Analyzing Algorithm and Sorting

ITX2010, CSX3003, IT2230

Data Structures and Algorithms, Information Structures

Learning Objectives

Students will be able to:

- Describe concept of six internal sorts – including Insertion sort, Shell sort, Selection sort, Heap sort, Bubble sort and Quick sort.
- Explain steps of work of each internal sorts
- Illustrate or implement all internal sort
- Examine the insertion sort algorithm to solve the sorting problem
- Specify algorithms using a pseudocode
- Introduce a notation that describe how running time increase with the item numbers to be sorted
- Introduce divide-and-conquer to develop a merge sort
- Analyze the merge sort's running time

Chapter Outline

1

Analysing Algorithms and Sorting

1) Insertion Sort
2) Shell Sort
3) Selection Sort
4) Heap Sort
5) Bubble Sort
6) Quick Sort

Sorting

- The goal is to rearrange the elements so that they are ordered from smallest to largest as ascending order (or largest to smallest as descending order). [1]
- Sorting is the process of arranging or ordering a collection of items such that each item and its successor satisfy a prescribed relationship. [2]
- Sorting is one of the most common data processing applications in computing today! [3]
- It is a process through which data are arranged according to their values. [4]

Sorting

- There are two main sort groups:
  1. Internal sort
  2. External sort

- **Internal sort** is a sort in which all of the data are held in primary memory during the sorting process.
- **External sort** uses primary memory for the data currently being sorted and secondary storage for any data that will not fit in primary memory.

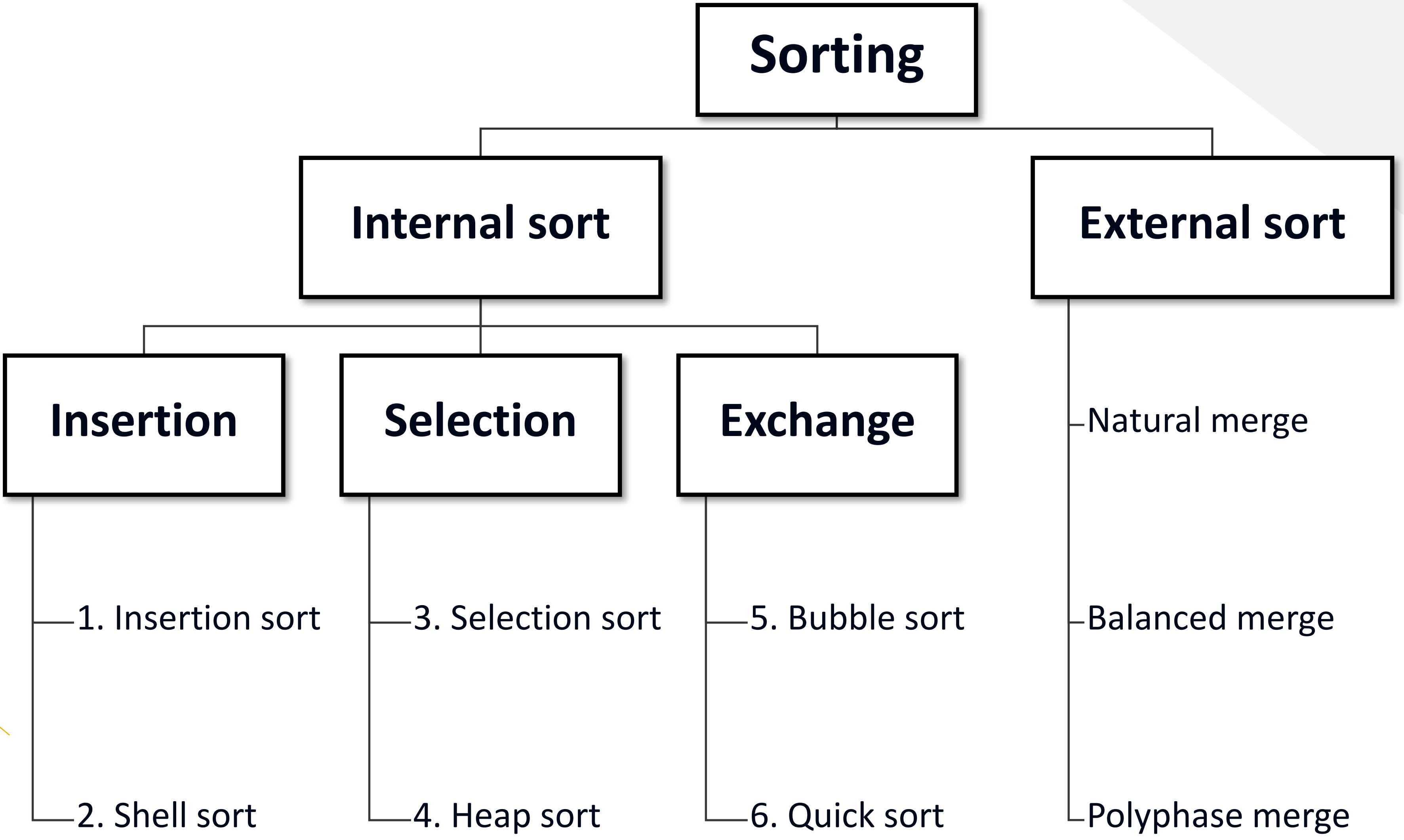


Fig 15-1 Sort Classification

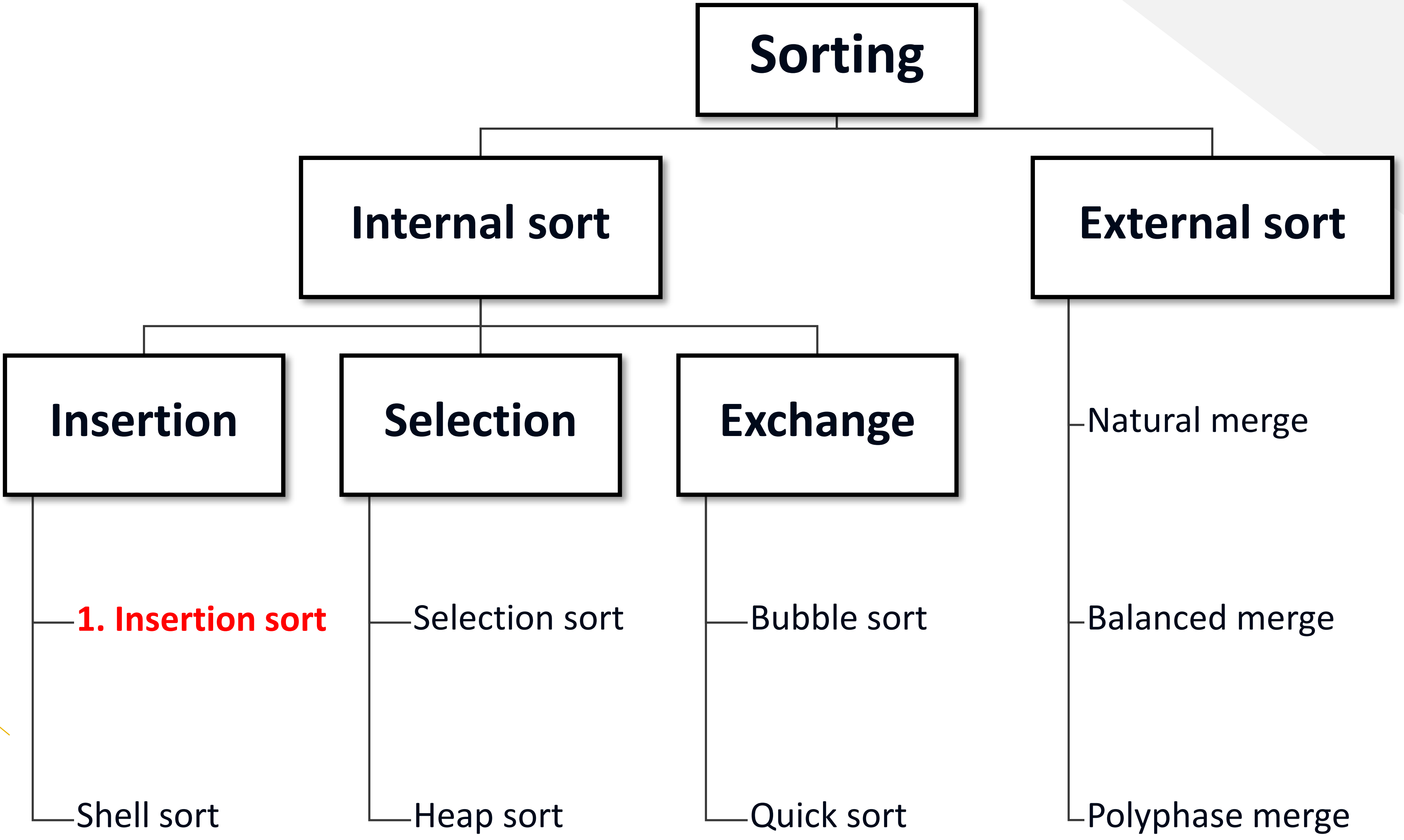


Fig 15-1 Sort Classification (Cont.)

Insertion Sort

- Insertion sort is one of the most common sorting technique used by card players.
- As you pick up each card, you insert it into the proper sequence in your hand.
- In each pass of an insertion sort, one or more pieces of data are inserted into their correct location in an ordered list.
- Two insertion sorts :
    1. Insertion sort (Straight insertion sort)
    2. Shell sort

Insertion Sort

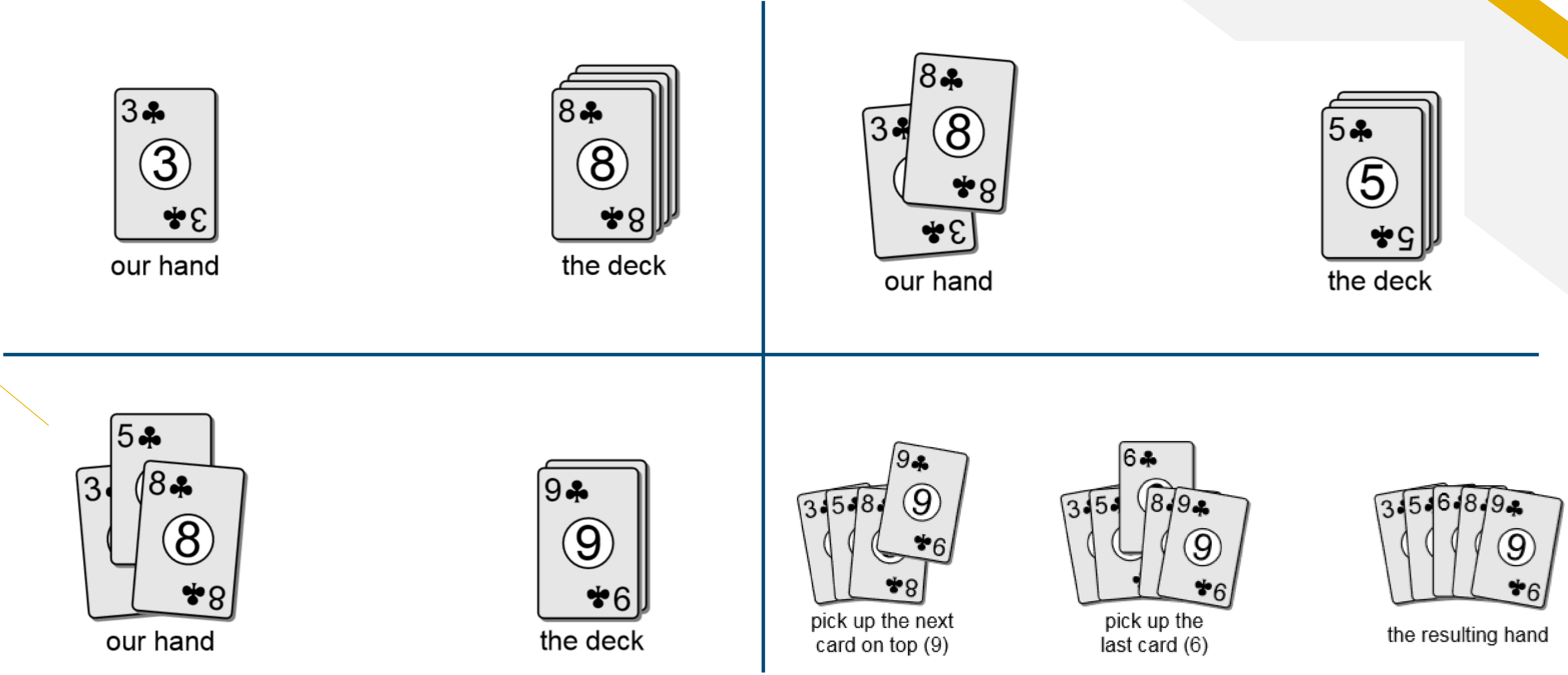
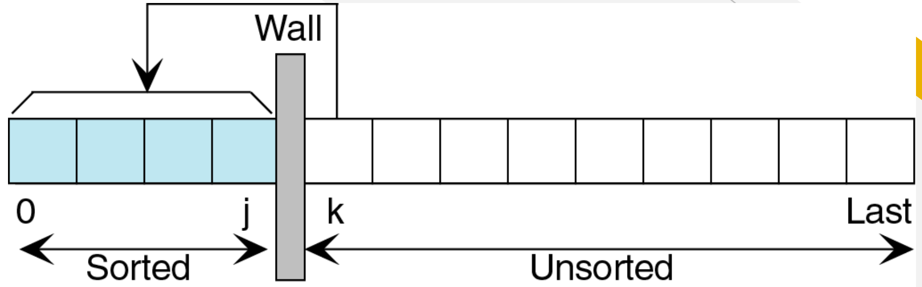


Fig 15-2 Insertion sort concept [2]

Insertion Sort

- Steps of work:
  1. Partition the list into two parts – including left part contains sorted list and the right part contains unsorted list
  2. Read the key in sequence
  3. Find the corrected position to insert the read key.
  4. Repeat step 2 until there is no longer key to be sorted.



