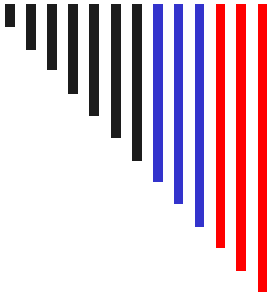
BLM212 Veri Yapıları

Sorting



Sorting

Hedefler

Sorting

- Sıralama en yaygın veri işleme uygulamalarından biridir.

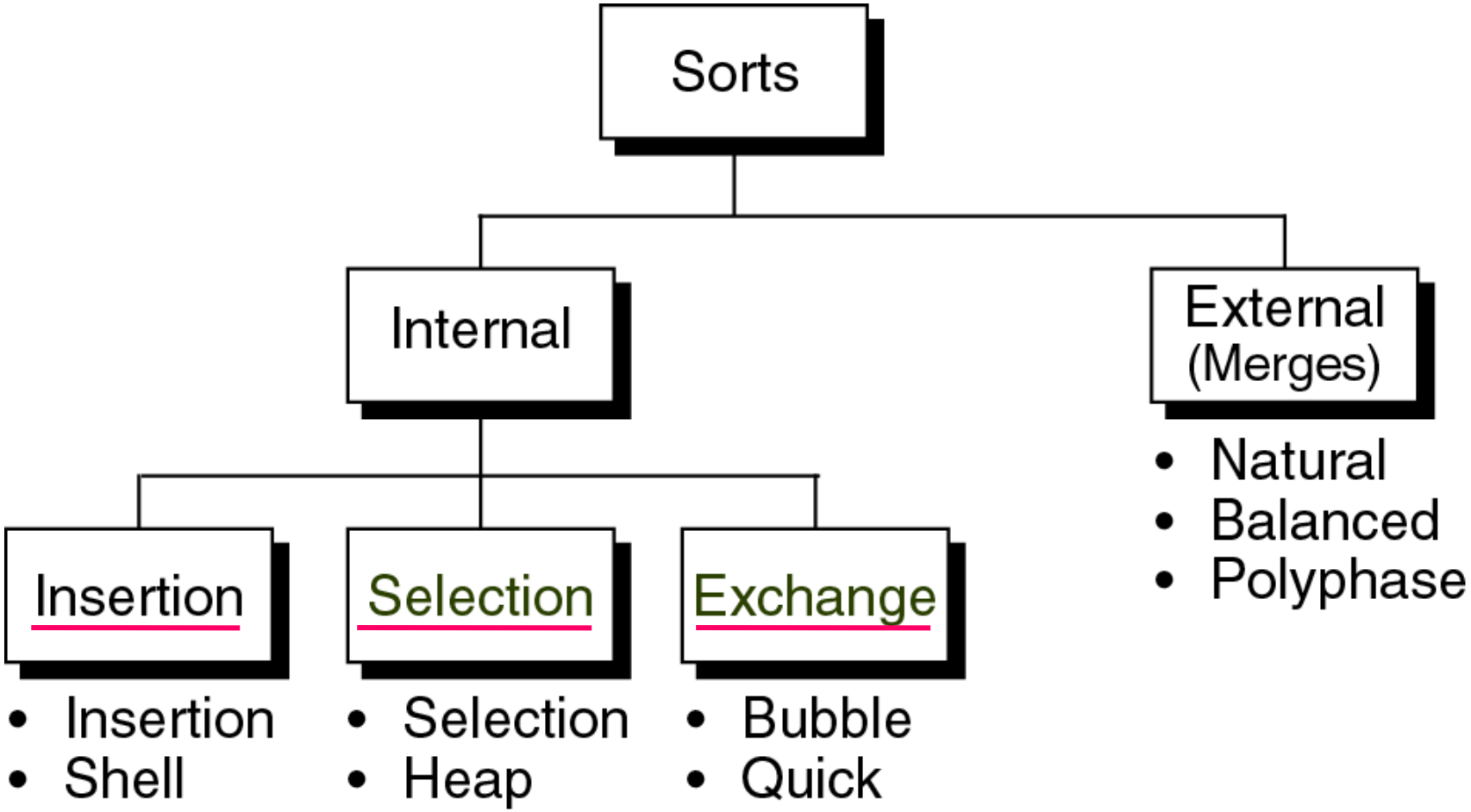
- Sıralama algoritmaları genellikle iki kategoriye ayrılır.
  - Dahili (internal sort,)
  - Harici (external sort)

Sorting

- internal sort: Dahili sıralamada, tüm veriler sıralama işlemi sırasında birincil bellekte tutulur.

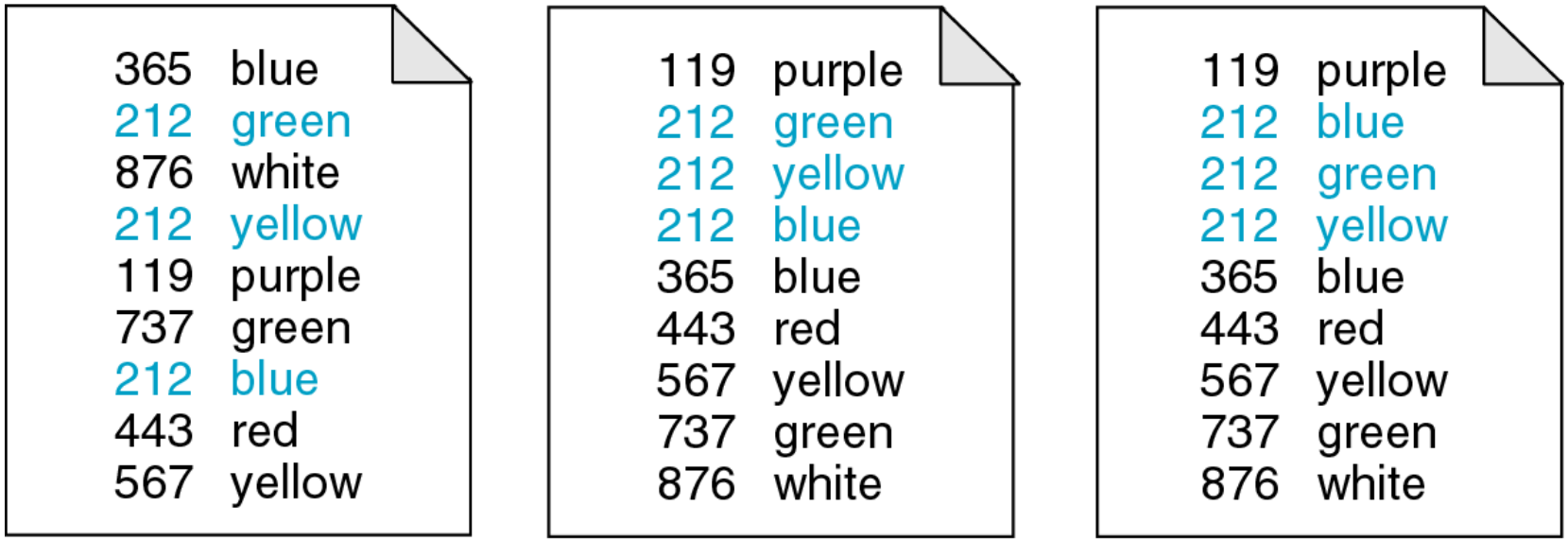
- external sort: Harici sıralama, halihazırda sıralanan veriler için birincil bellek ve birincil belleğe sığmayan veriler için ikincil depolama kullanır.
  - Örneğin, 20.000 kayıtlık bir dosya, yalnızca 1000 kayıt içeren bir dizi kullanılarak da sıralanabilir. Sıralama işlemi sırasında, bir kerede yalnızca 1000 kayıt hafızadadır; diğer 19.000 kişi ikincil depolamada bir dosyada tutulur.

Sort Classification



Sort Stability

Sıralama istikrarlılığı, eşit/aynı anahtarlara sahip verilerin çıktıda göreceli olarak giriş sırasını koruduğunu gösteren bir sıralama özelliğidir.



Buble sort ve **straight insertion sort** sıralaması istikrarlıdır, diğerlerinin tümü istikrarsızdır.

Sort Efficiency (*Sıralama verimliliği*)

- Bir sıralamanın göreceli verimliliğinin bir ölçüsüdür.
- Genellikle sıralı olmayan bir listeyi sıralamak için gereken karşılaştırma ve hamle sayısının bir tahminidir.

- Bununla birlikte, genel olarak, mümkün olan en iyi sıralama algoritmaları ***n* log *n*** düzenindedir;
  - yani, ***O(n* log *n)***
- İnceleyeceğimiz sıralamaların çoğu $O(n^2)$. En iyisi, quick sort : ***O(n* log *n)*** 'dir.

Selection Sort

- Listedeki en küçük öğeyi seçip sıralı bir listeye yerleştiriyoruz.
- Bu adım tüm veriler sıralanana kadar tekrarlanır.

