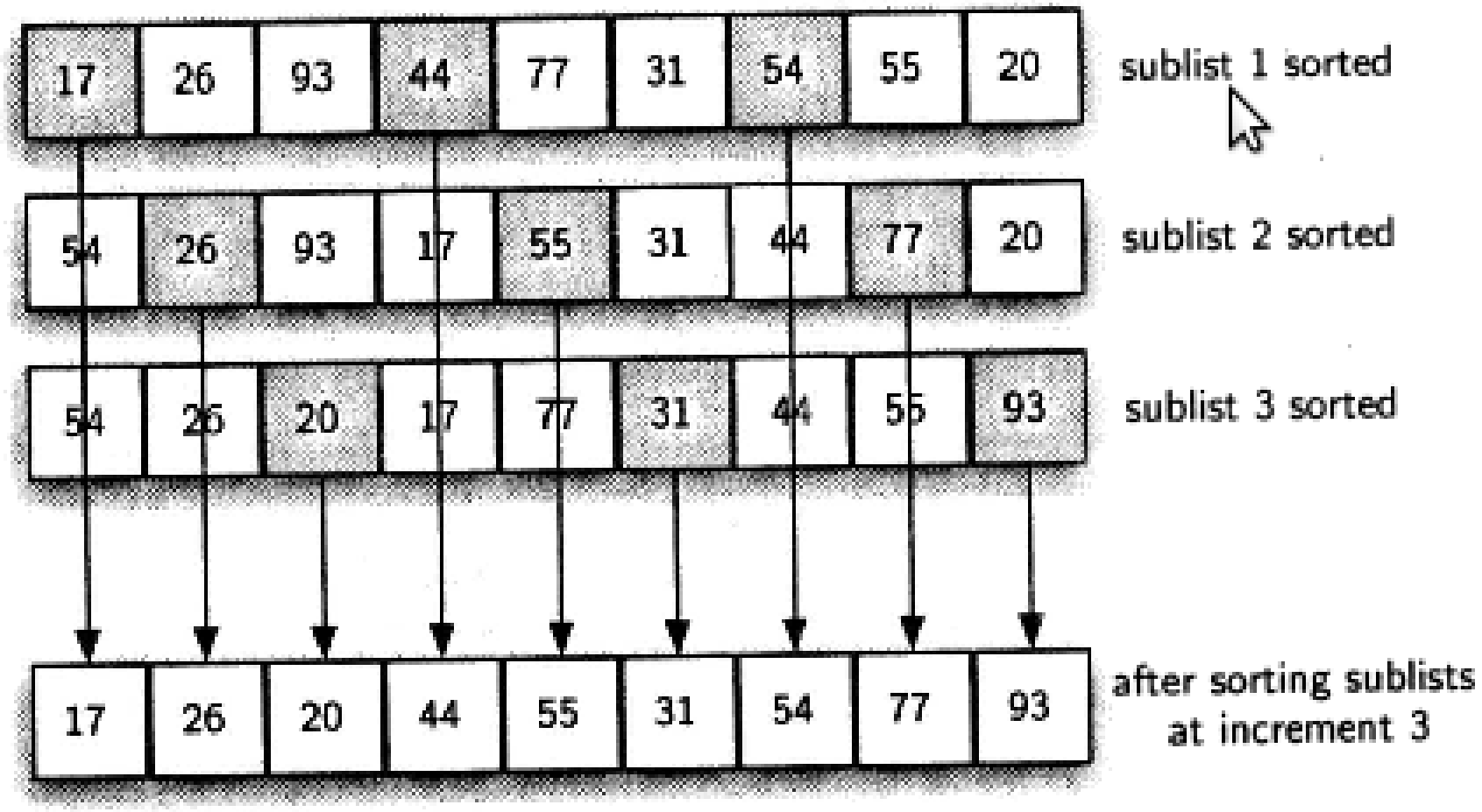
nihayi listeyi sırala

→ sıralı altlisteler oluşan nihayi listeyi sırala

→ atlama = 1



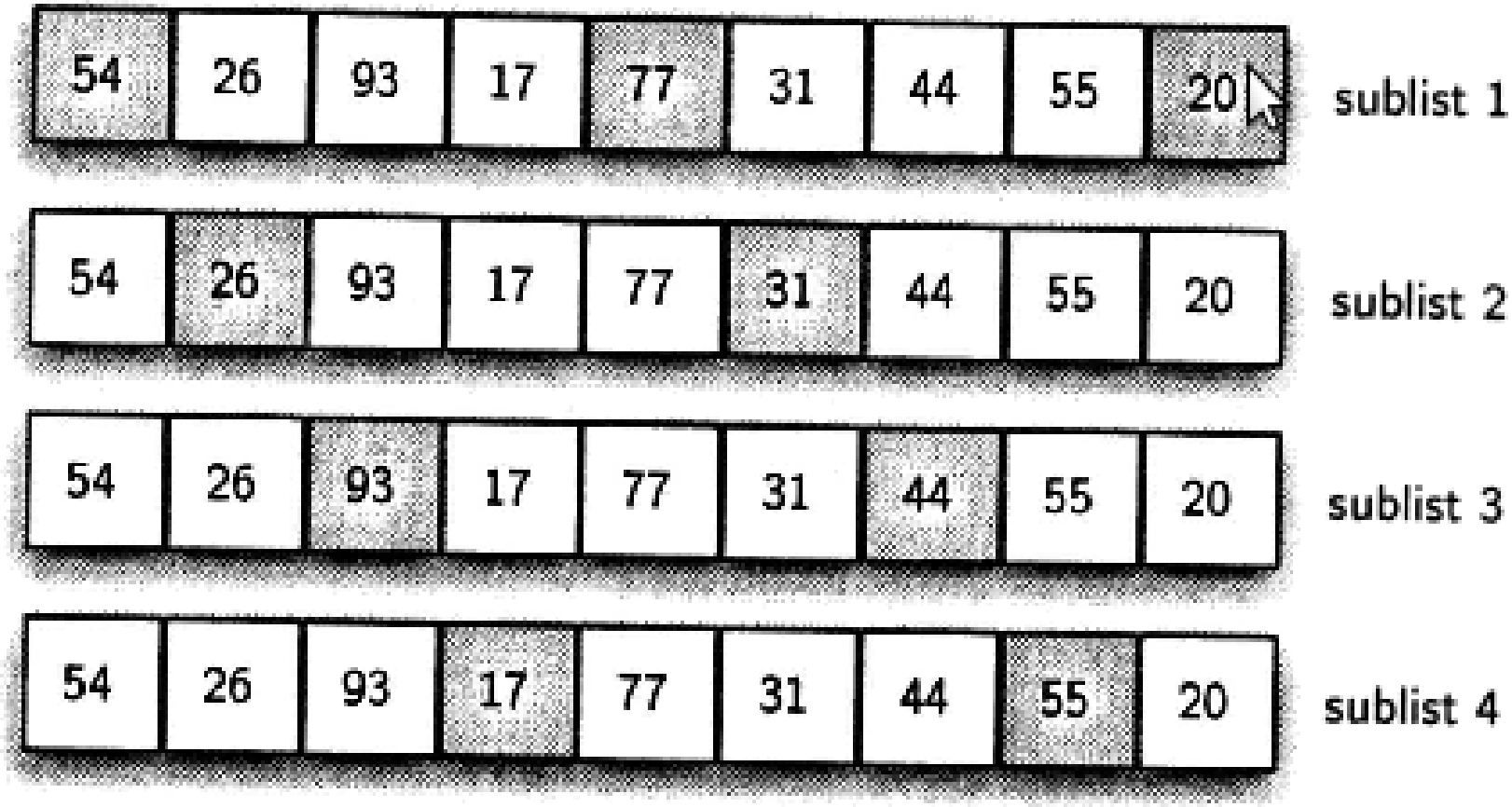
ayrıntılar



→ kaç adet sıralama: 3x altlisteler + 1x nihayi = 4x insertion sort

boyut yarılama

→ altlisteler özyineli bir şekilde boyut yarılamayla oluşturulup, sıralanabilir



→ atlama = n/2 seç, altlisteleri sırala

→ atlama = n/4 seç, altlisteleri sırala

→ atlama = 1 seç, altlisteleri sırala (=nihayi listeyi sırala)

gerçekleme

→ gerçekleme

```
1       def shellSort(alist):
2           sublistcount = len(alist)/2
3           while sublistcount > 0:
4
5             for startposition in range(sublistcount):
6               gapInsertionSort(alist,startposition,sublistcount)
7
8             print "After increments of size",sublistcount,\
9                   "The list is",alist
10
11            sublistcount = sublistcount / 2
12
13      def gapInsertionSort(alist,start,gap):
14          for i in range(start+gap,len(alist),gap):
15
16              currentvalue = alist[i]
17              position = i
18
19              while position>=gap and \
20                      alist[position-gap]>currentvalue:
21                  alist[position]=alist[position-gap]
22                  position = position-gap
23
24              alist[position]=currentvalue
```

demo

→ demo

```
>>> from listing_4_21 import *
>>> alist = [54, 26, 93, 17, 77, 31, 44, 55, 20]
>>> shellSort(alist)
After increments of size 4 The list is
             [20, 26, 44, 17, 54, 31, 93, 55, 77]
After increments of size 2 The list is
             [20, 17, 44, 26, 54, 31, 77, 55, 93]
After increments of size 1 The list is
             [17, 20, 26, 31, 44, 54, 55, 77, 93]
```

insertion X shell sort

→ insertion sorttan çok daha iyidir, diyemeyiz

→ atlama=1 iken insertion sort olur

→ altlistelerden, ana listeye hemen hemen doğru yerlerinde değerlerin gelmeleri

→ bu ise daha az karşılaştırma (veya kaydırma)

analiz

- → yarılayarak ilerlersek: O(n^2)
- → eğer boşlukları gap=2^k-1 (burada k=1,3,7,15,31,...) olacak şekilde seçersek O(n^3/2)