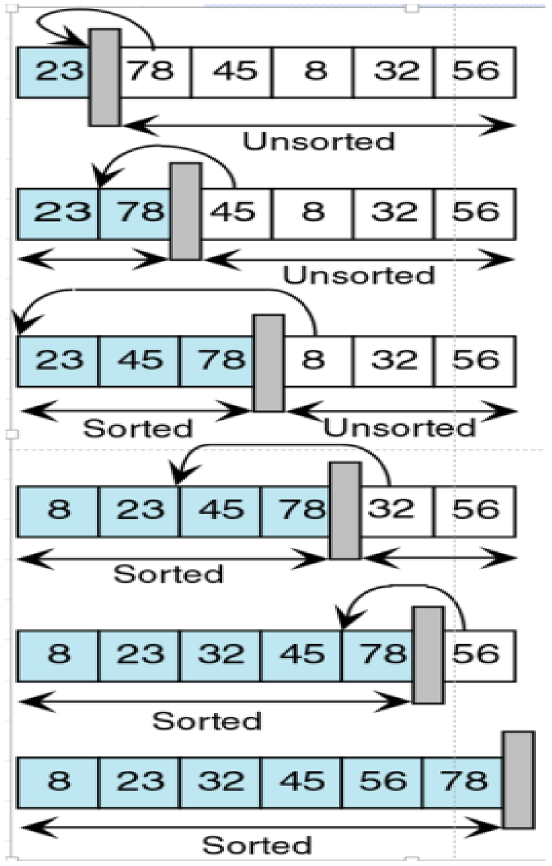
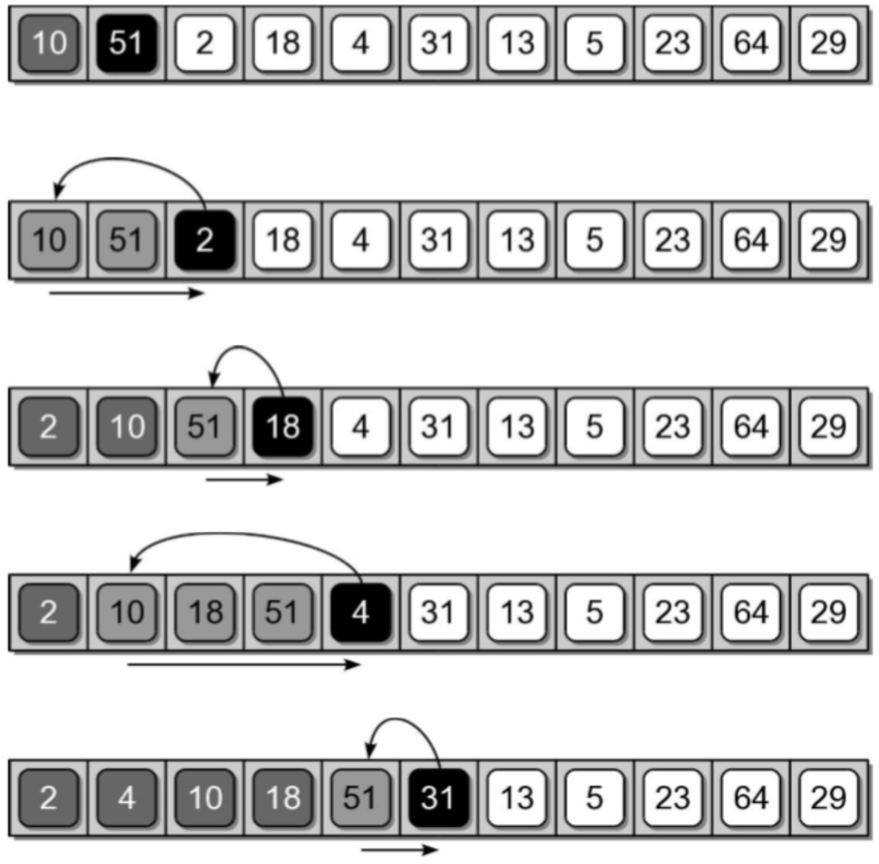
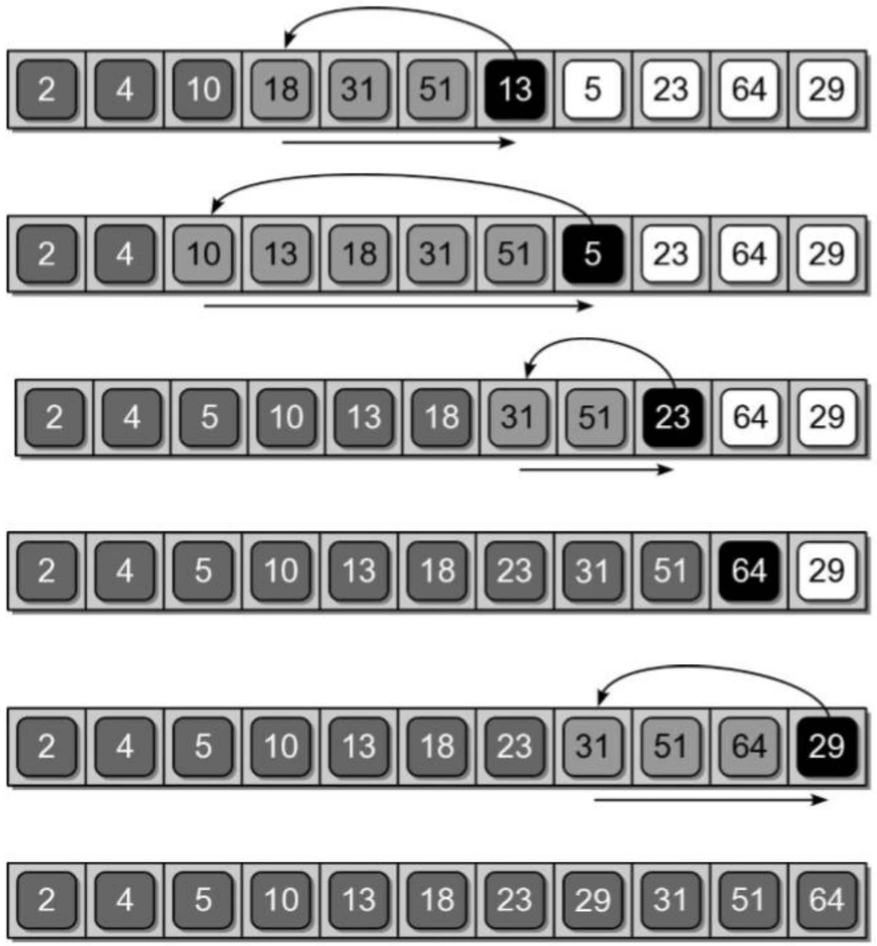


Fig 15-3 Insertion sort logical view [3]

Tutorial 1: Illustration of Insertion Sort
Illustration [2]





Tutorial 2: Insertion Sort Implementation [2]

1.1

```
# Sorts a sequence in ascending order using the insertion sort algorithm.
def insertionSort( theSeq ):
  n = len( theSeq )
   # Starts with the first item as the only sorted entry.
  for i in range( 1, n ) :
     # Save the value to be positioned.
    value = theSeq[i]
     # Find the position where value fits in the ordered part of the list.
    pos = i
    while pos > 0 and value < theSeq[pos - 1] :
       # Shift the items to the right during the search.
      theSeq[pos] = theSeq[pos - 1]
      pos -= 1

     # Put the saved value into the open slot.
    theSeq[pos] = value
```

Insertion Sort

- It can be determined in several ways:
  - The number of loops in the sort
  - The number of moves and compares needed to sort the list.
- In Big-O notation, the quadratic-dependent loop is $O(n^2)$

Insertion Sort

```
INSERTION-SORT($A$, $n$)
for $i = 2$ to $n$
    $key = A[i]$
    // Insert $A[i]$ into the sorted subarray $A[1 : i - 1]$.
    $j = i - 1$
    while $j > 0$ and $A[j] > key$
        $A[j + 1] = A[j]$
        $j = j - 1$
    $A[j + 1] = key$
```

Insertion Sort

[7]

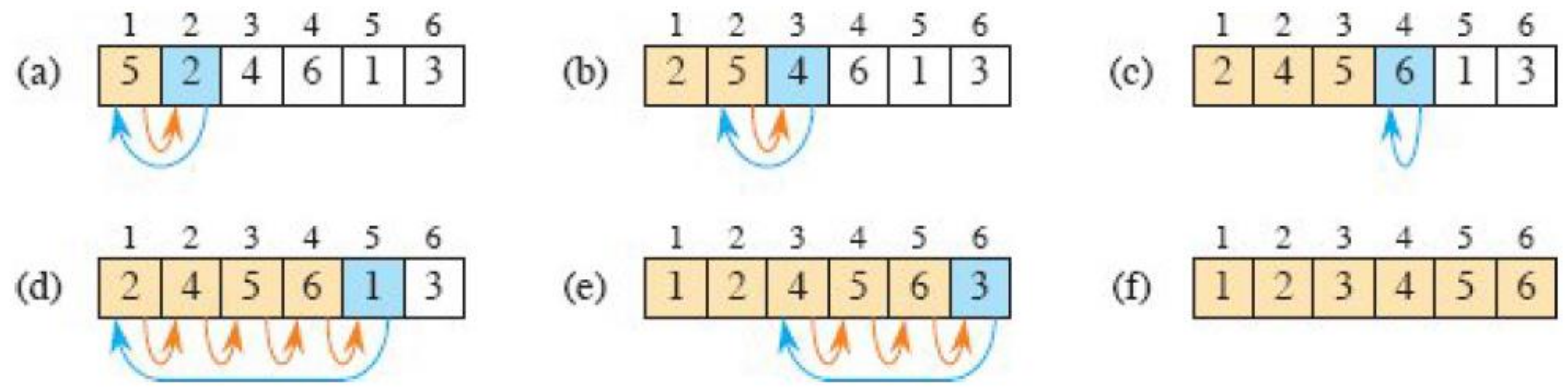


Figure 2.2 The operation of INSERTION-SORT(A, n), where A initially contains the sequence $\langle 5, 2, 4, 6, 1, 3 \rangle$ and $n = 6$. Array indices appear above the rectangles, and values stored in the array positions appear within the rectangles. **(a)–(e)** The iterations of the **for** loop of lines 1–8. In each iteration, the blue rectangle holds the key taken from $A[i]$, which is compared with the values in tan rectangles to its left in the test of line 5. Orange arrows show array values moved one position to the right in line 6, and blue arrows indicate where the key moves to in line 8. **(f)** The final sorted array.

Insertion Sort

[7]

```
INSERTION-SORT($A$, $n$)                                          cost   times
1  for $i = 2$ to $n$                                             $c_1$  $n$
2      $key = A[i]$                                               $c_2$  $n - 1$
3      // Insert $A[i]$ into the sorted subarray $A[1 : i - 1]$.  $0$    $n - 1$
4      $j = i - 1$                                                $c_4$  $n - 1$
5      while $j > 0$ and $A[j] > key$                             $c_5$  $\sum_{i=2}^{n} t_i$
6          $A[j + 1] = A[j]$                                      $c_6$  $\sum_{i=2}^{n} (t_i - 1)$
7          $j = j - 1$                                            $c_7$  $\sum_{i=2}^{n} (t_i - 1)$
8      $A[j + 1] = key$                                           $c_8$  $n - 1$
```

$$
\begin{aligned}
T(n) = {} & c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{i=2}^{n} t_i + c_6 \sum_{i=2}^{n} (t_i - 1) \\
& + c_7 \sum_{i=2}^{n} (t_i - 1) + c_8(n-1) .
\end{aligned}
$$

Insertion Sort

[7]

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

$$= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8) \;. \quad (2.2)$$

We can express this worst-case running time as $an^2 + bn + c$ for constants a, b, and c that again depend on the statement costs c_k (now, $a = c_5/2 + c_6/2 + c_7/2$, $b = c_1 + c_2 + c_4 + c_5/2 - c_6/2 - c_7/2 + c_8$, and $c = -(c_2 + c_4 + c_5 + c_8)$). The running time is thus a ***quadratic function*** of n. $\Theta(n^2)$

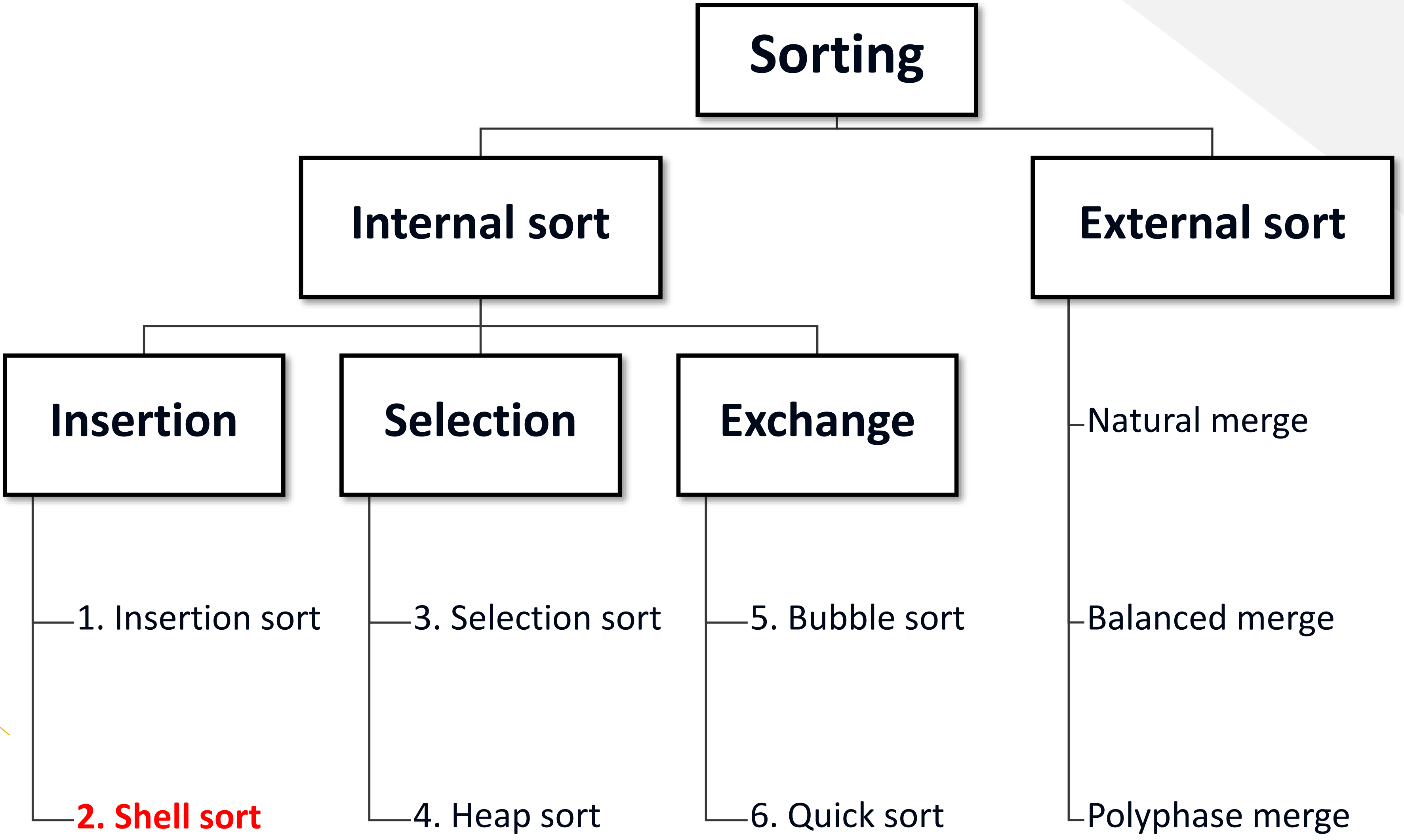


Fig 15-1 Sort Classification (Cont.)

Shell Sort

- Donald L. Shell improved version of the insertion sort in which the diminishing partitions are used to sort the data.
- Given a list of N elements, the list is divided into K segments, where K is known as the increment, each segment contains N/K or less elements.

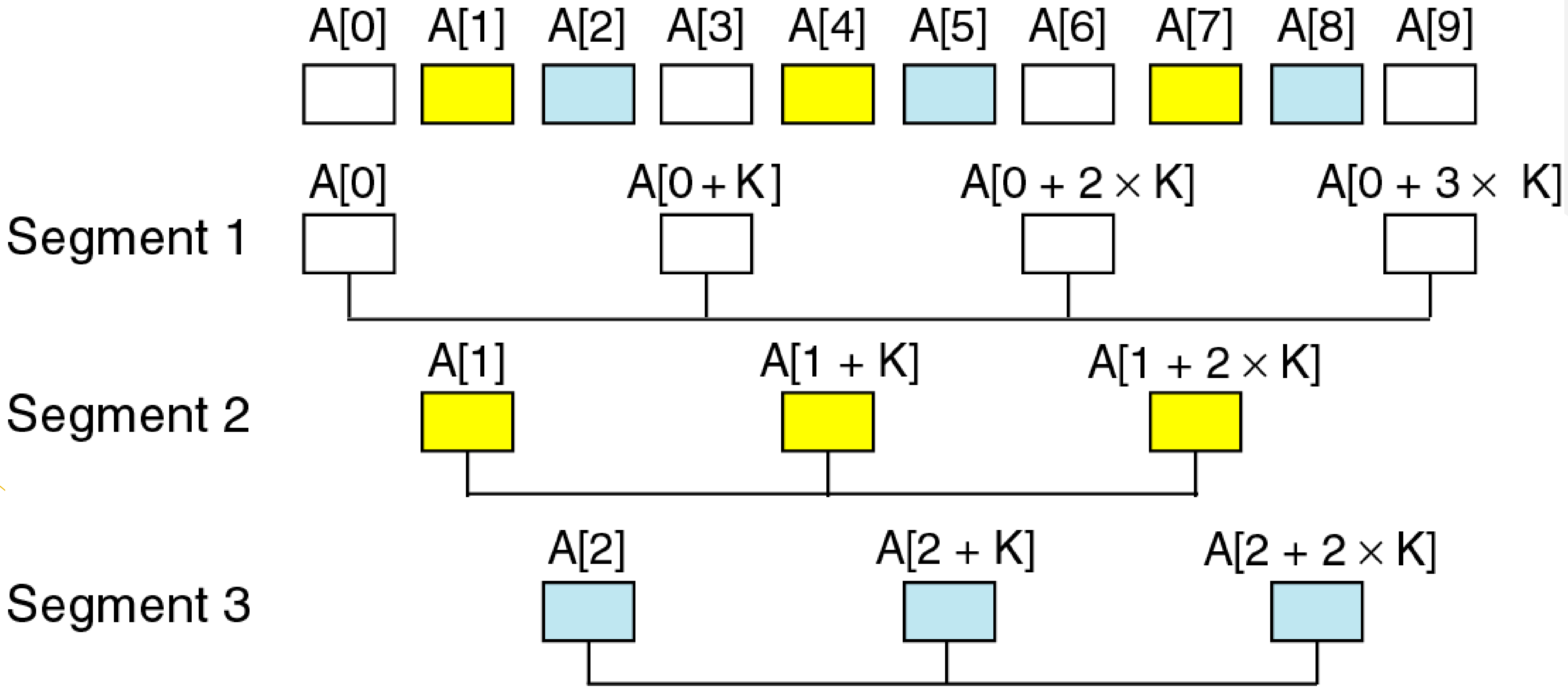


Fig 15-4 Shell sort segmentation **[3]**

Shell Sort

- There is not an increment size that is best for all situations
- Method to eliminate completely an element being in more than one list: **[3]**
    1. Use prime numbers to be an increment.
    2. Setting the increment to half the list size and dividing by 2 each pass.
    3. Knuth suggests, however, that you should not start with an increment greater than one-third of the list size.
    4. The increment be a power of two minus one or fibonacci series.

Shell Sort

- Most use the simple series, setting the increment to half the list size and dividing by two each pass.
- This approach would be to add 1 whenever the increment is odd.
- Therefore, if the objective is to obtain the most efficiency sort, the solution is to use quick sort rather than trying to optimize the increment size in the shell sort. **[3]**

Shell Sort

- Steps of work:
  1. Read a key in sequence
  2. Compare the key with candidate keys form every group.
  3. Repeat step 1 until there is no more key to be sorted.