1. **Straight Selection Sort**
2. **Heap Sort**

Selection Sort
Straight Selection Sort

- Liste, iki alt listeye ayrılır: sıralı ve sırasız.
- Sıralanmamış alt listeden en küçük eleman seçilir ve sıralanmamış verilerin başındaki elemanla değiştirilir.
- İki alt liste arasındaki duvar bir ileriye hareket ettirilir.

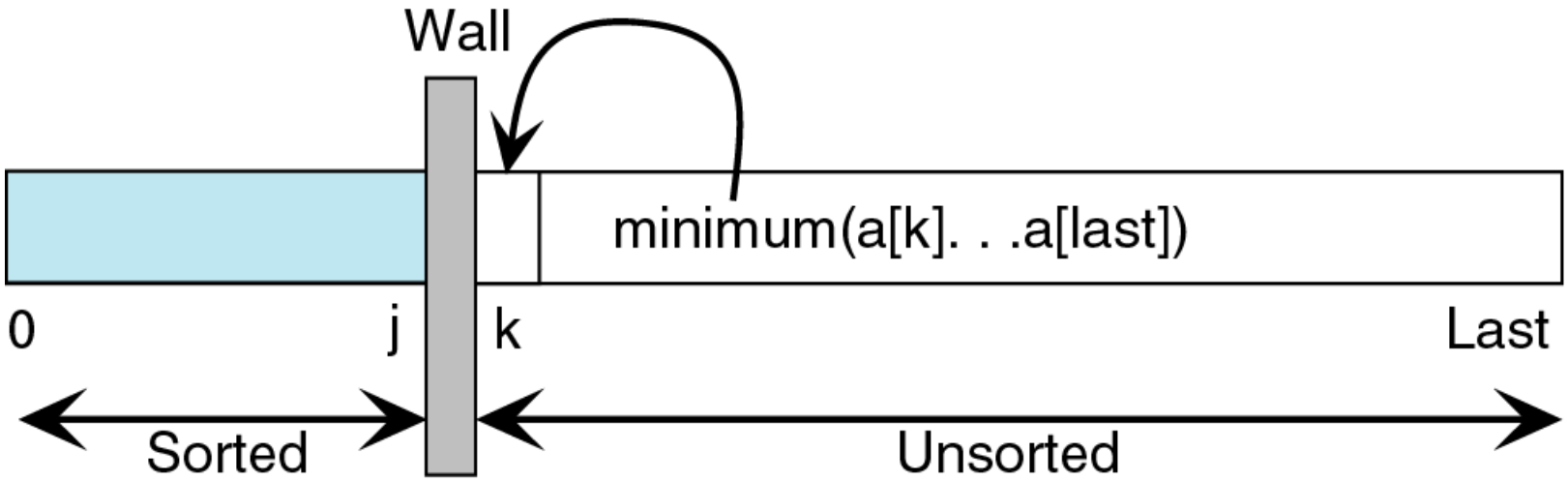


Figure 11-8

Selection Sort

Straight Selection Sort

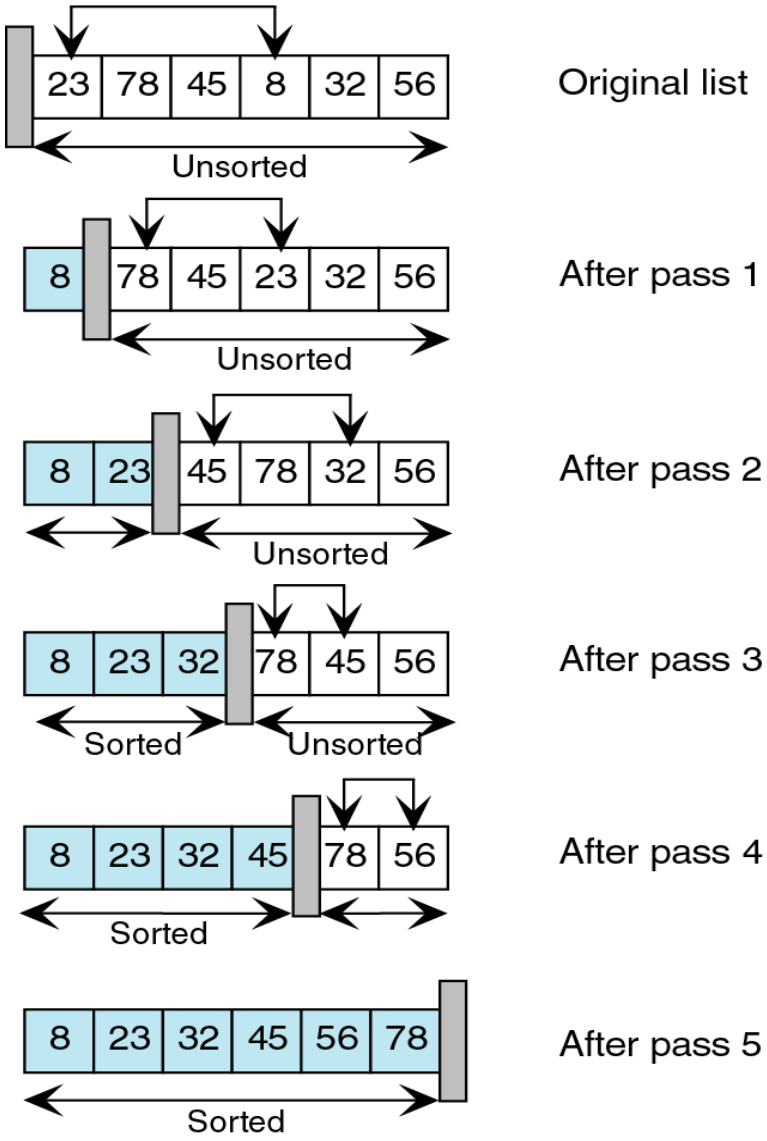


Figure 11-9

Selection Sort
Straight Selection Sort

algorithm **selectionSort**(ref list <array>, val last <index>)

Sort list[1..last] by selecting smallest element in unsorted portion of array and exchanging it with element at the beginning of the unordered list.

Pre List is must contain at least one item.

last is an index to last record in array.

Post List has been rearranged smallest to largest.

Selection Sort
Straight Selection Sort

1. current = 1
2. loop (current < last)
   1. smallest = current
   2. walker = current +1
   3. loop (walker <= last)
      1. if (list[walker] < list[smallest])
         - smallest = walker
      2. walker = walker + 1
   - smallest selected: exchange with current element! exchange (list, current, smallest)
   - current = current + 1
3. return