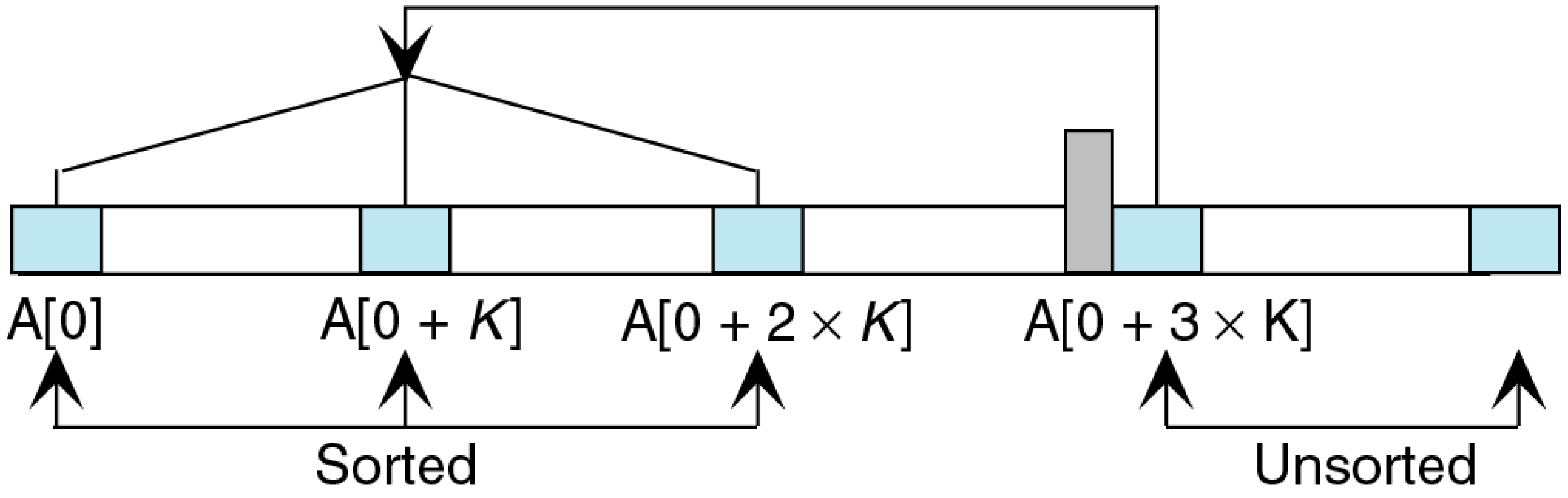
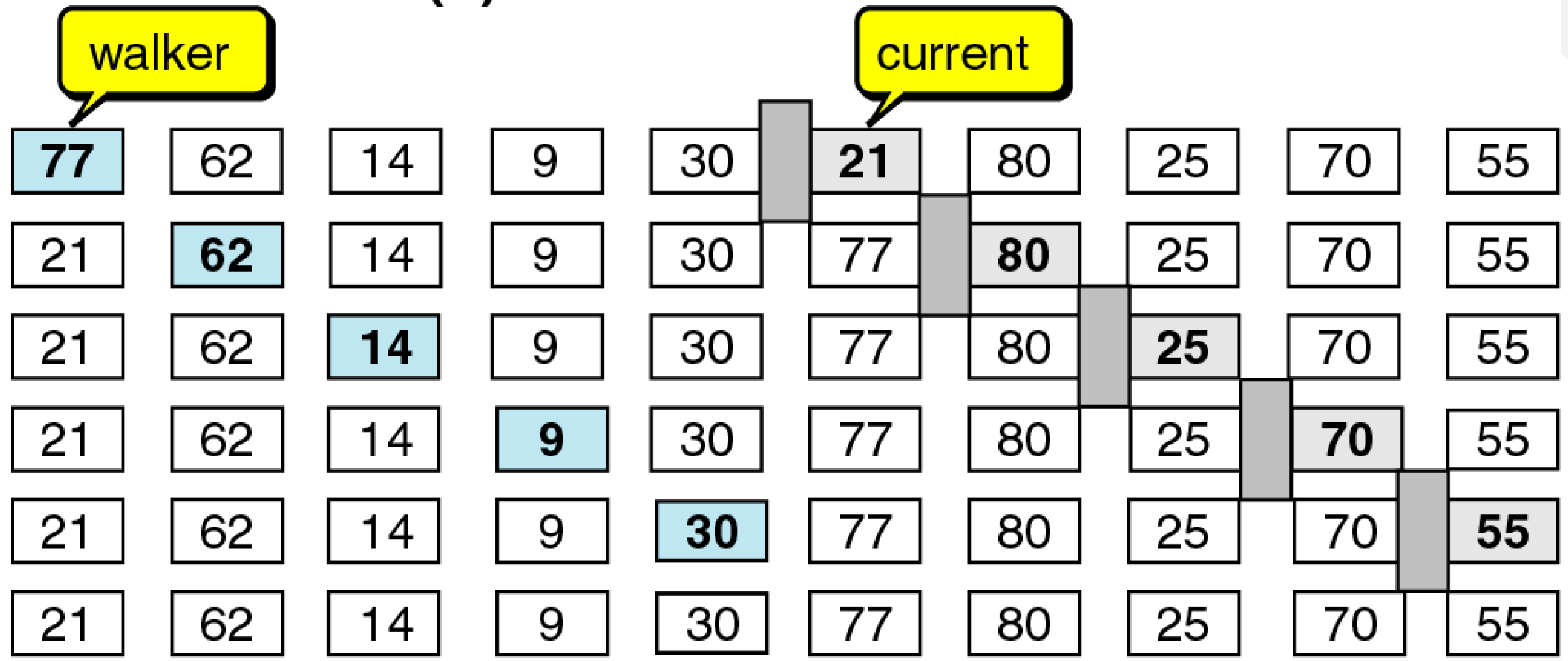


Fig 15-5 Shell sort steps of work [3]

Tutorial 3: Shell Sort Illustration [3]

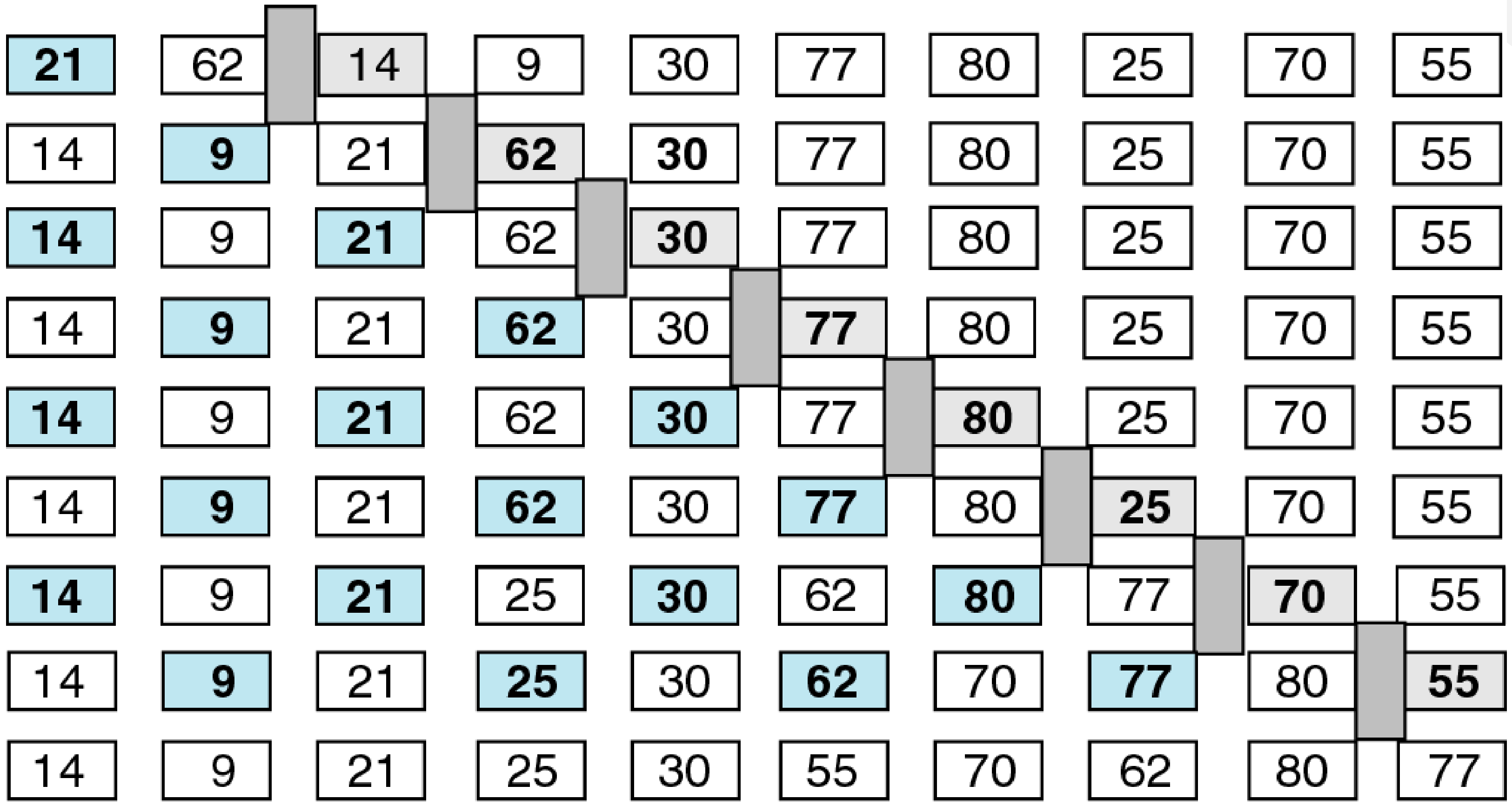
(a) First increment: $K = 5$



77	62	14	9	30	21	80	25	70	55
21	62	14	9	30	77	80	25	70	55
21	62	14	9	30	77	80	25	70	55
21	62	14	9	30	77	80	25	70	55
21	62	14	9	30	77	80	25	70	55
21	62	14	9	30	77	80	25	70	55

Tutorial 3: Shell Sort Illustration [3] (Cont.)

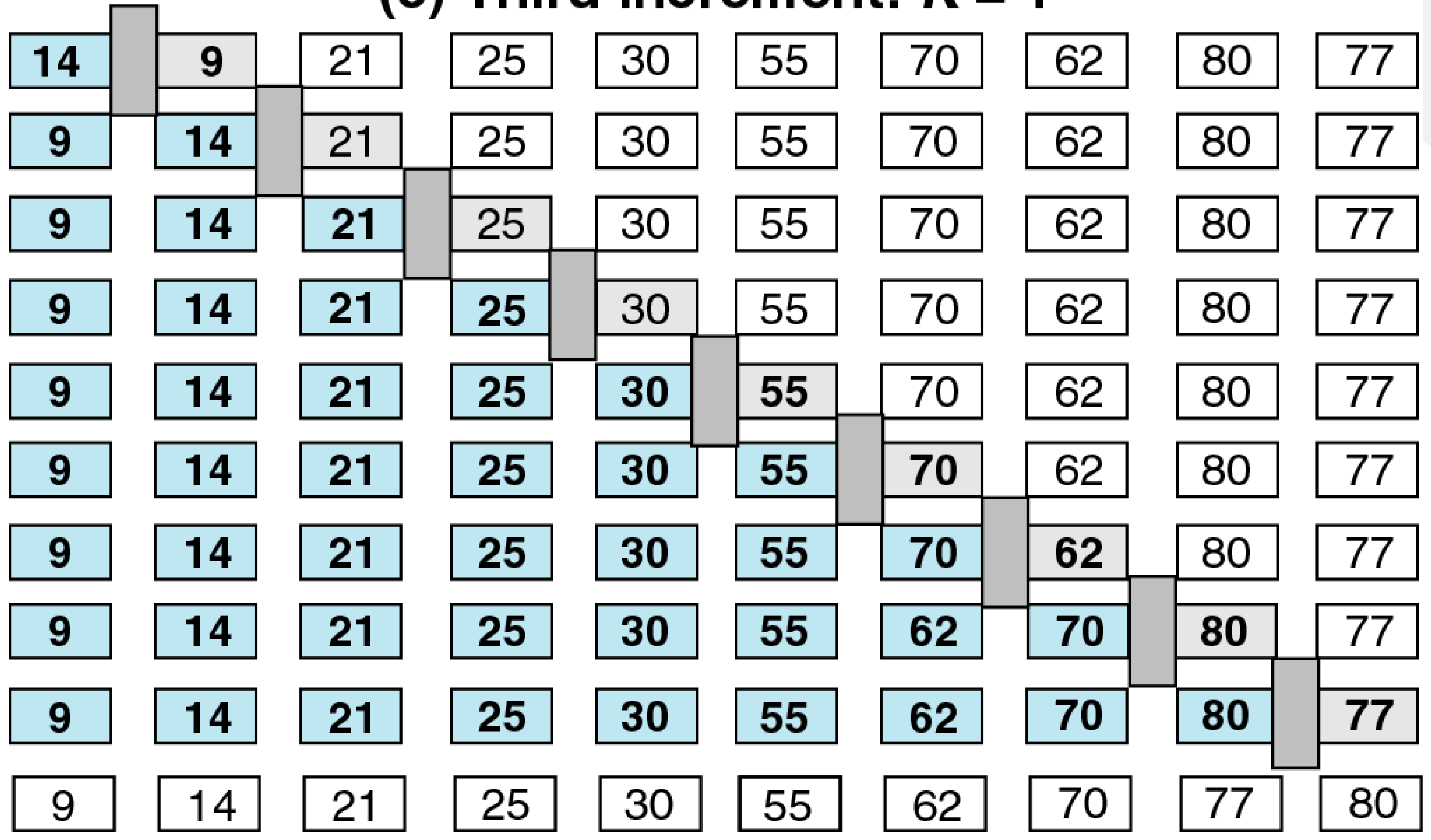
(b) Second increment: *K* = 2



21	62	14	9	30	77	80	25	70	55
14	9	21	62	30	77	80	25	70	55
14	9	21	62	30	77	80	25	70	55
14	9	21	62	30	77	80	25	70	55
14	9	21	62	30	77	80	25	70	55
14	9	21	62	30	77	80	25	70	55
14	9	21	25	30	62	80	77	70	55
14	9	21	25	30	62	70	77	80	55
14	9	21	25	30	55	70	62	80	77

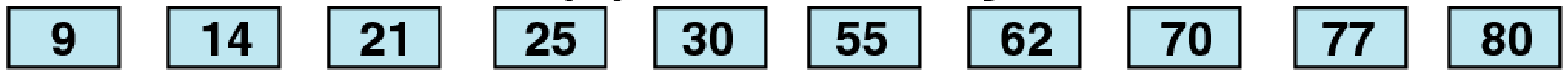
Tutorial 3: Shell Sort Illustration [3] (Cont.)

(c) Third increment: $K = 1$



14	9	21	25	30	55	70	62	80	77
9	14	21	25	30	55	70	62	80	77
9	14	21	25	30	55	70	62	80	77
9	14	21	25	30	55	70	62	80	77
9	14	21	25	30	55	70	62	80	77
9	14	21	25	30	55	70	62	80	77
9	14	21	25	30	55	70	62	80	77
9	14	21	25	30	55	62	70	80	77
9	14	21	25	30	55	62	70	80	77
9	14	21	25	30	55	62	70	77	80

(d) Sorted array



9	14	21	25	30	55	62	70	77	80

Shell Sort

- The total number of iterations for the outer loop and the first inner loop is [3]

  log2 n * [(n-n/2)+(n-n/4)+(n -n/2)+…+1] = n $\log_2$ n

- The shell sort efficiency is O(n $\log_2$ n)

- Knuth tells us that the sort effort for the shell sort cannot be mathematically derived and approximately estimates 15$n^{1.25}$
- Therefore, reducing Knuth's analysis to a Big-O notation O($n^{1.25}$).

Shell Sort

- Heuristic studies indicate that the straight insertion sort is more efficient than the shell sort for small lists.

Number of elements	Number of loops	
	Straight Insertion Sort	Shell Sort
25	625	55
100	10,000	316
500	250,000	2,364
1000	1,000,000	5,623
2000	4,000,000	13,374

Table 15-1 Comparison between Insertion sort and Shell sort performances

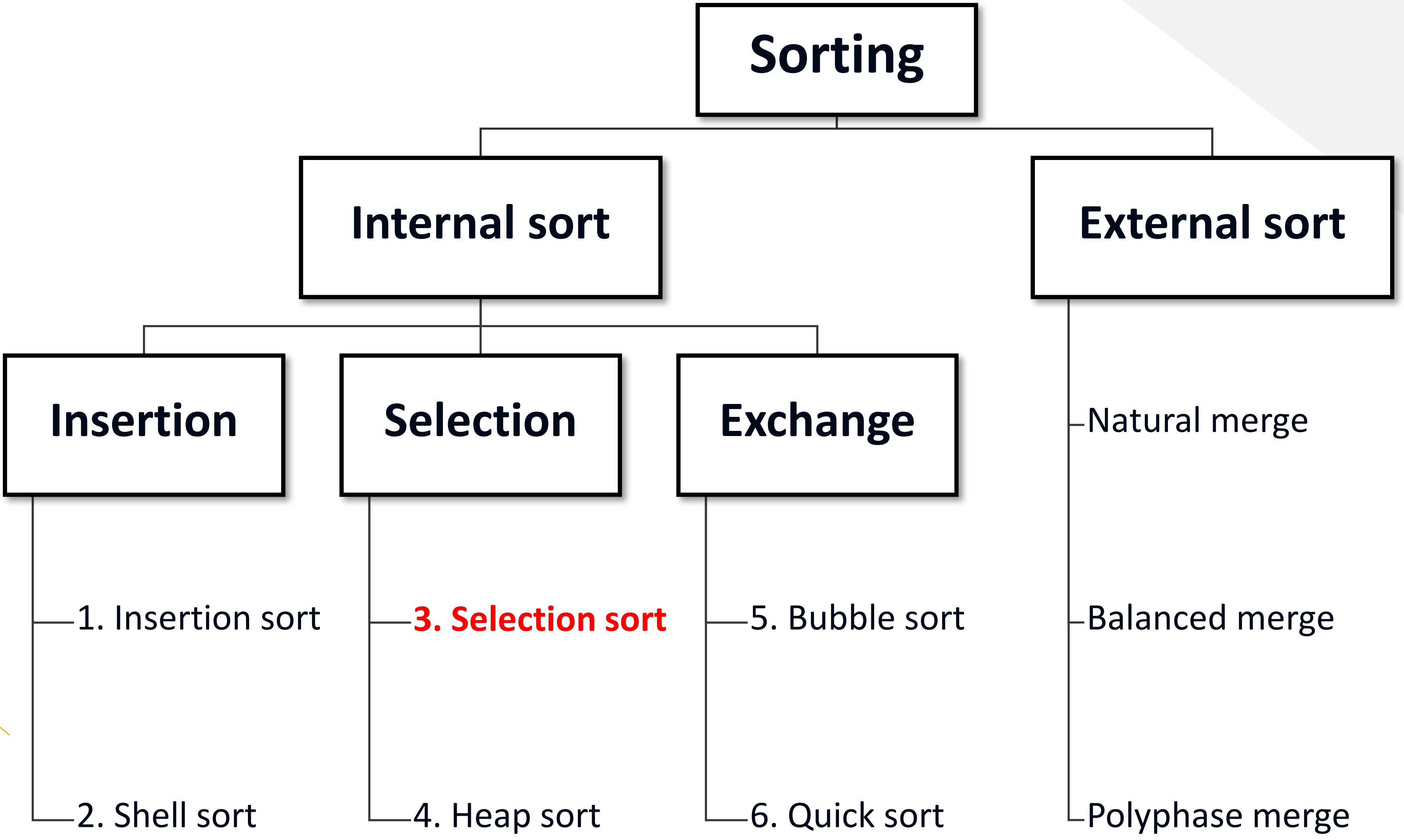


Fig 15-1 Sort Classification (Cont.)

Selection Sort

- Its concept relies on a fashion works similar to what a human use to sort a list of values.

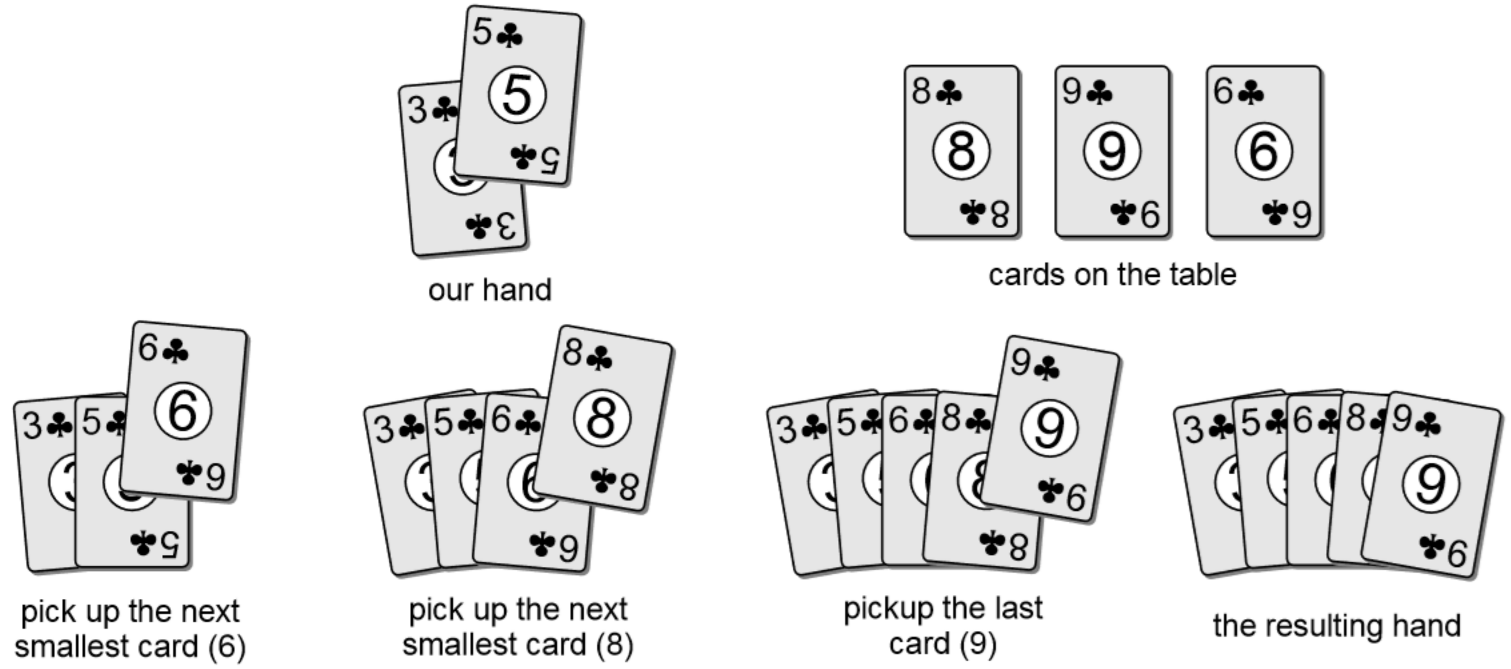

our hand
cards on the table
pick up the next smallest card (6)
pick up the next smallest card (8)
pickup the last card (9)
the resulting hand

Fig 15-6 Picking up card with selection sort concept [2]

Selection Sort

- Two selection sorts:
  1. Selection sort (Straight selection sort)
  2. Heap sort
- It requires a search to select the smallest item and place it in a sorted list.

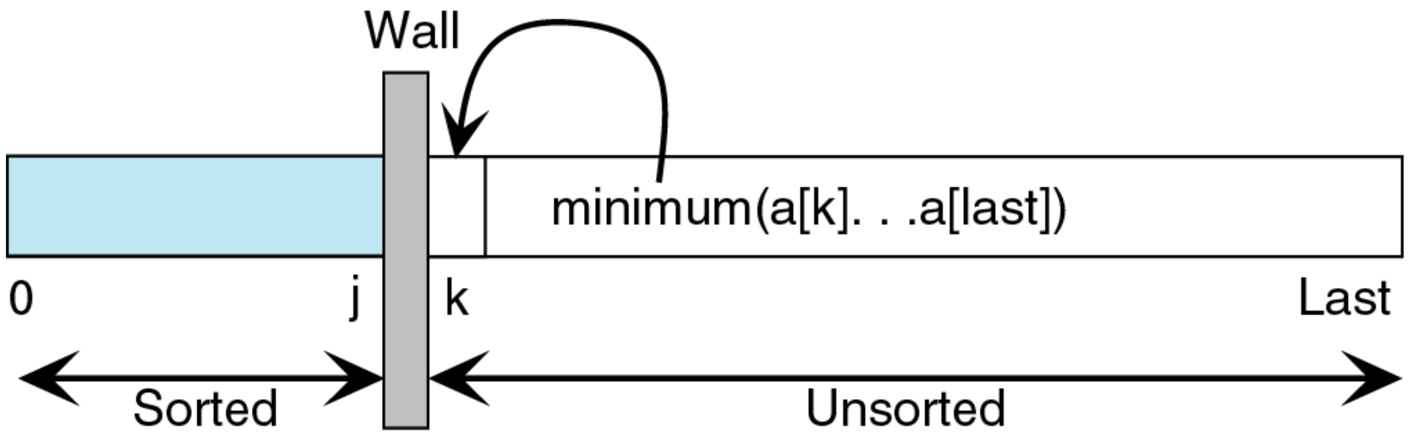


Fig 15-7 Selection sort concept [3]

Selection Sort

- Step of works:
  1. Select the smallest element from the unsorted sublist
  2. Swap it with the element at the beginning of the unsorted data
  3. Repeat to step 1 until there is no key to be sorted

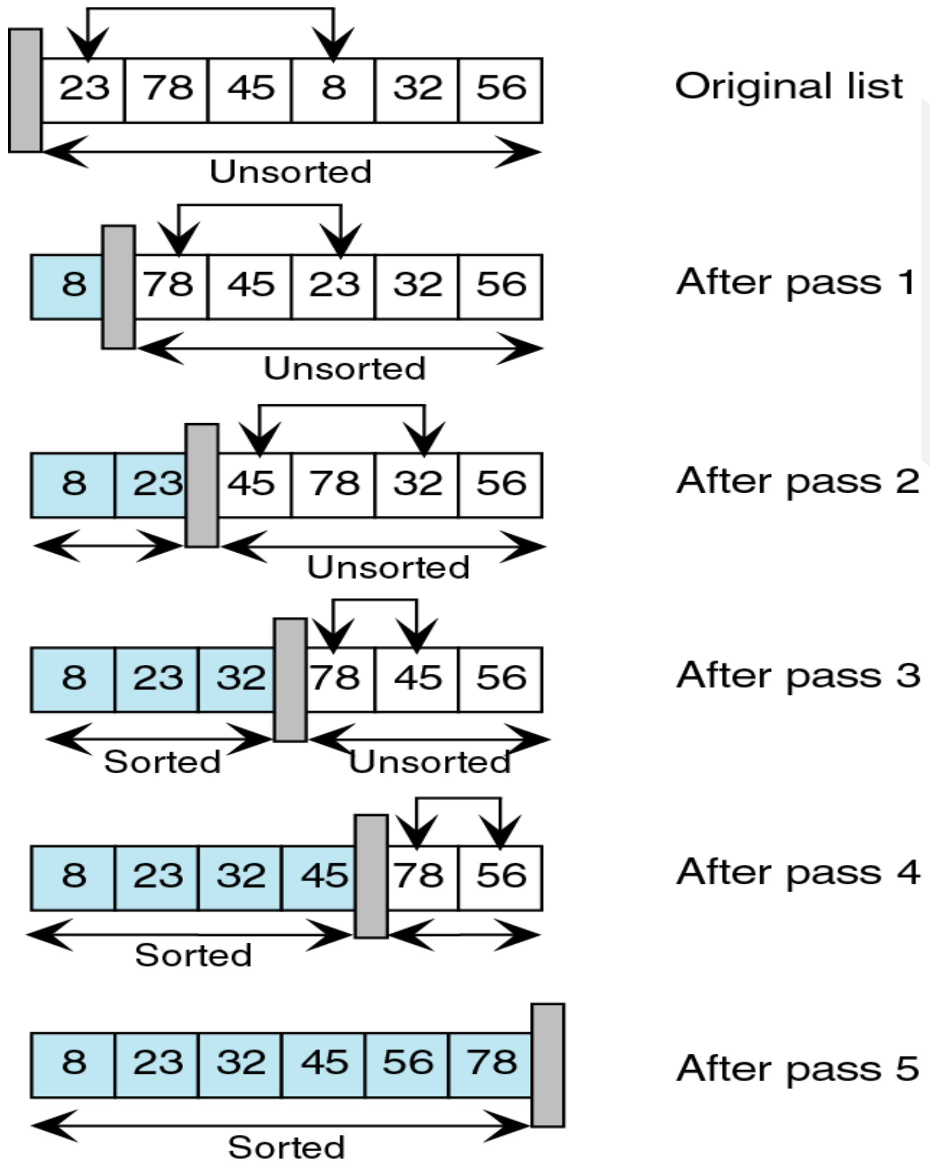
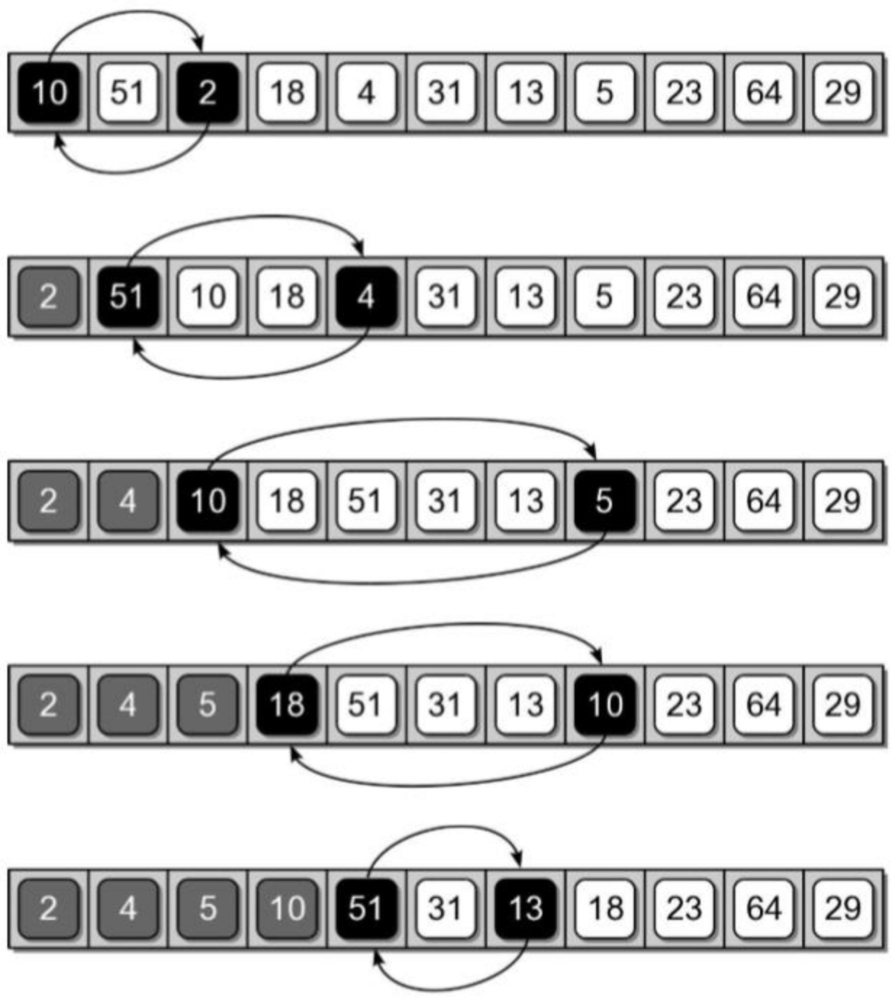
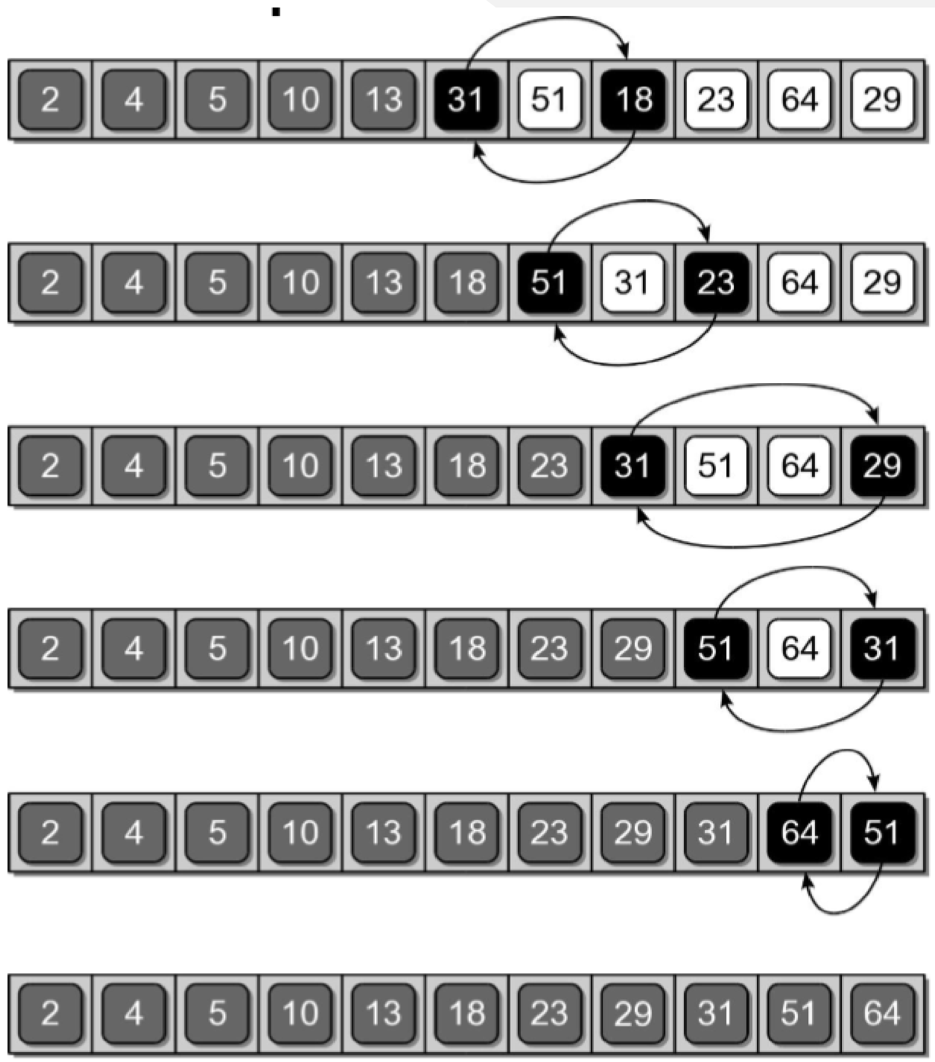