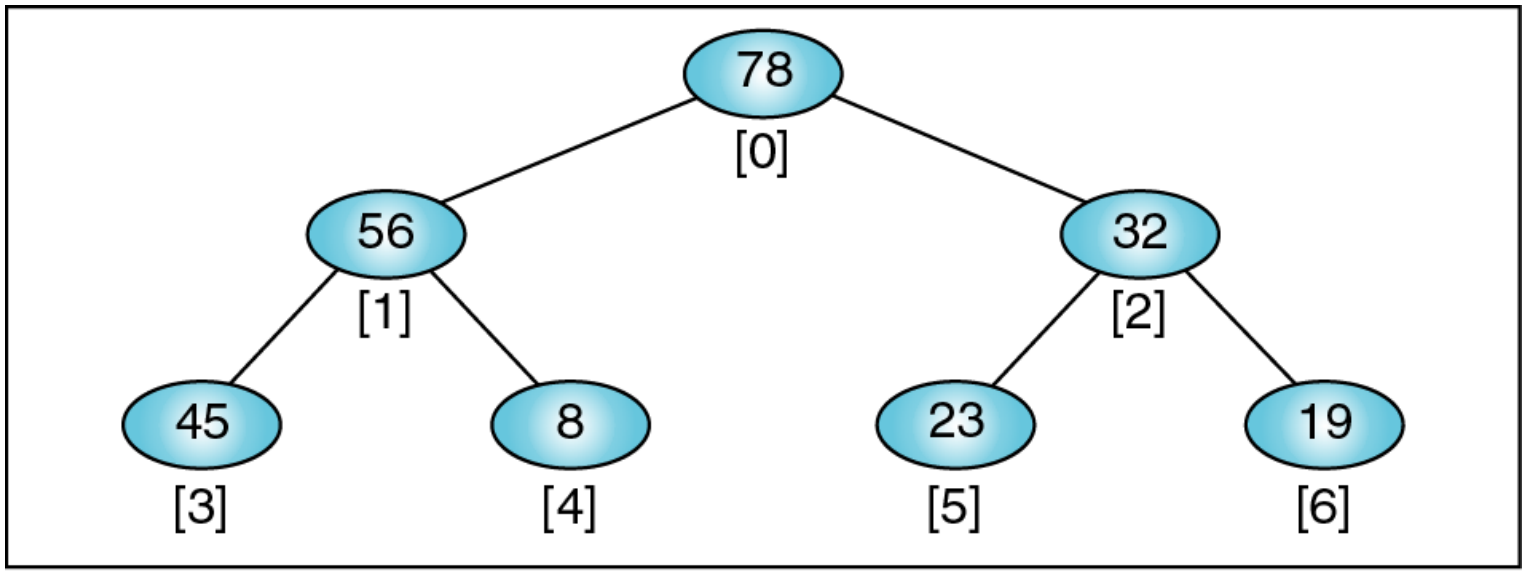
end **selectionSort**

executes n – 1 times

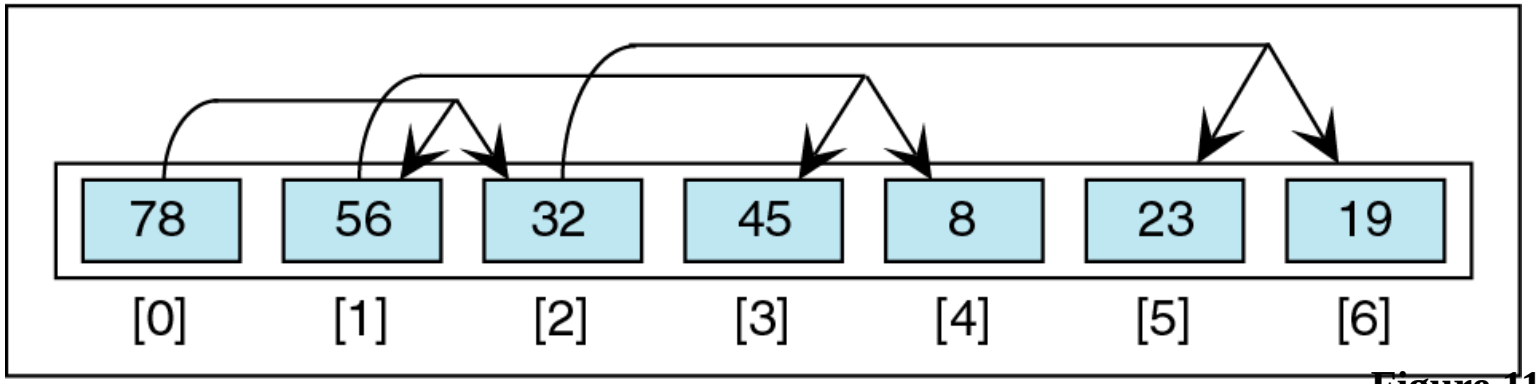
executes n – 1 times

$O(n^2)$

Selection Sort
Heap Sort



(a) A heap in its logical form



(b) A heap in an array

Figure 11-10

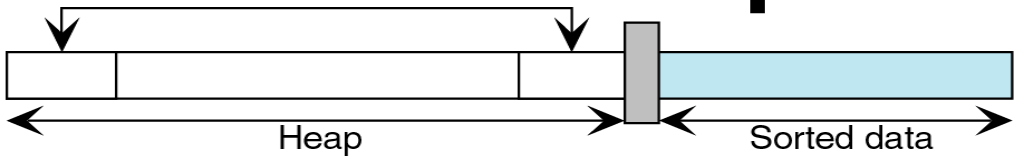
Selection Sort
Heap Sort

ALGORITHM 12-2 Heap Sort

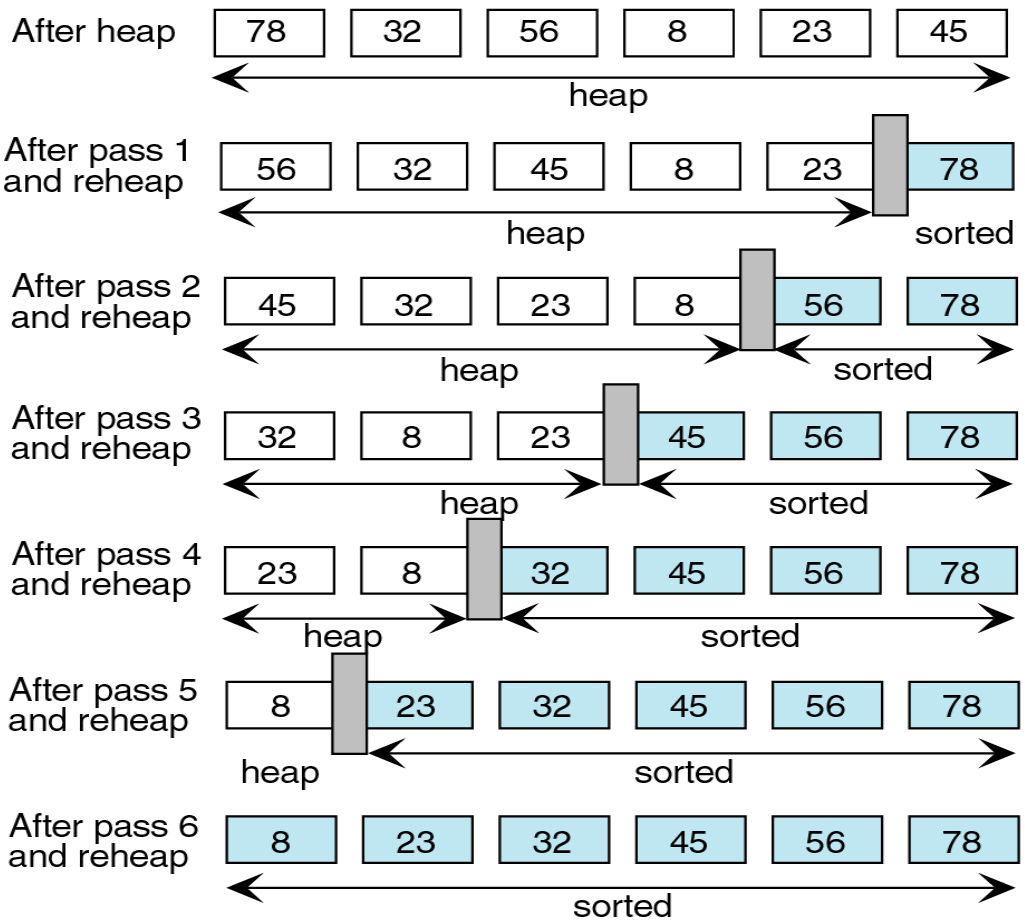
```
Algorithm heapSort (heap, last)
Sort an array, using a heap.
   Pre  heap array is filled
        last is index to last element in array
   Post heap array has been sorted
   Create heap
 1 set walker to 1
 2 loop (heap built)
   1   reheapUp (heap, walker)
   2   increment walker
 3 end loop
   Heap created. Now sort it.
 4 set sorted to last
 5 loop (until all data sorted)
   1   exchange (heap, 0, sorted)
   2   decrement sorted
   3   reheapDown (heap, 0, sorted)
 6 end loop
end heapSort
```

Selection Sort

Heap Sort



(a) Heap sort exchange process



(b) Heap sort process

İkili ağacın dallarını bir kökten bir yaprağa kadar takip etmek log n döngü gerektirir. Sıralama çabası, dış döngü iç döngü ile çarpılır, bu nedenle heap sort için

***n* (log*n*)**

n	Number of loops	
	Straight selection	Heap
25	625	116
100	10,000	664
500	250,000	4482
1000	1,000,000	9965
2000	4,000,000	10,965

TABLE 12-1 Comparison of Selection Sorts

The straight selection sort efficiency is $O(n^2)$.

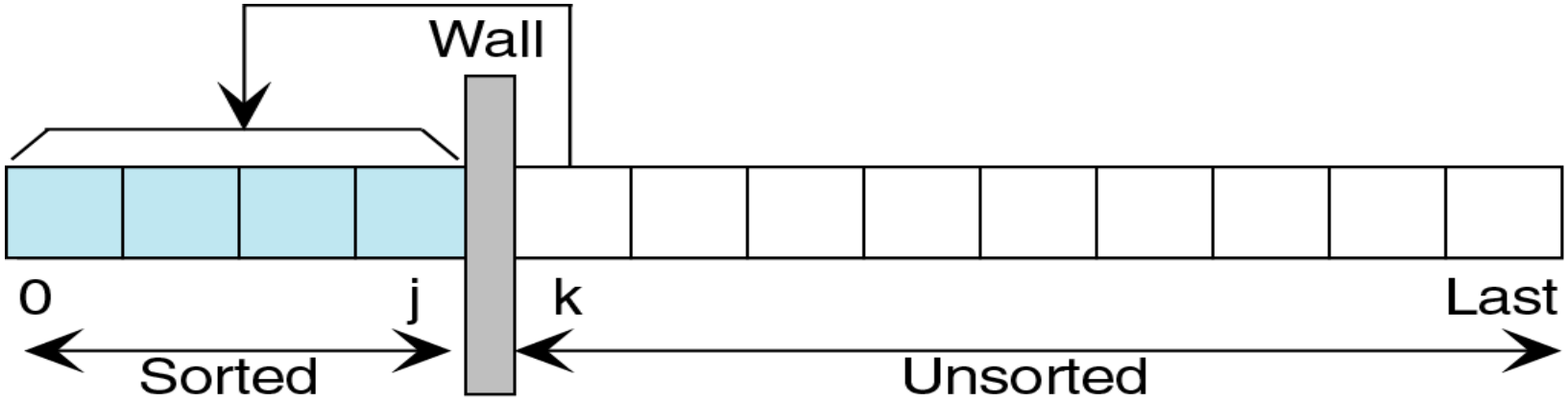
The heap sort efficiency is $O(n \log n)$.