Fig 15-6 Selection sort steps of work [3]

Tutorial 4: Selection Sort Illustration [2]





Tutorial 5: Selection Sort Implementation [2]

```python
# Sorts a sequence in ascending order using the selection sort algorithm.
def selectionSort( theSeq ):
  n = len( theSeq )
  for i in range( n - 1 ):
     # Assume the ith element is the smallest.
    smallNdx = i
     # Determine if any other element contains a smaller value.
    for j in range( i + 1, n ):
      if theSeq[j] < theSeq[smallNdx] :
        smallNdx = j

    # Swap the ith value and smallNdx value only if the smallest value is
    # not already in its proper position. Some implementations omit testing
    # the condition and always swap the two values.
   if smallNdx != i :
      tmp = theSeq[i]
      theSeq[i] = theSeq[smallNdx]
      theSeq[smallNdx] = tmp
```

Selection Sort

- The selection sort which makes n-1 passes over the list to reposition n-1 values.
- The selection sort efficiency is $O(n^2)$
- The difference between the selection and bubble sort is that the selection sort reduces the number of swaps required to sort the list O(n). [2]

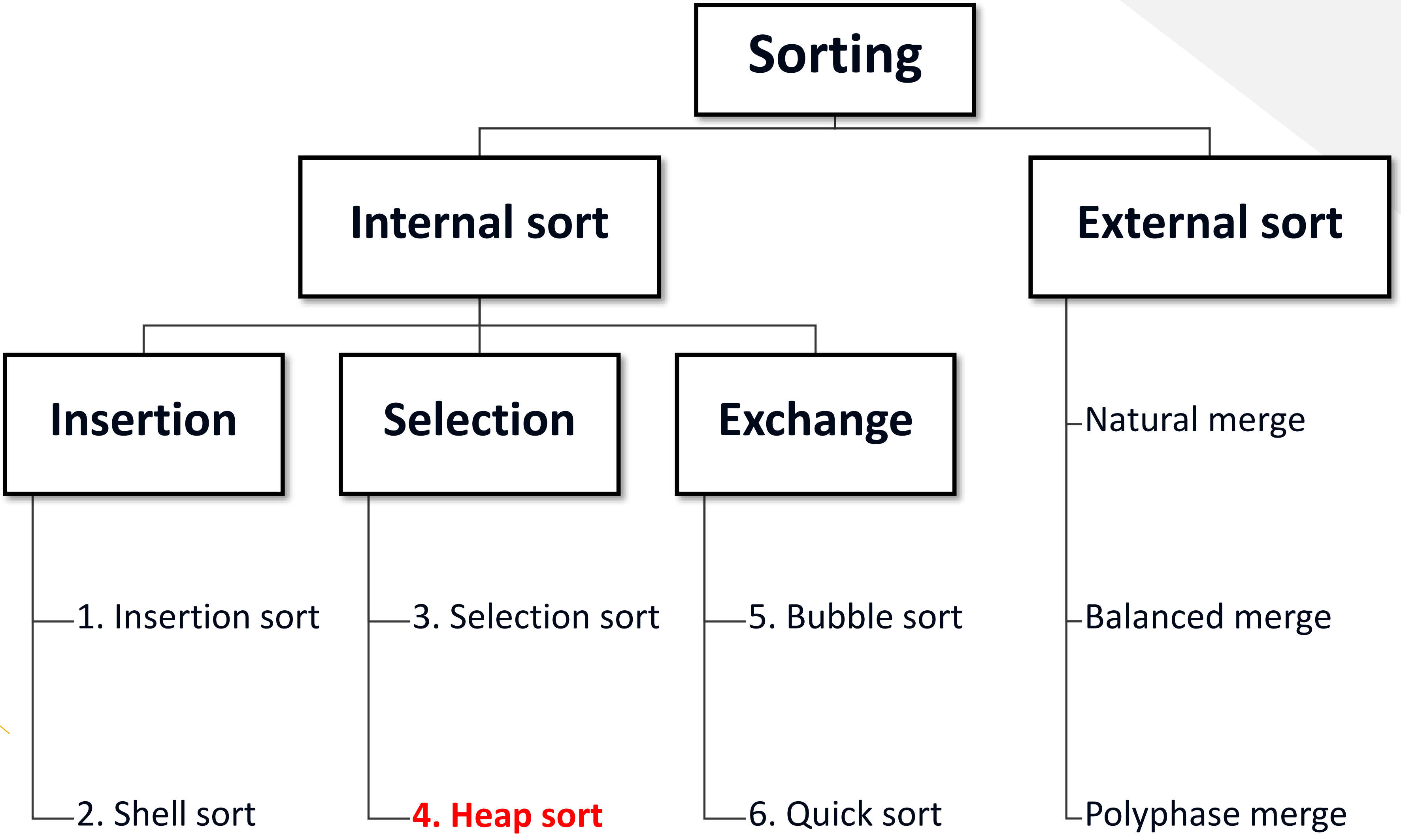
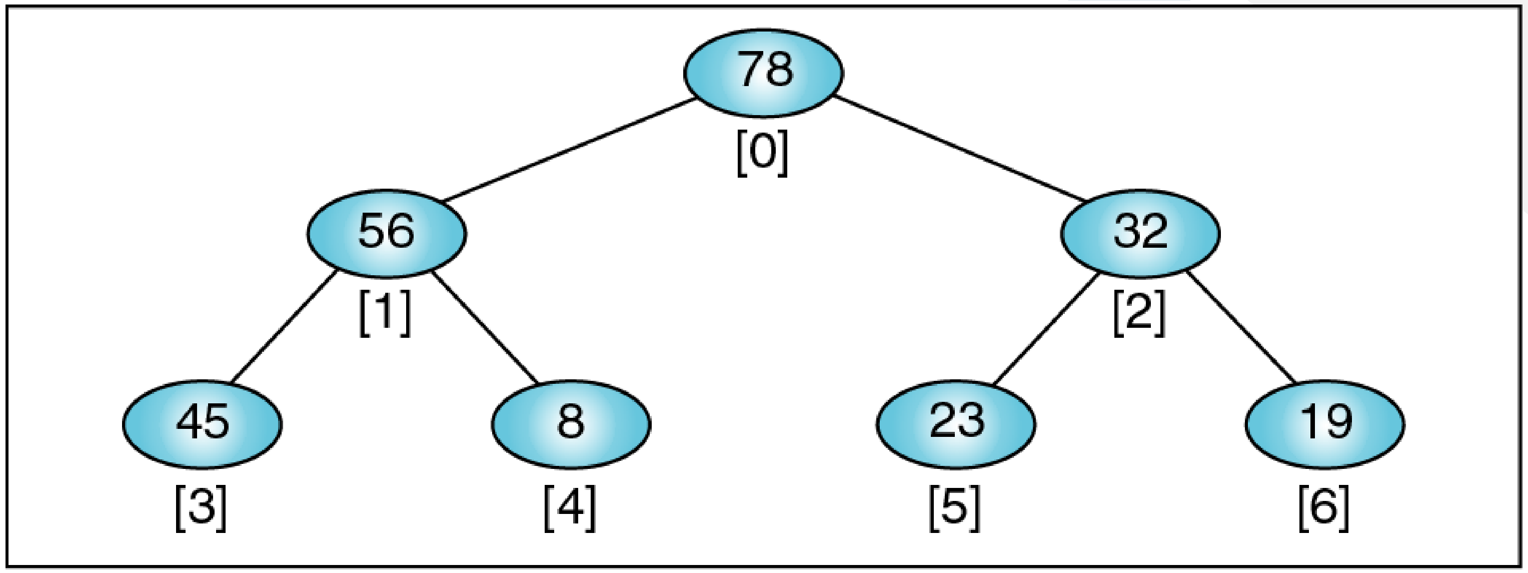


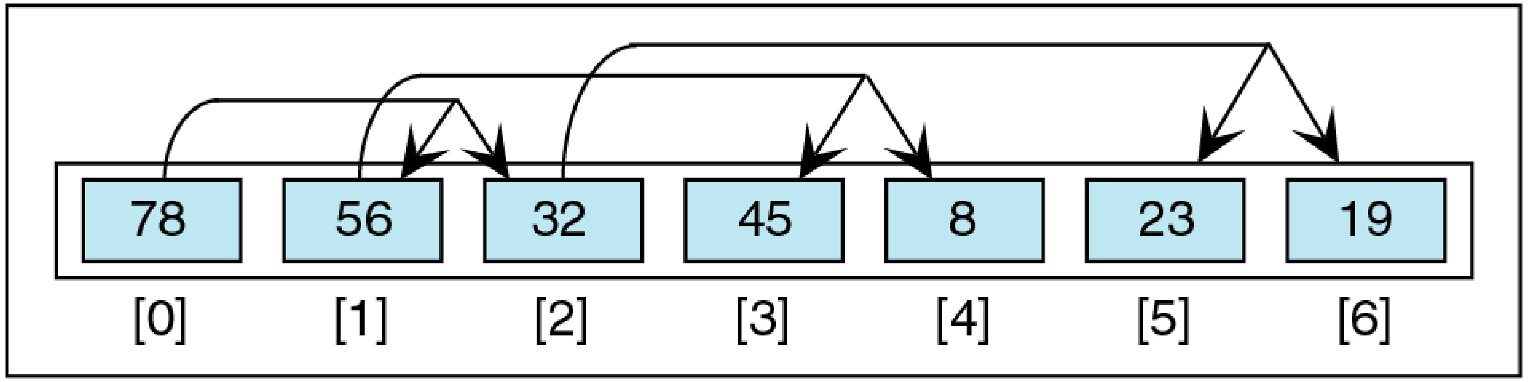
Fig 15-1 Sort Classification (Cont.)

Heap Sort

- Heap sort algorithm is an improved version of straight selection sort in which the largest (or smallest) element at the root is selected and exchange with the last element in the unsorted list.
- Heap is a tree structure in which the root contains the largest (or smallest) element in the tree.



(a) A heap in its logical form



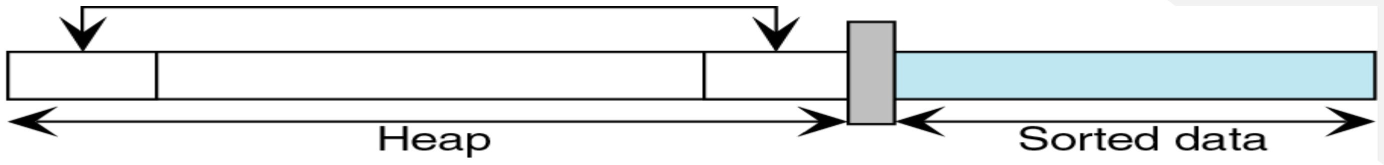
(b) A heap in an array

Fig 15-7 Heap sort concept [3]

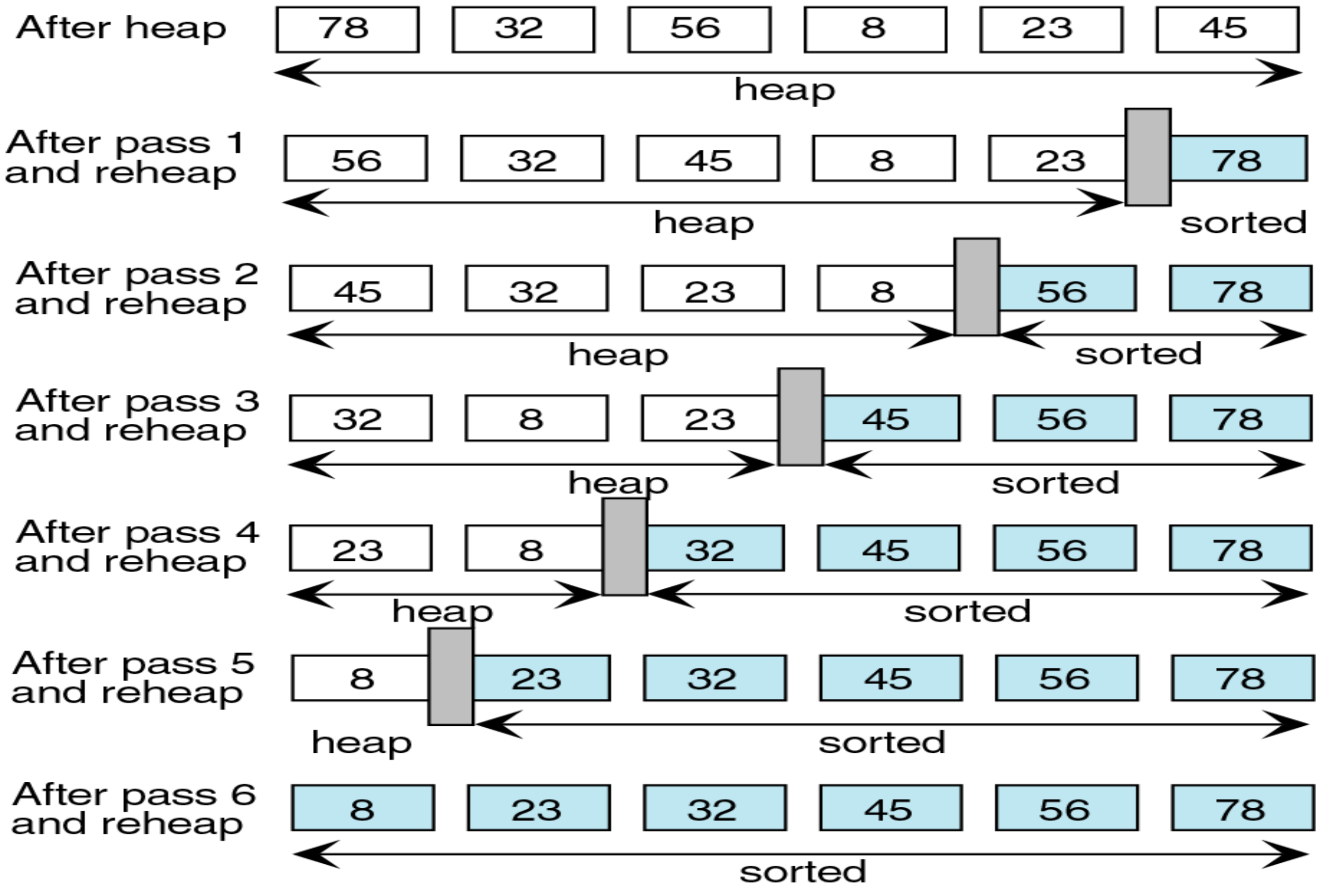
Selection Sort

- Step of works:
  1. Build the max heap if you want to sort in ascending order and the min heap if descending order is required.
  2. Delete heap and exchange the delete key to the last position which has just available from removing that key.
  3. Delete heap until heap is empty.