Insertion Sort

- Sıralamak için iki veri parçası kullanır: sıralı (sorted) ve sıralanmamış (unsorted).
- Bir sıralamadaki her geçişte, bir veya daha fazla veri parçası doğru konumlarına eklenir.
    1. The straight insertion sort
    2. The shell sort

Insertion Sort
Straight Insertion Sort

- Sıralamak için iki veri parçası kullanır: sıralı ve sıralanmamış
- Her geçişte, sıralanmamış alt listenin ilk öğesi, uygun yere eklenerek sıralı alt listeye aktarılır.



- Verileri sıralamak için en fazla **n-1** geçiş gerekir.

Figure 11-3

Insertion Sort

Straight Insertion Sort

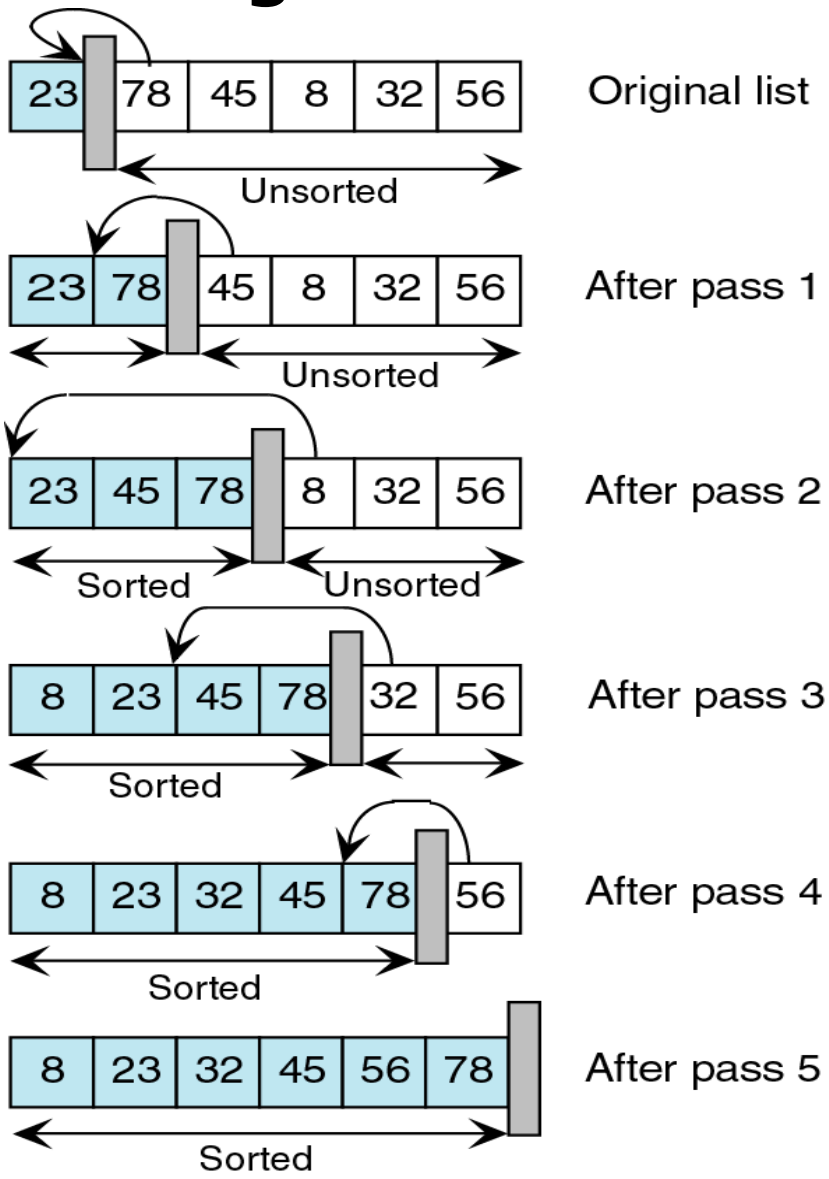


Figure 11-4

Insertion Sort
Straight Insertion Sort

algorithm **insertionSort**(ref list <array>, val last <index>)

Sort list[1..last] using insertion sort. The array is divided into sorted and unsorted lists. With each pass, the first element of unsorted list is inserted into sorted list.

Pre List must contain at least one element.

last is an index to last element in the list.

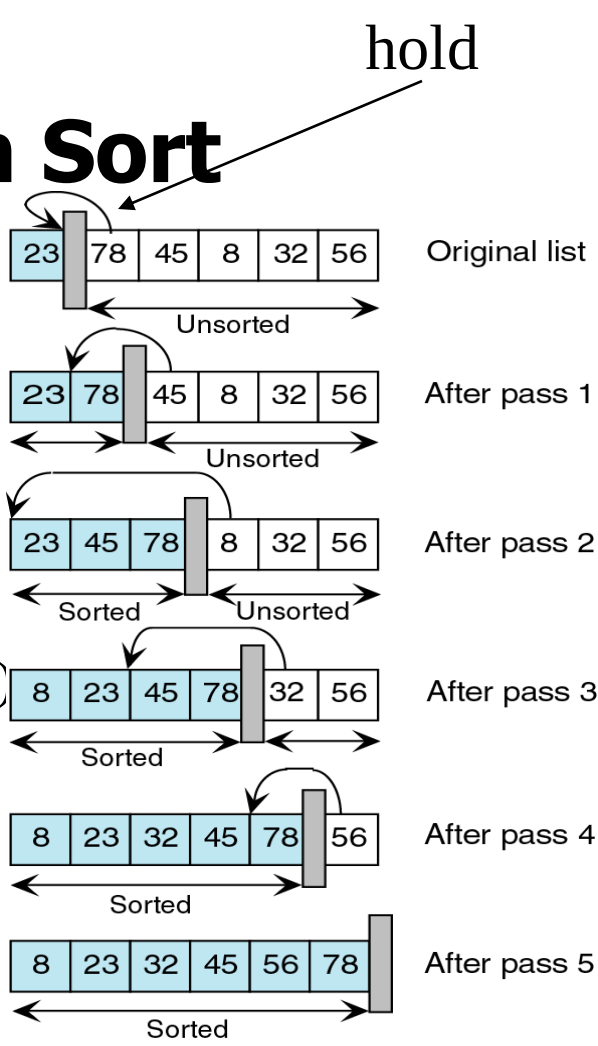
Post List has been rearranged.

Insertion Sort

Straight Insertion Sort

1. current = 2
2. loop (current <= last)
    1. hold = list[current]
    2. walker = current -1
    3. loop (walker >=1 AND hold.key < list[walker].key)
        - list[walker+1] = list[walker]
        - walker = walker -1
    4. list[walker +1] = hold
    5. current = current +1
3. return

end **insertionSort**

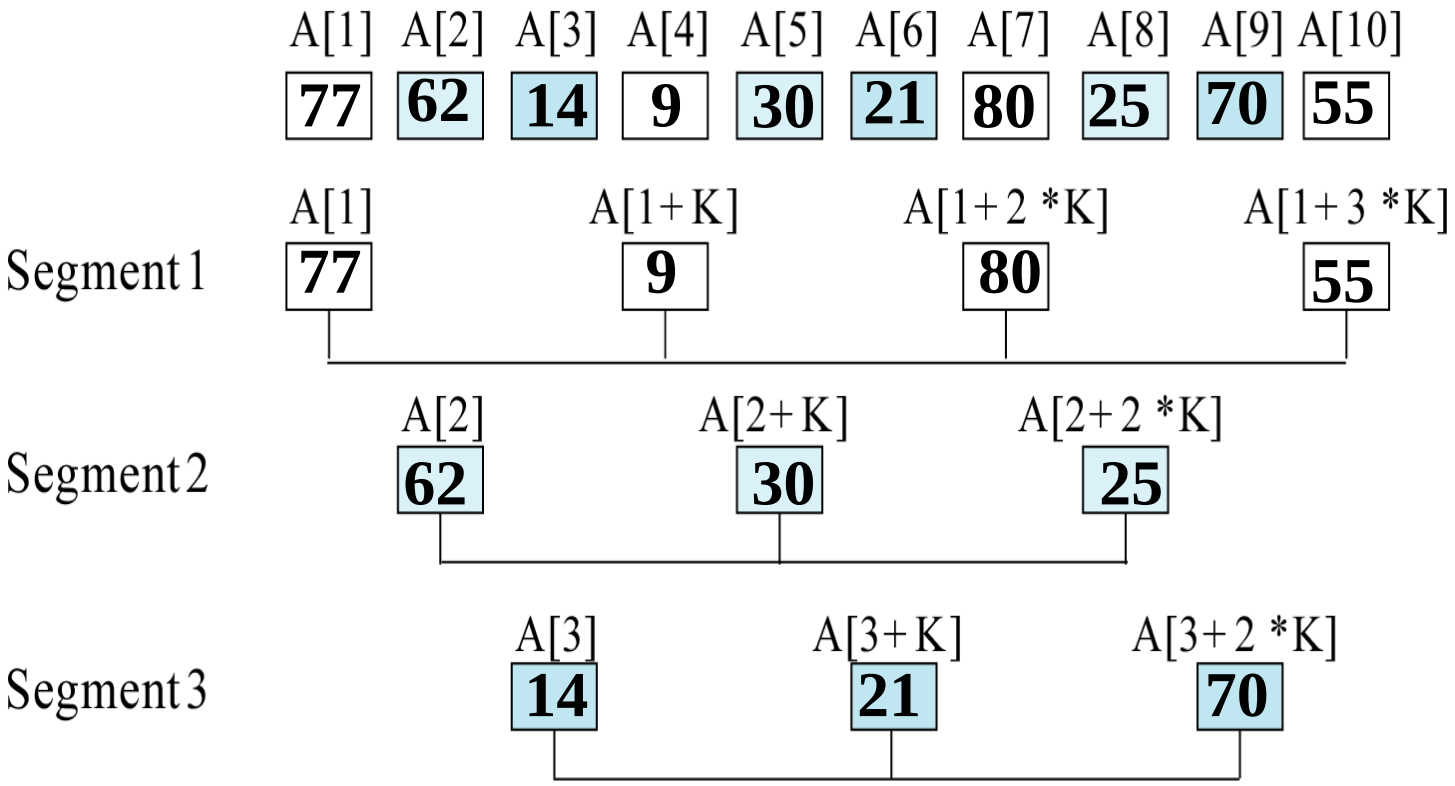


Inside loop is quadraticly dependent to outer loop and outer loop executes n-1 times.
f(n)=n((n+1)/2)
$O(n^2)$

Insertion Sort
Shell Sort

- Bu algoritmayı bulan : Donald L.Shell.
- Liste, K artım değeri (increment value) olarak bilinen K segmente ayrılmıştır.
- Her segment N/K veya daha az eleman içerir.
- Her geçişten sonra, her segmentteki veriler sıralanır ve K değeri azaltılır.
- Süreç sonunda yalnızca bir segment kalmışsa, o zaman liste sıralanmış demektir.

Insertion Sort
Shell Sort



K = 3, üç segment ve K artış değeridir
Her bir segment N/K veya daha az eleman içerir.
Eğer tek bir segment varsa, o zaman liste sıralıdır.