Tutorial 5: Heap Sort Illustration [3]



(a) Heap sort exchange process



(b) Heap sort process

Heap Sort

- The heap sort efficiency is $O(n \log_2 n)$

n	Number of loops		
	Straight Insertion Sort Straight Selection Sort	Shell Sort	Heap sort
25	625	55	116
100	10,000	316	664
500	250,000	2,364	4,482
1000	1,000,000	5,623	9,965
2000	4,000,000	13,374	10,965

Table 15-2 Comparison performances

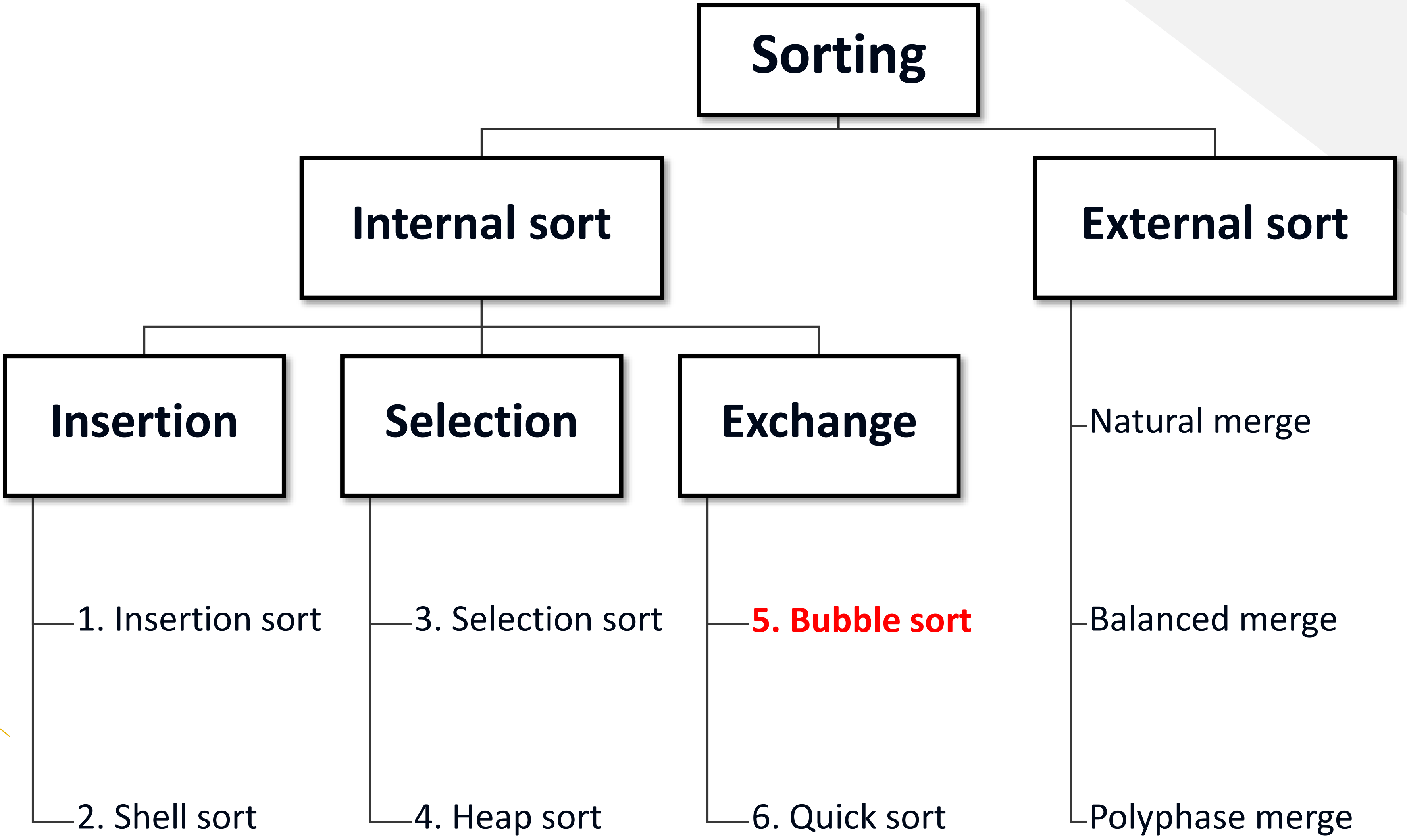


Fig 15-1 Sort Classification (Cont.)

Bubble Sort

- Elements that are out of order are exchanged until the entire list is sorted.
- The smallest element is bubble from the unsorted sublist and moved to the sorted sublist.

- Two exchange sorts are:
  1. The bubble sort
  2. The most efficient general purpose sort, quick sort

Bubble Sort

- Steps of work:

It arrange the items by iterating over the list and larger key bubbles to the end of list.

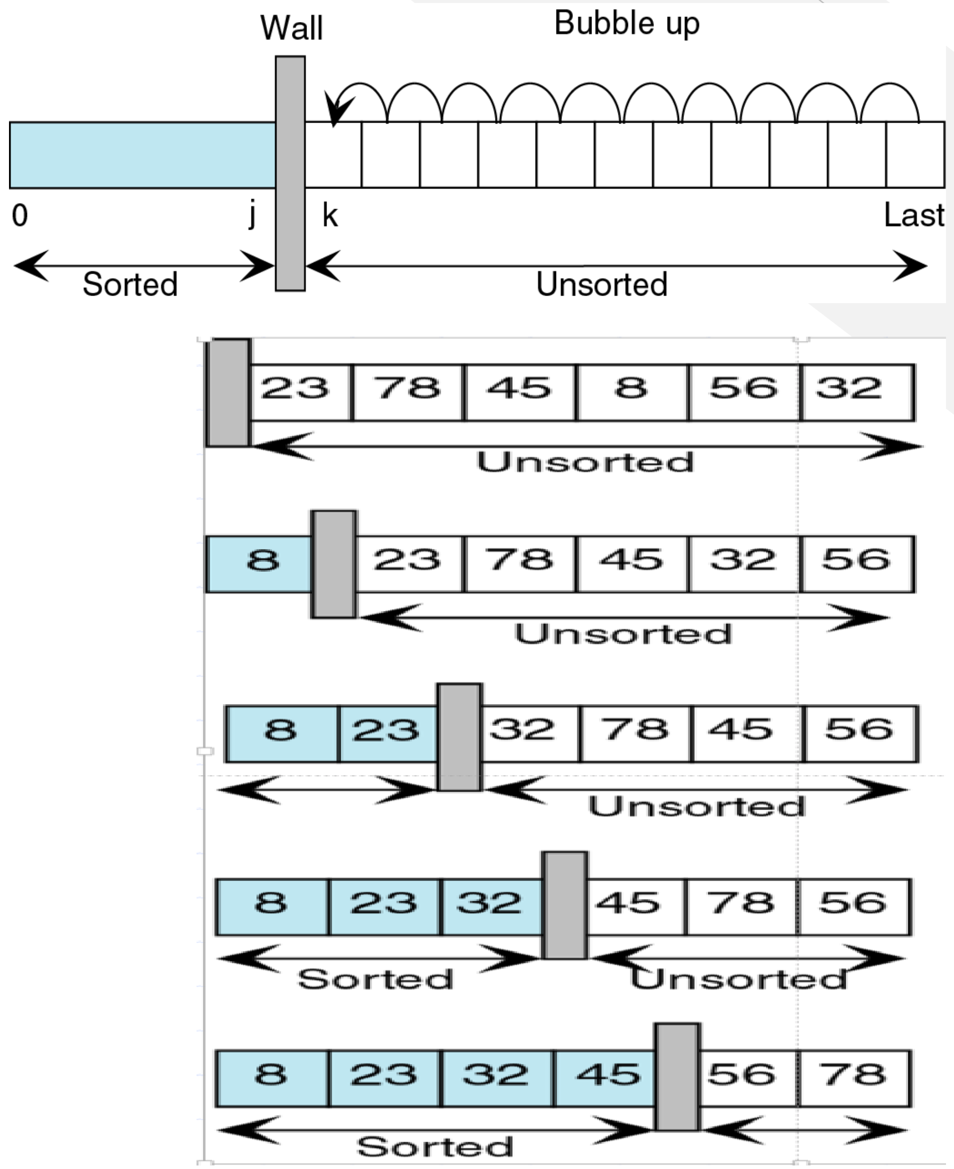
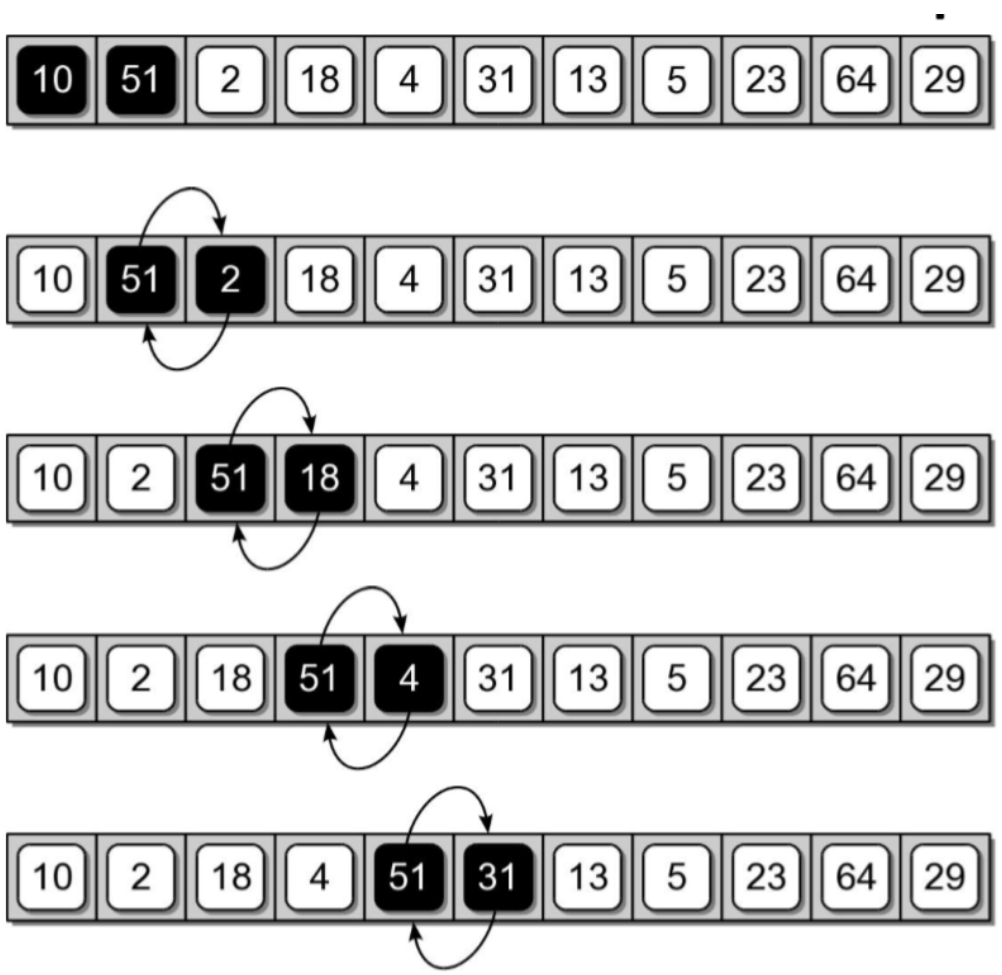
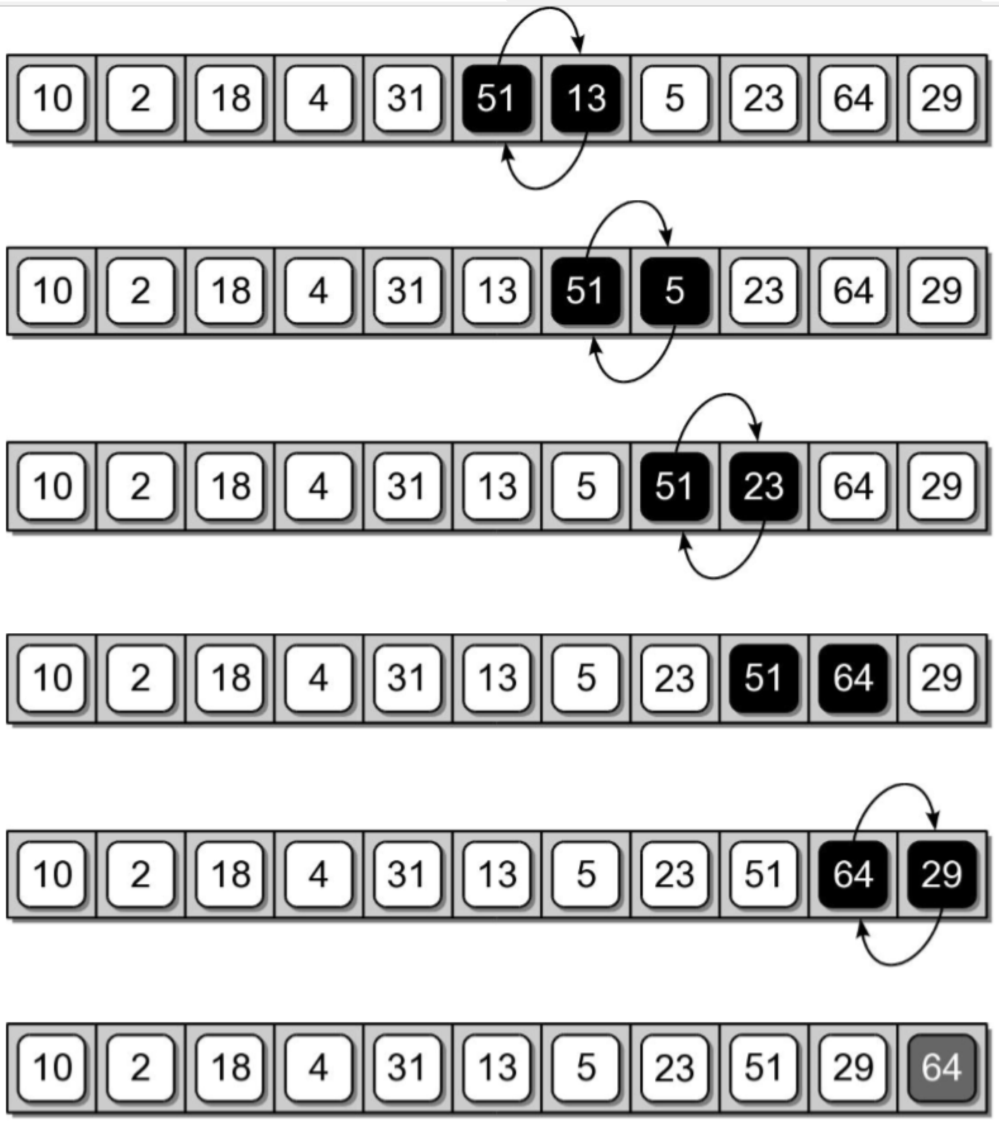


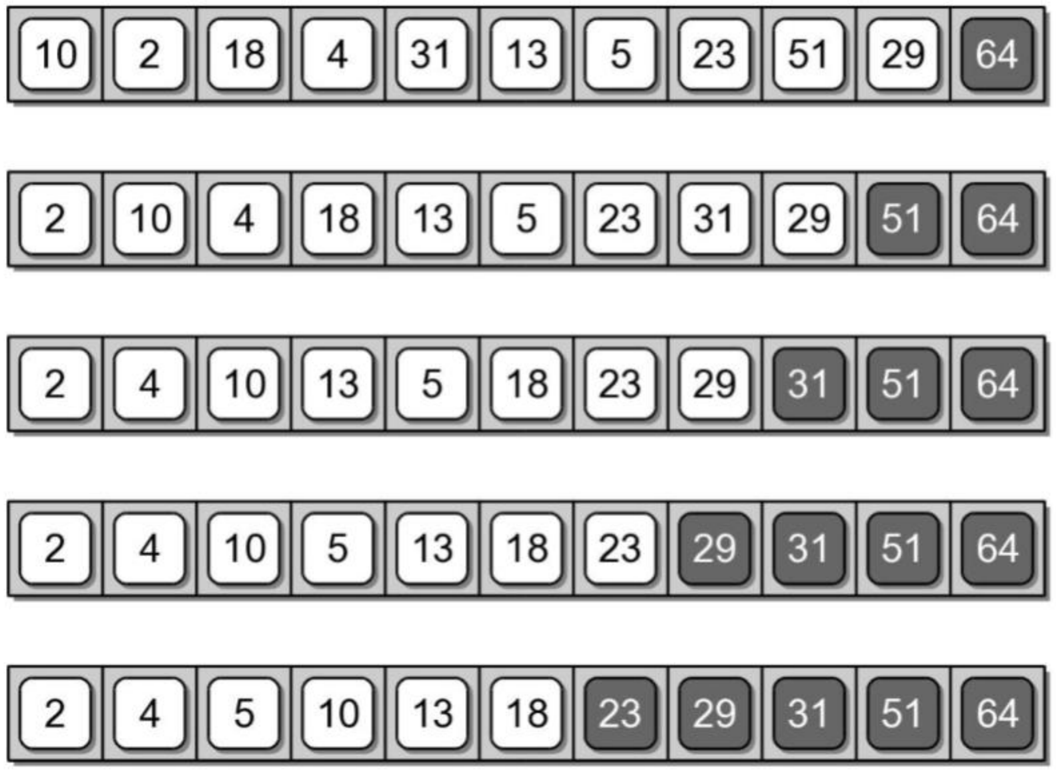
Fig 15-8 Bubble sort steps of work [3]