Figure 11-5

Insertion Sort
Shell Sort

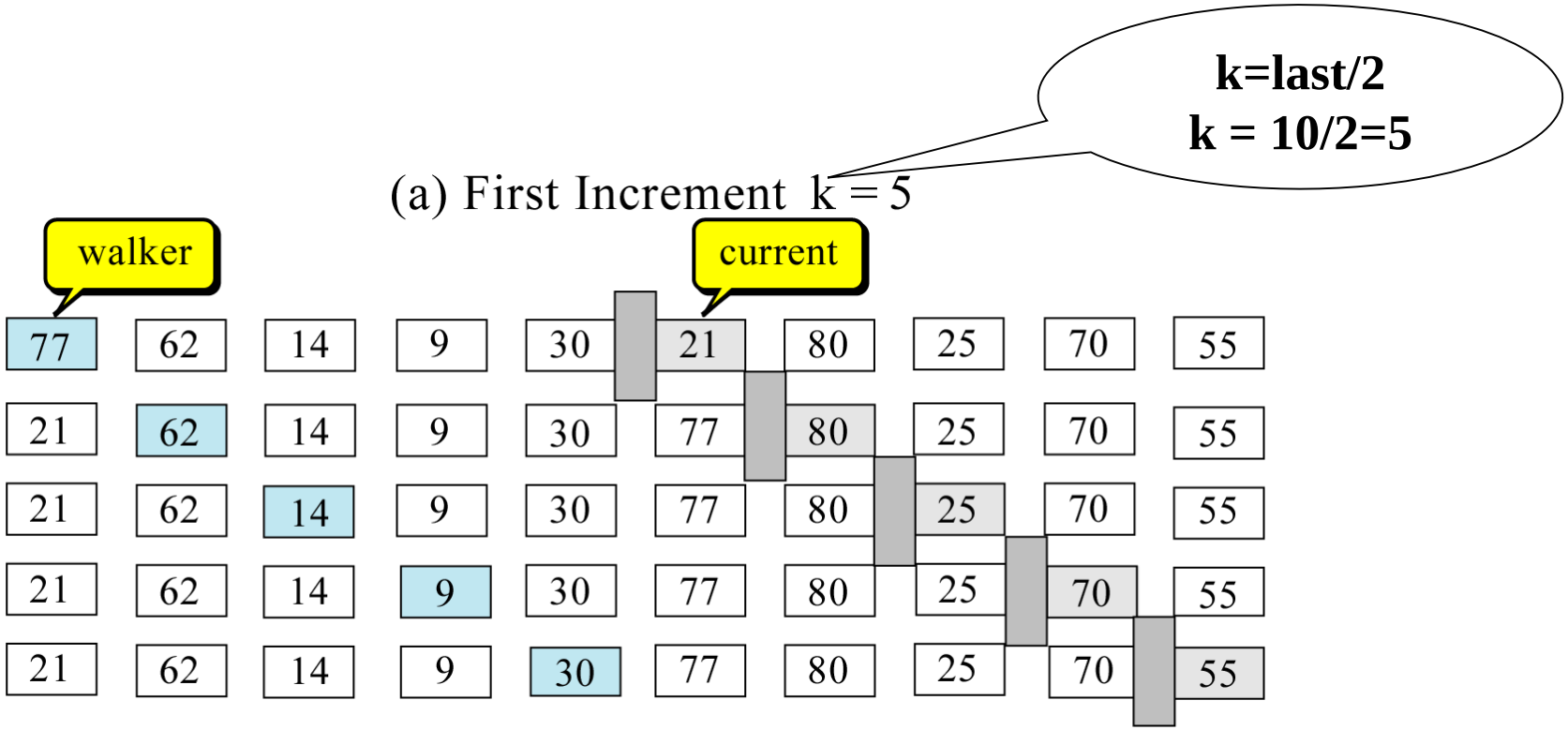


FIGURE 12-10 Diminishing Increments in Shell Sort

Insertion Sort
Shell Sort

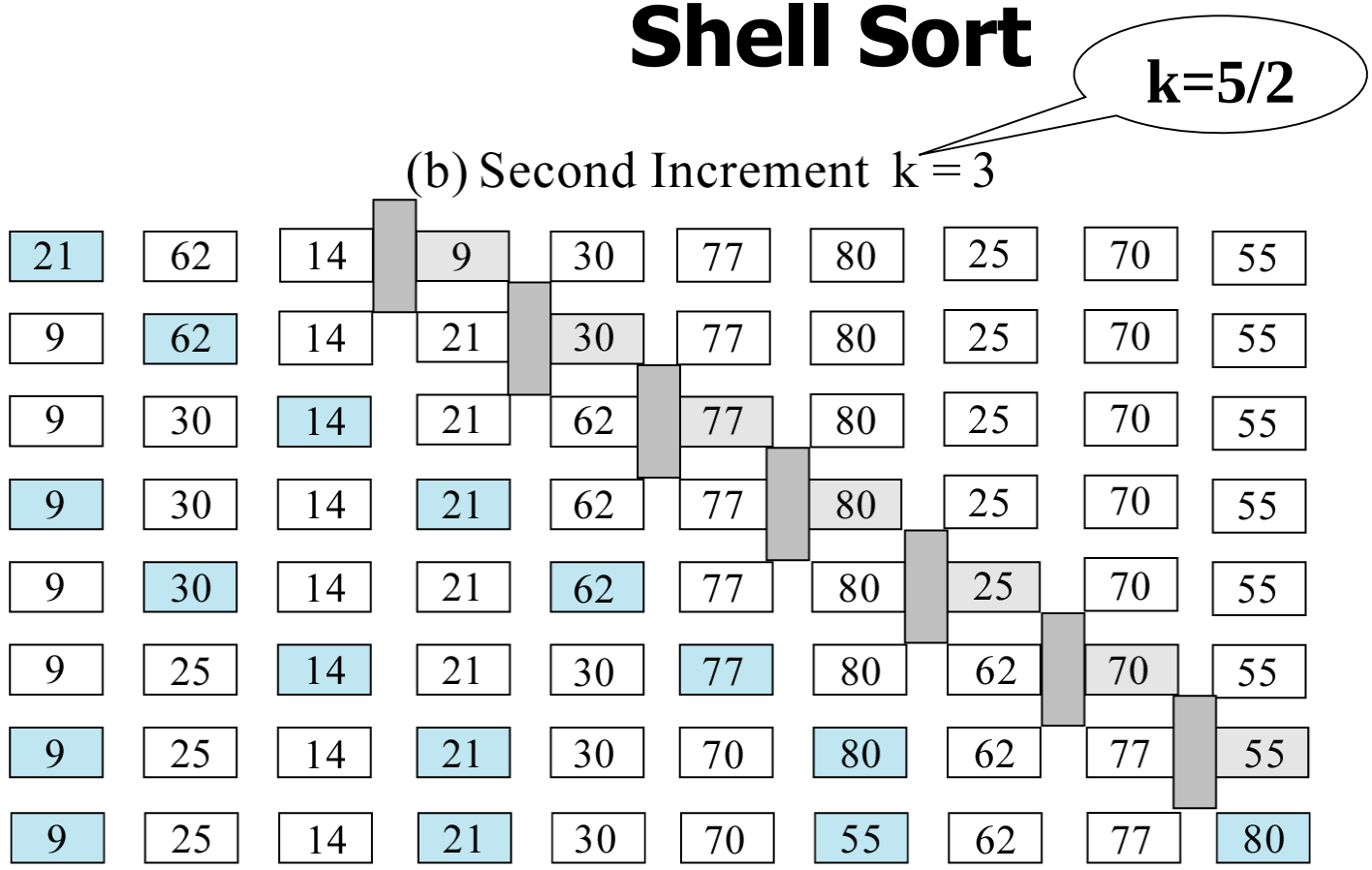


Figure 11-6, b

FIGURE 12-10 Diminishing Increments in Shell Sort

Insertion Sort
Shell Sort

k=3/2

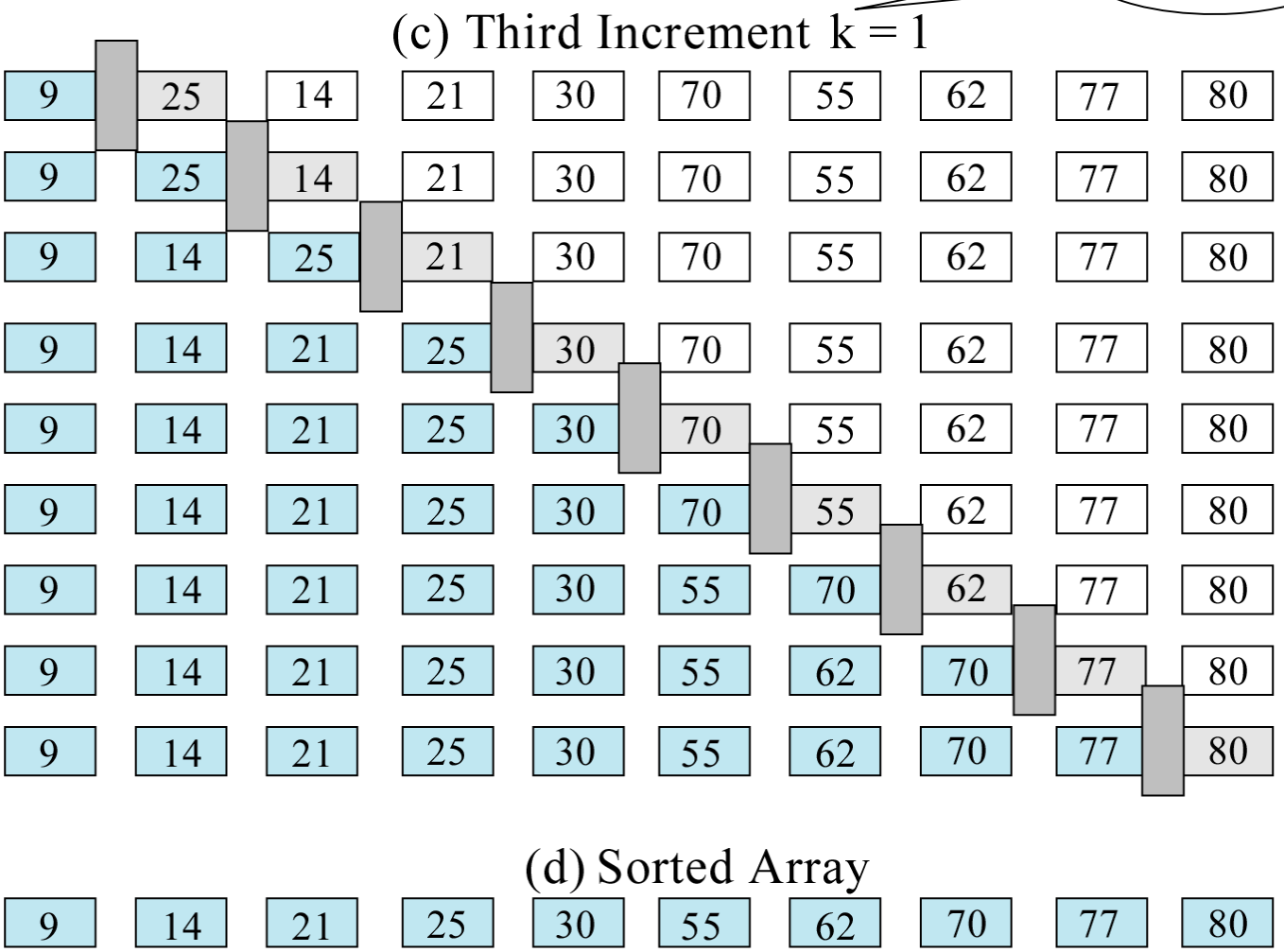
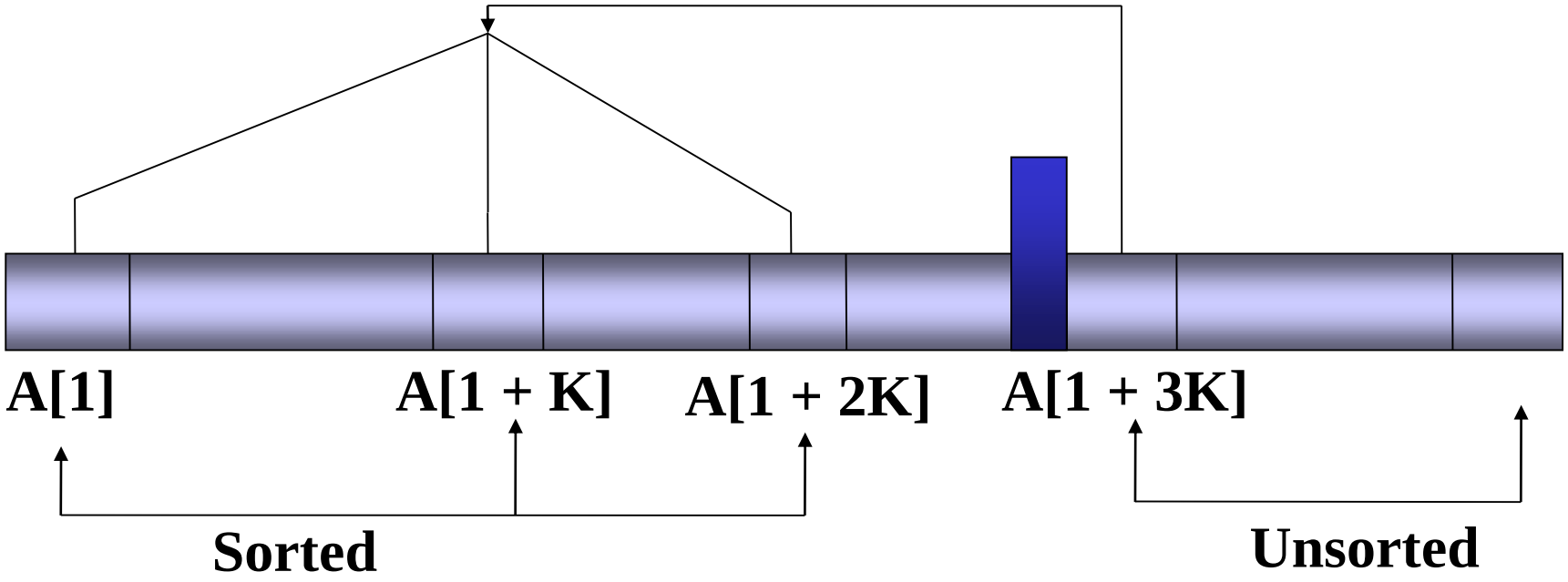


FIGURE 12-10 Diminishing Increments in Shell Sort

Insertion Sort
Shell Sort



Insertion Sort
Shell Sort

algorithm **shellSort**(ref list <array>, val last <index>)

Sort list[1], list[2],..,list[last] are sorted in place. After the sort,
their keys will be in order,

list[1].key <= list[2].key <=..<=list[last].key.

Pre List is an unordered array of records.

last is an index to last record in array.

Post List is ordered on list[i].key.

Insertion Sort
Shell Sort

1. incre = last /2
2. loop (incre not 0)
    1. current = 1 + incre
    2. loop (current <= last)
        1. hold = list[current]
        2. walker = current – incre
        3. loop (walker >= 1 AND hold.key < list[walker].key)
            - list[walker+incre] = list[walker]
            - walker = walker – incre
        4. list[walker + incre] = hold
        5. current = current + 1
    3. incre = incre /2
3. return

end shellSort

executes $\log_2 n$

n - increment times

each time n = n/2

$\log_2 n[(n-n/2)+(n-n/4)+..+1]=n.\log_2 n$
$O(n.\log_2 n)$
We still need to include the third loop!
$O(n^{1.25})$

Deneysel çalışmalar ortala sıralama verimliliğinin bu olduğunu gösterir

The shell sort efficiency is $O(n^{1.25})$.

n	Number of loops		
	Straight insertion Straight selection	Shell	Heap
25	625	55	116
100	10,000	316	664
500	250,000	2364	4482
1000	1,000,000	5623	9965
2000	4,000,000	13,374	10,965

TABLE 12-2 Comparison of Insertion and Selection Sorts

The straight insertion sort efficiency is $O(n^2)$.

The shell sort efficiency is $O(n^{1.25})$.

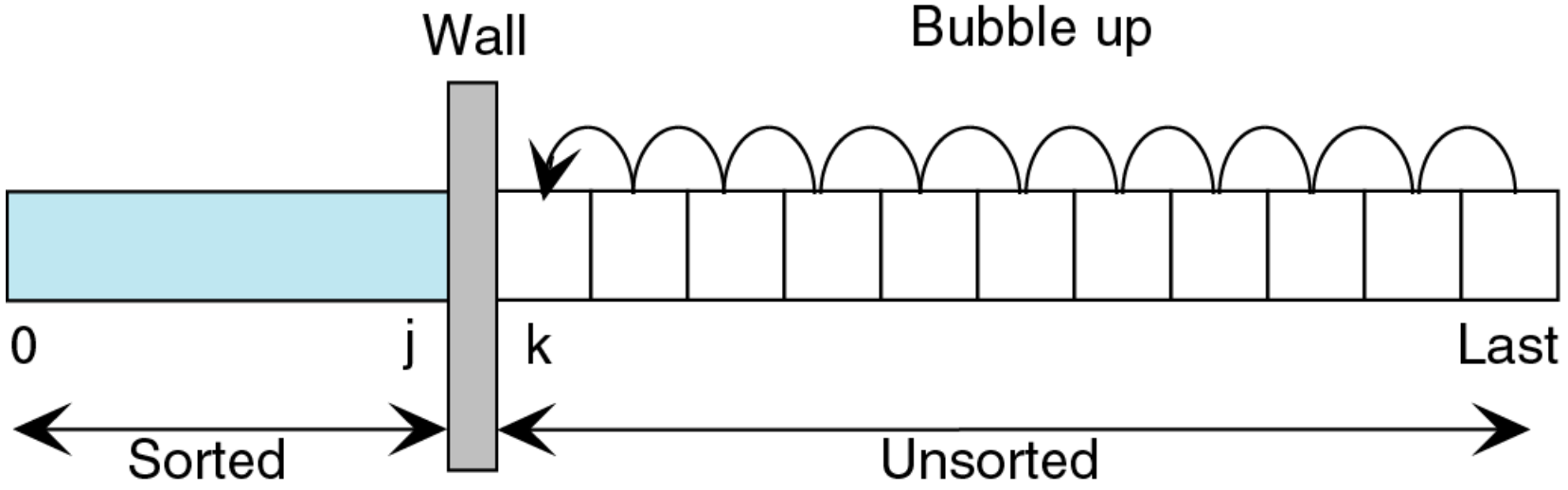
Exchange Sort

Değiş-tokuşlu sıralama

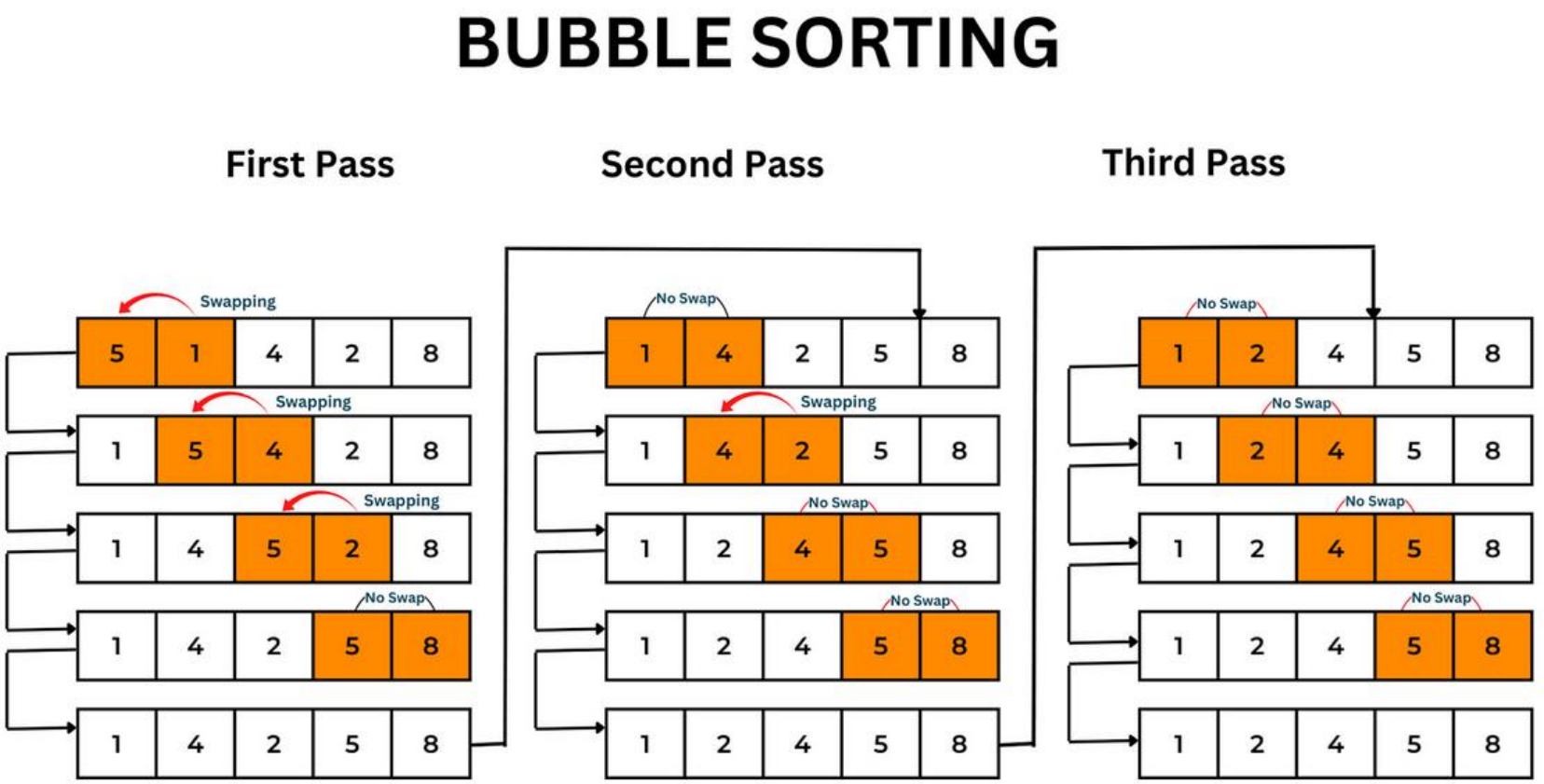
1. Bilgisayar bilimlerindeki en yaygın sıralama: **buble sort.**
2. En etkin genel sıralama: **quick sort** .

Exchange Sort
Bubble Sort

- Liste iki alt listeye ayrılmıştır: sıralanmış ve sıralanmamış.
- En küçük eleman sıralanmamış listeden sıralı listeye taşınır.
- Duvar bir eleman sağa hareket ettirilir.
- Bu sıralama, verileri sıralamak için n-1 geçiş gerektirir.



Exchange Sort
Bubble Sort



Exchange Sort
Bubble Sort

algorithm **bubleSort**(ref list <array>, val last <index>)

Sort an array, list[1..last] using buble sort.Adjacent elements are compared and exchanged until list is completely ordered.

Pre List is must contain at least one item.

last is an index to last record in array.

Post List has been rearranged smallest to largest.

Exchange Sort
Bubble Sort

- current = 1
- sorted = false
- loop (current <= last AND sorted false)

1. walker = last
2. sorted = true
3. loop (walker > current)
    1. if (list[walker] < list[walker-1])
        - sorted = false
        - exchange (list, walker, walker-1)
    2. walker = walker – 1
4. current = current + 1

1. return

end **bubleSort**

executes n times

executes (n+1)/2 times

$f(n) = n((n+1)/2)$
$O(n^2)$

Exchange Sort

- Buble sort'ta, listeden her geçişte **komşu** elemanlar karşılaştırılır ve muhtemelen değiştirilir, yani bir öğeyi doğru konumuna taşımak için birçok değiş-tokuş gerekebilir.