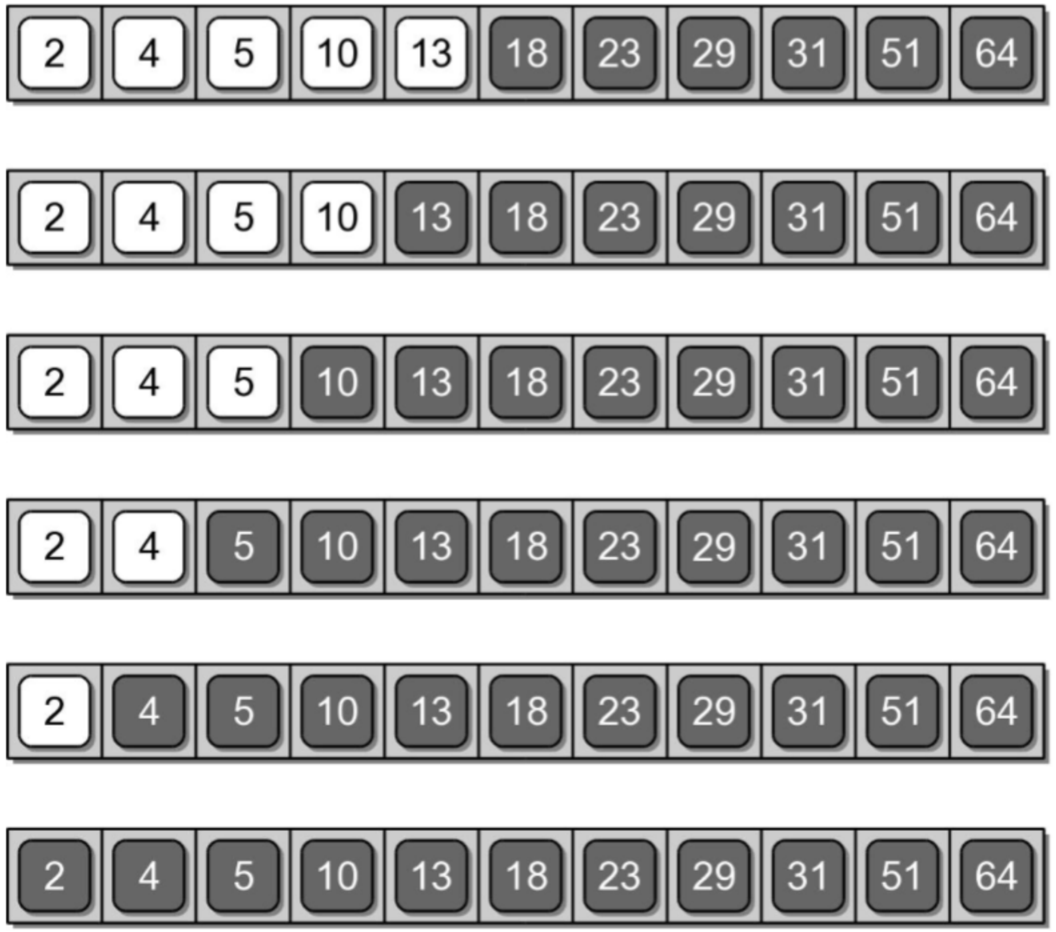
Tutorial 6: Bubble Sort Illustration [2]





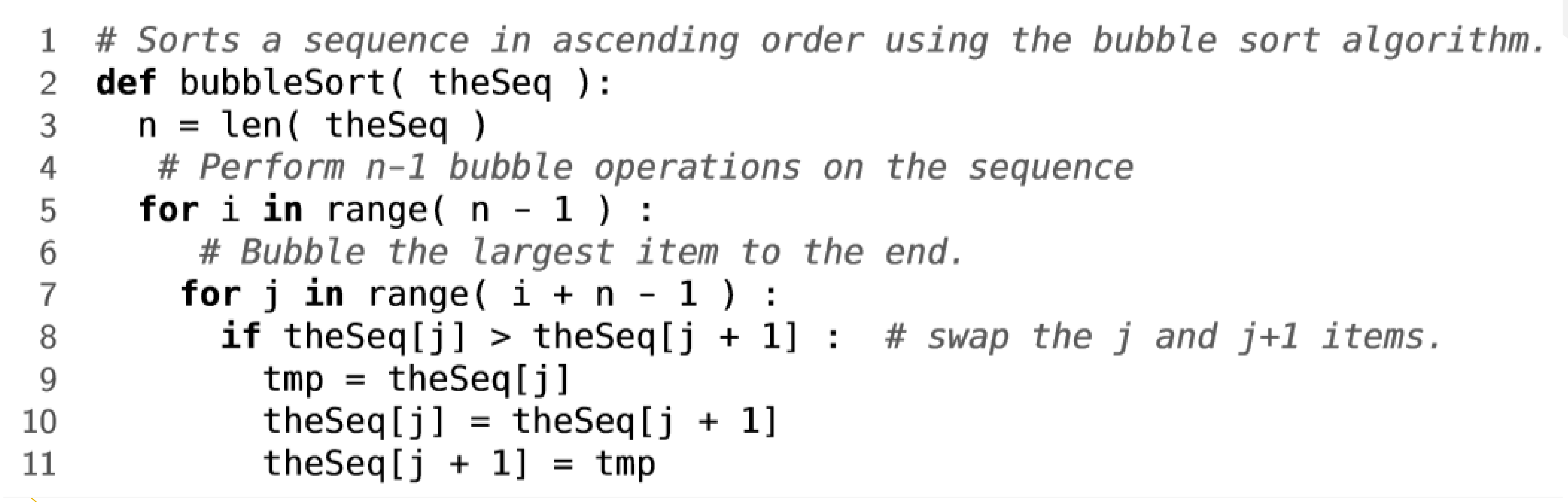
Tutorial 6: Bubble Sort Illustration [2] (Cont.)





Tutorial 7: Bubble Sort Implementation [2]

1.5



```python
# Sorts a sequence in ascending order using the bubble sort algorithm.
def bubbleSort( theSeq ):
  n = len( theSeq )
   # Perform n-1 bubble operations on the sequence
  for i in range( n - 1 ) :
     # Bubble the largest item to the end.
    for j in range( i + n - 1 ) :
      if theSeq[j] > theSeq[j + 1] :  # swap the j and j+1 items.
        tmp = theSeq[j]
        theSeq[j] = theSeq[j + 1]
        theSeq[j + 1] = tmp
```

Bubble Sort

- The total number of iterations for the inner loop will be the sum of the first n-1 keys which makes the selection sort efficiency is $O(n^2)$.

- It always perform n^2 iterations of the inner loops which is the one of the most inefficient sorting algorithm. [2]

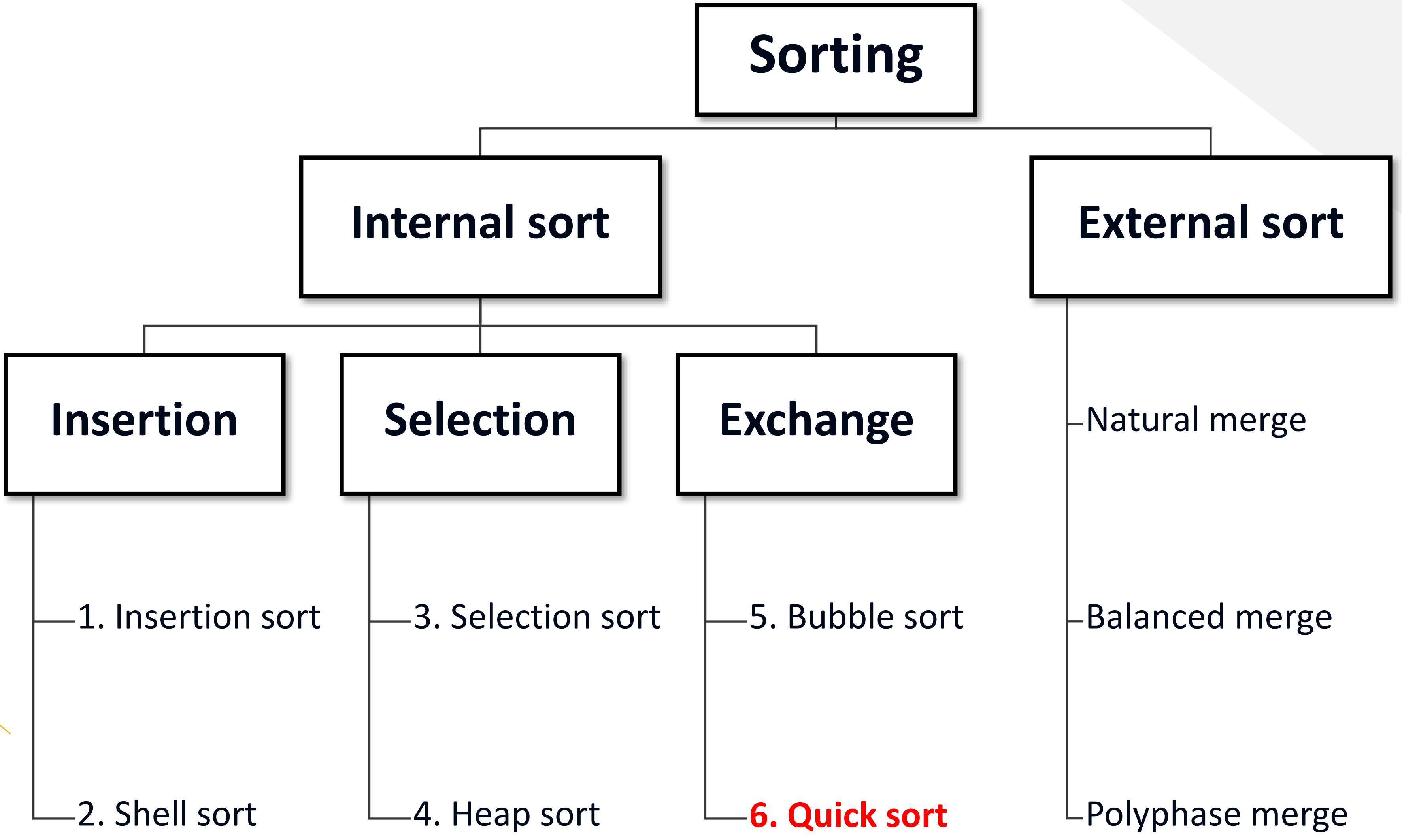


Fig 15-1 Sort Classification (Cont.)

Quick Sort

- Quick sort is an exchange sort, developed by C.A.R. Hoare in 1962.
- It is more efficient than the bubble sort because a typical exchange involves elements that are far apart so that fewer exchanges are required to correctly position an element. [3]

Quick Sort

- Each iteration of the quick sort selects an element, known as pivot and divides the list into three groups:
  1. A partition of elements whose keys are less than pivot’s key,
  2. The pivot element that is placed in its ultimately correct location in the list, and
  3. A partition of elements greater or equal t pivot’s key.

Quick Sort

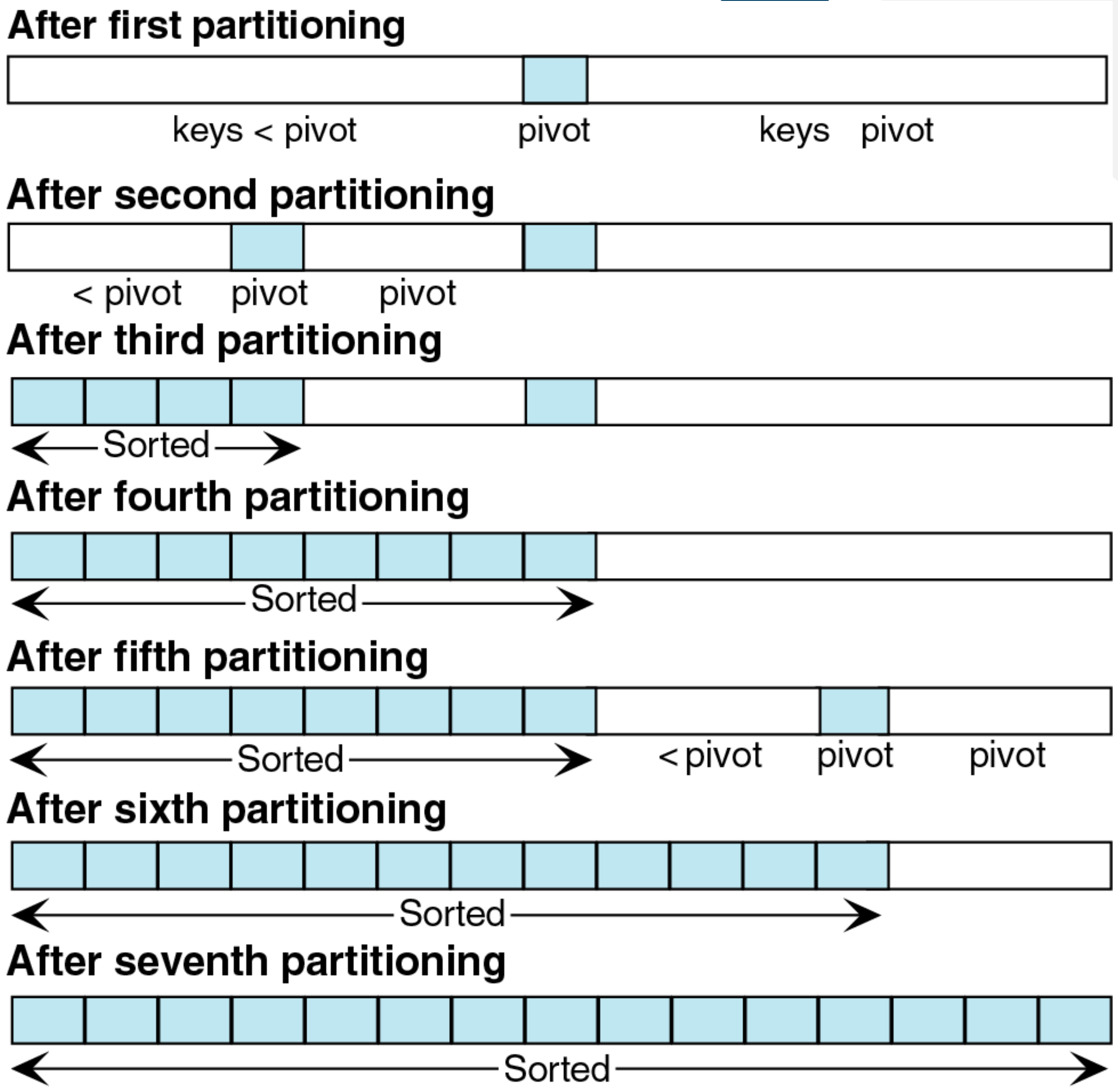


Fig 15-9 Quick sort partition [3]

Quick Sort

- Pivot key selection:
    1. Hoare’s original algorithm selected the pivot key as the first element in the list
    2. R. C. Singleton improved the sort by selecting the pivot key as the median value of three elements: left, right an d an element in the middle of the list

Quick Sort

- Steps of work:
  1. It divided sorted list into subsequences
  2. Recur to sort each subsequence
  3. Combine the sorted subsequences by a simple concatenation

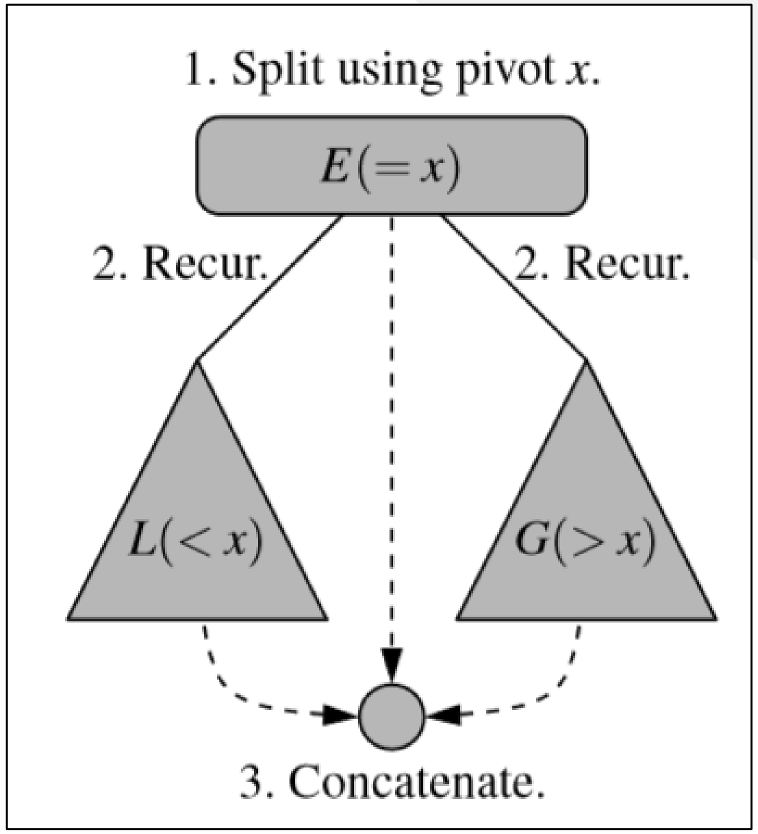


Fig 15-10 Quick sort concept [1]