- Quick sort'ta ise **birbirlerinden çok uzaktaki** elemanların değiş tokuşu söz konusudur böylece bir elemanı doğru konuma getirmek için daha az değiş-tokuşa ihtiyaç duyulur.

Exchange Sort
Quick Sort

- Her bir yineleme için "pivot" olarak adlandırılan bir öğe seçer.
- Listeyi üç gruba ayırır:
  1. Pivot anahtarından daha küçük anahtar değerlere sahip ilk grubun öğeleri.
  2. Pivot eleman
  3. Pivot anahtarından daha büyük anahtar değerlere sahip ilk grubun öğeleri.
- Sıralama, sol bölümü quick sort ve ardından sağ bölümü quick sort algoritması ile sıralayarak devam eder.

Exchange Sort
Quick Sort

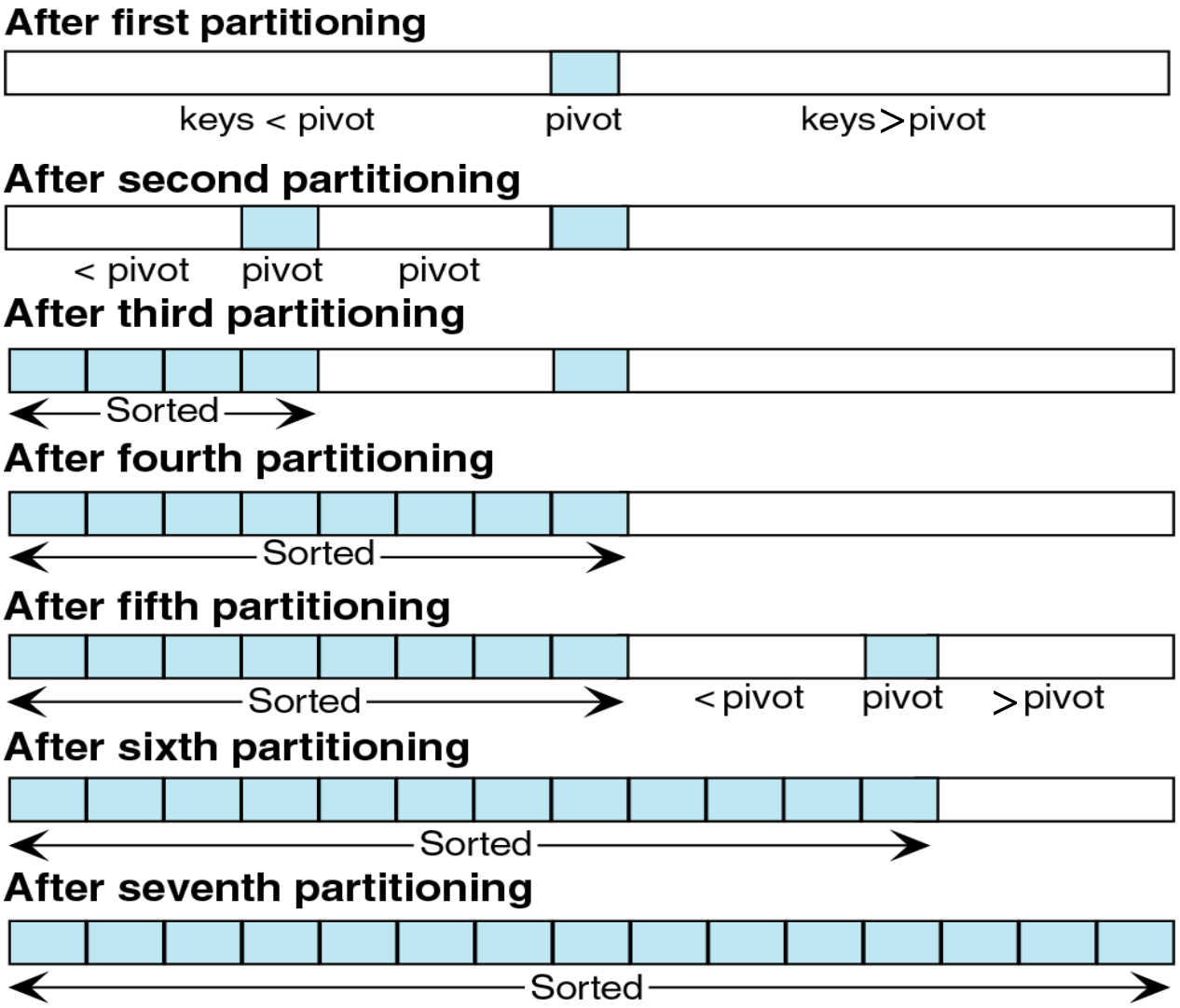


Figure 11-14

Exchange Sort
Quick Sort Operation

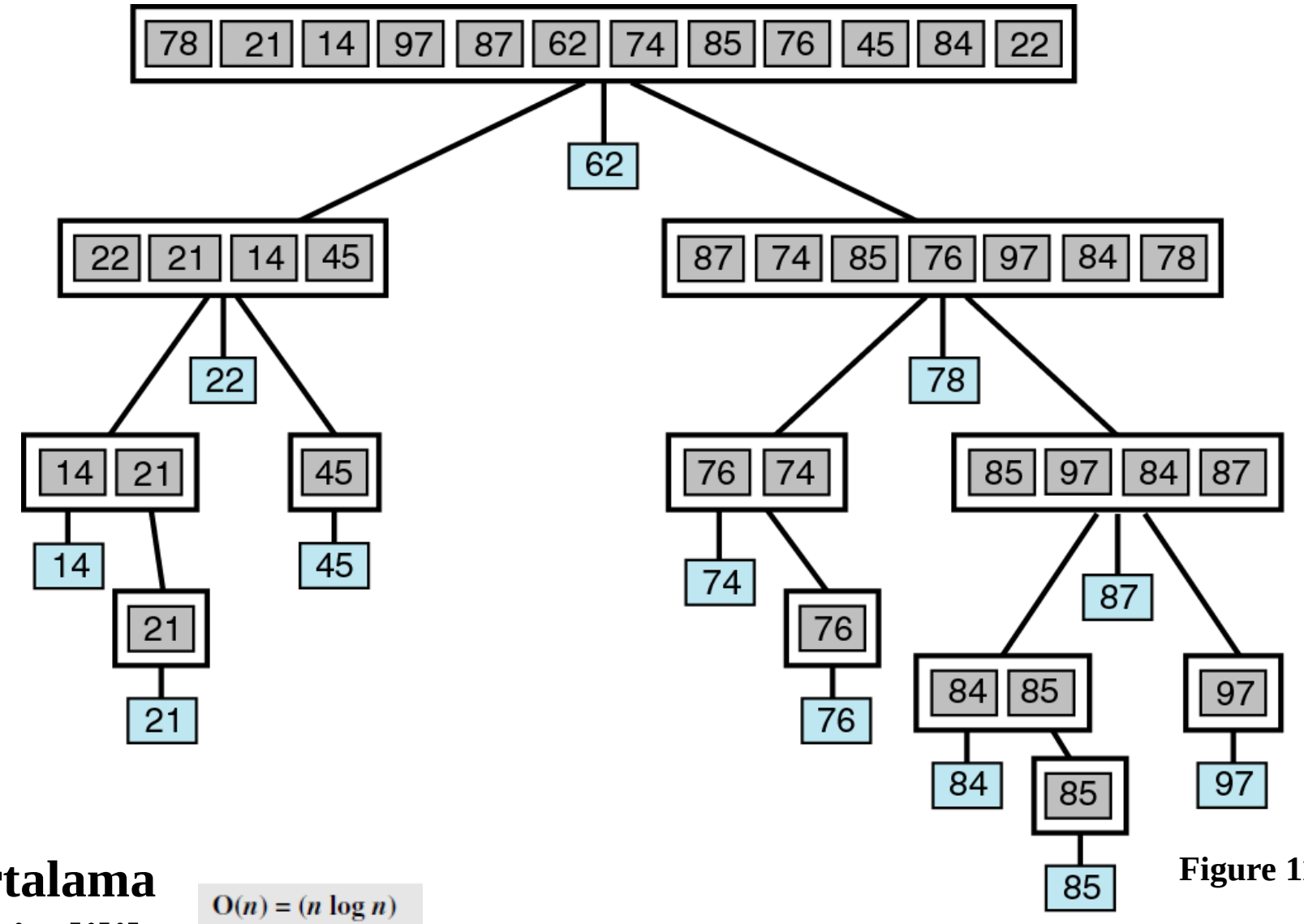


Figure 11-16

Ortalama verimlilik

$O(n) = (n \log n)$

n	Number of loops		
	Straight insertion Straight selection Bubble sorts	Shell	Heap and quick
25	625	55	116
100	10,000	316	664
500	250,000	2364	4482
1000	1,000,000	5623	9965
2000	4,000,000	13,374	10,965

TABLE 12-3 Sort Comparisons