Quick Sort

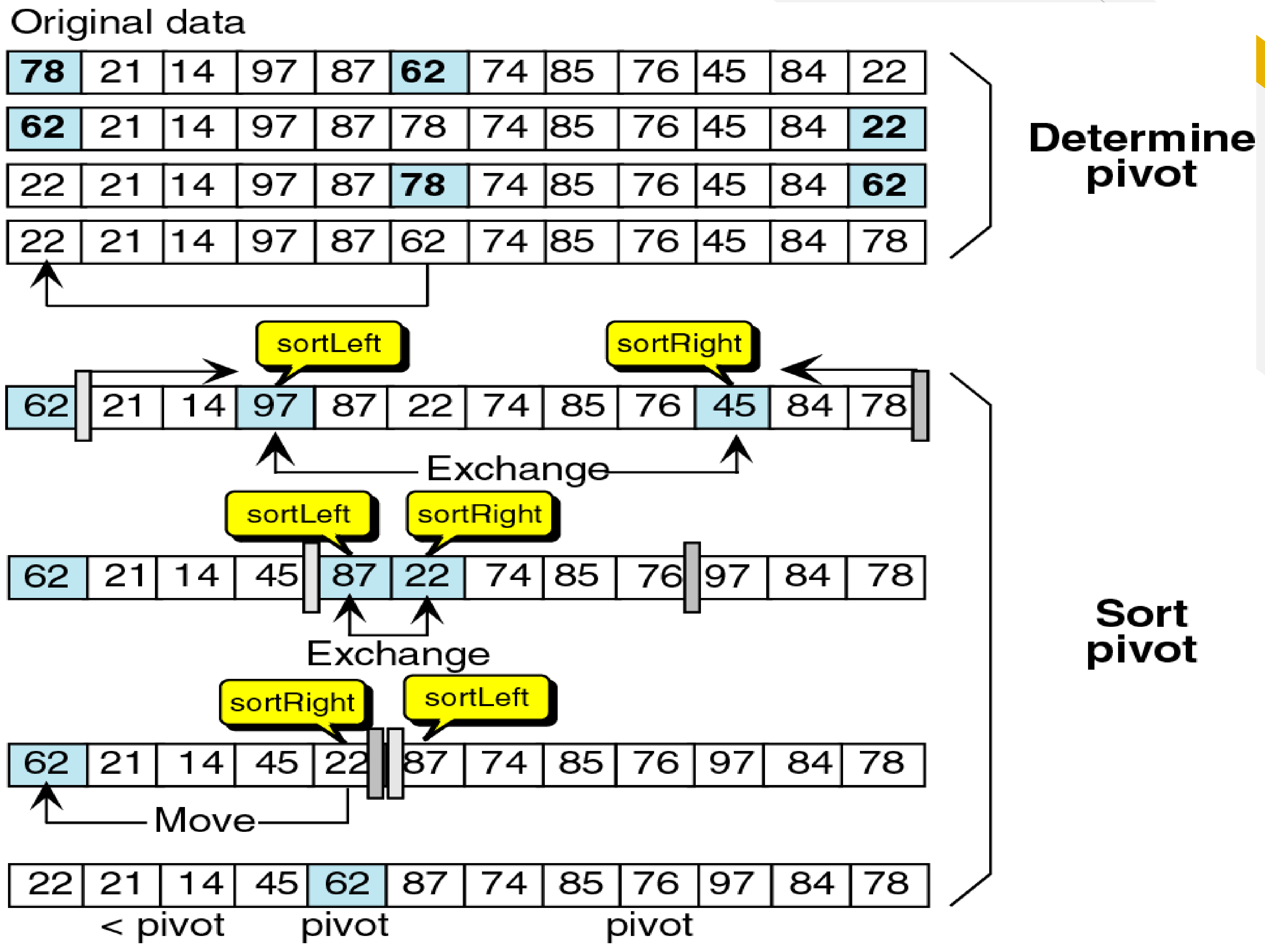


Fig 15-11 Quick sort steps of work [3]

Quick Sort

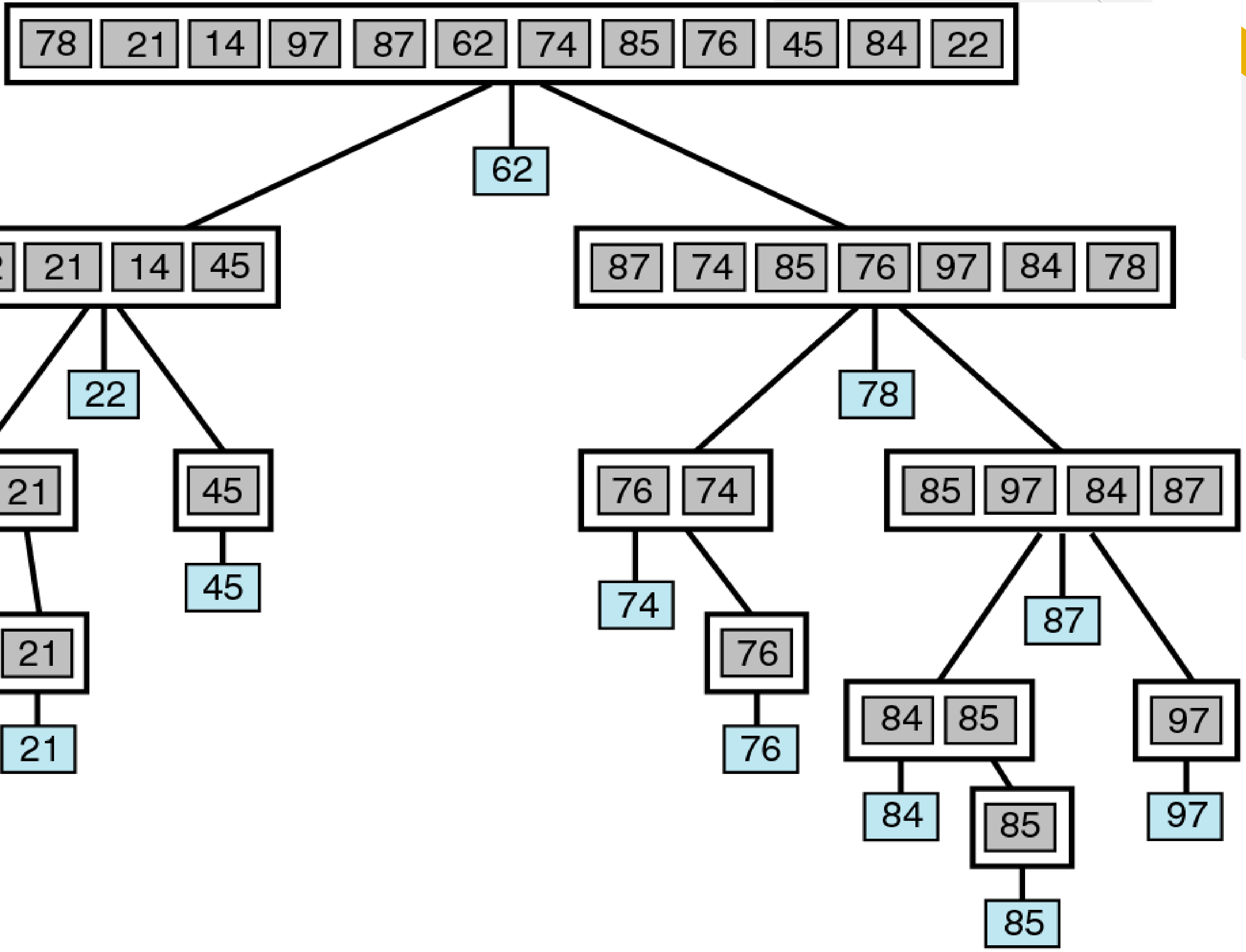
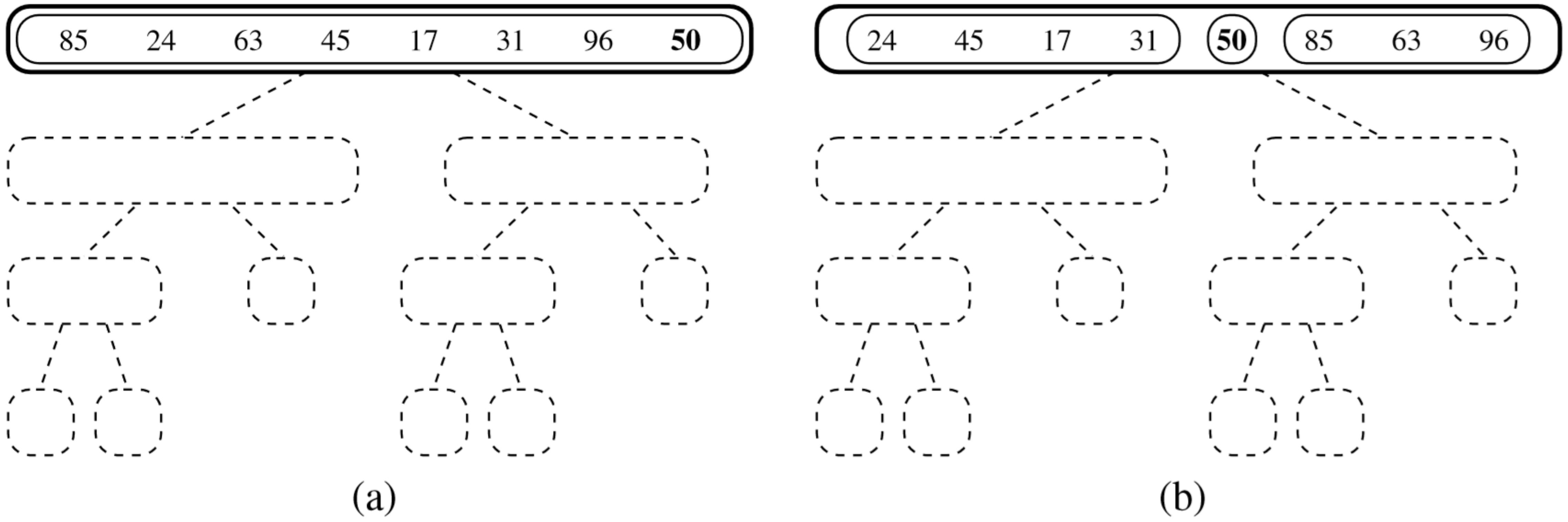


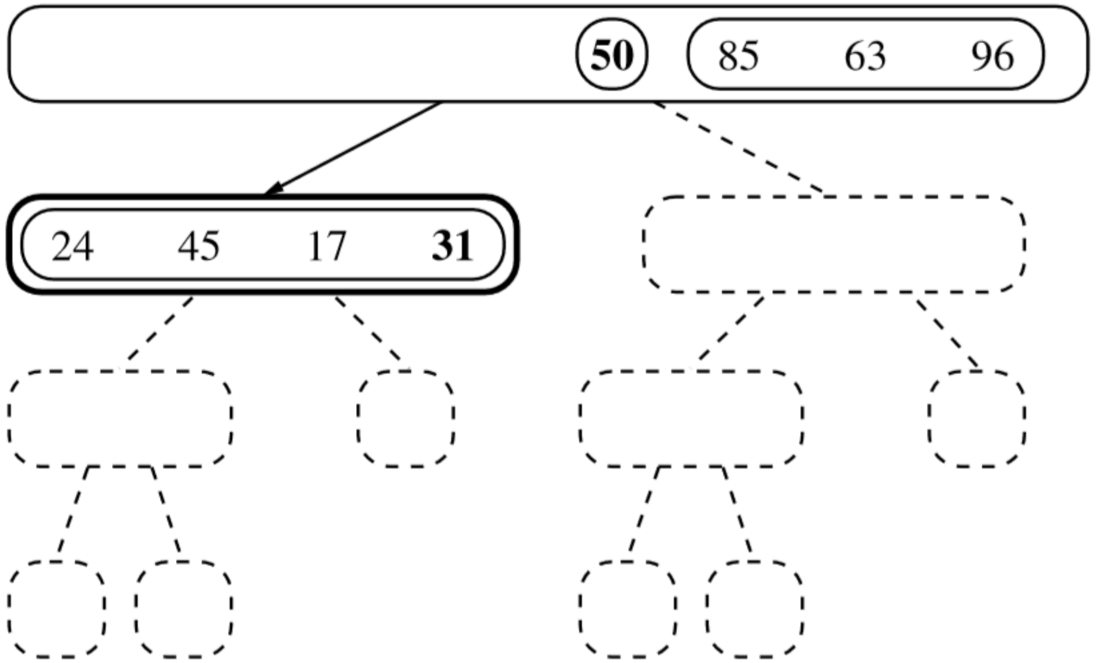
Fig 15-11 Quick sort steps of work (Cont.) [3]

Tutorial 8: Quick Sort Illustration [1]

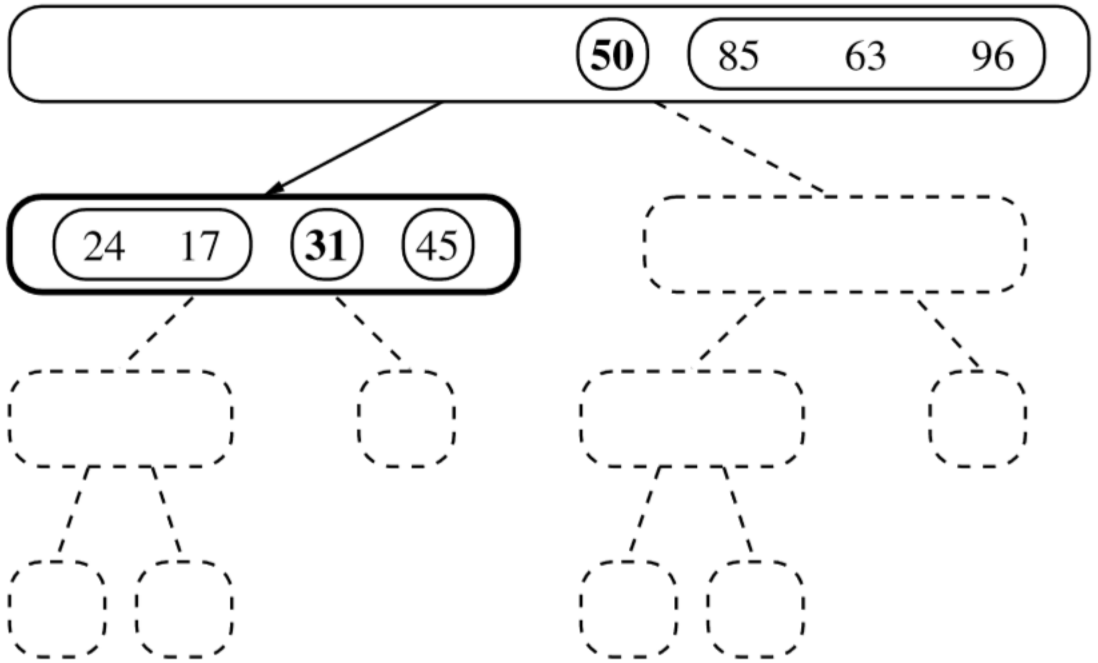
1.6



Tutorial 8: Quick Sort Illustration [1] (Cont.)

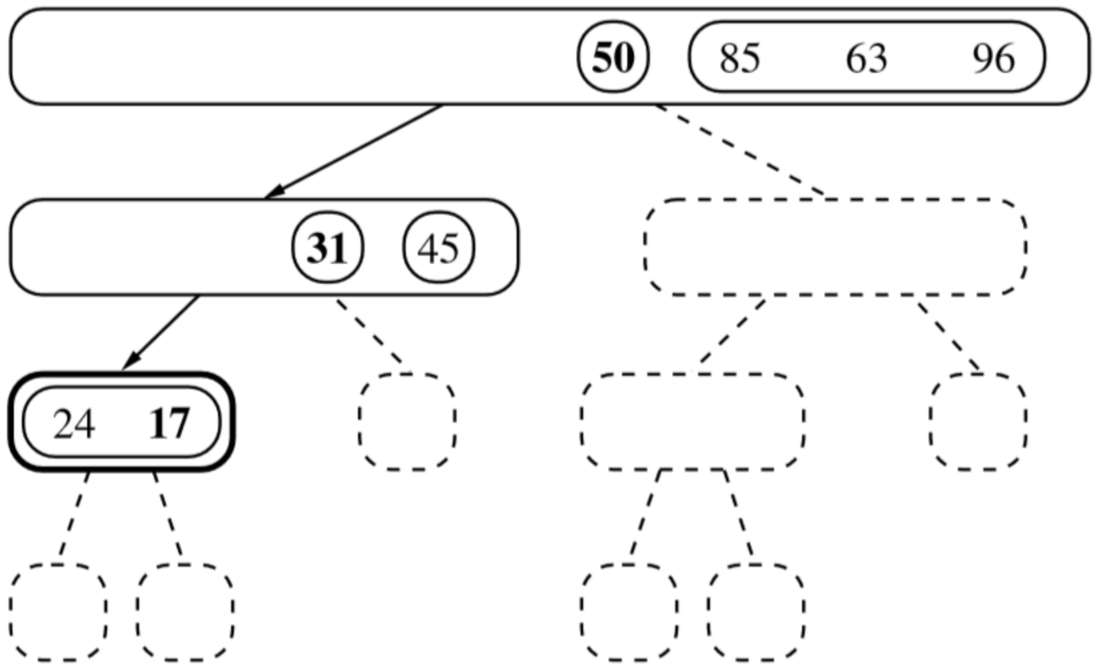


(c)

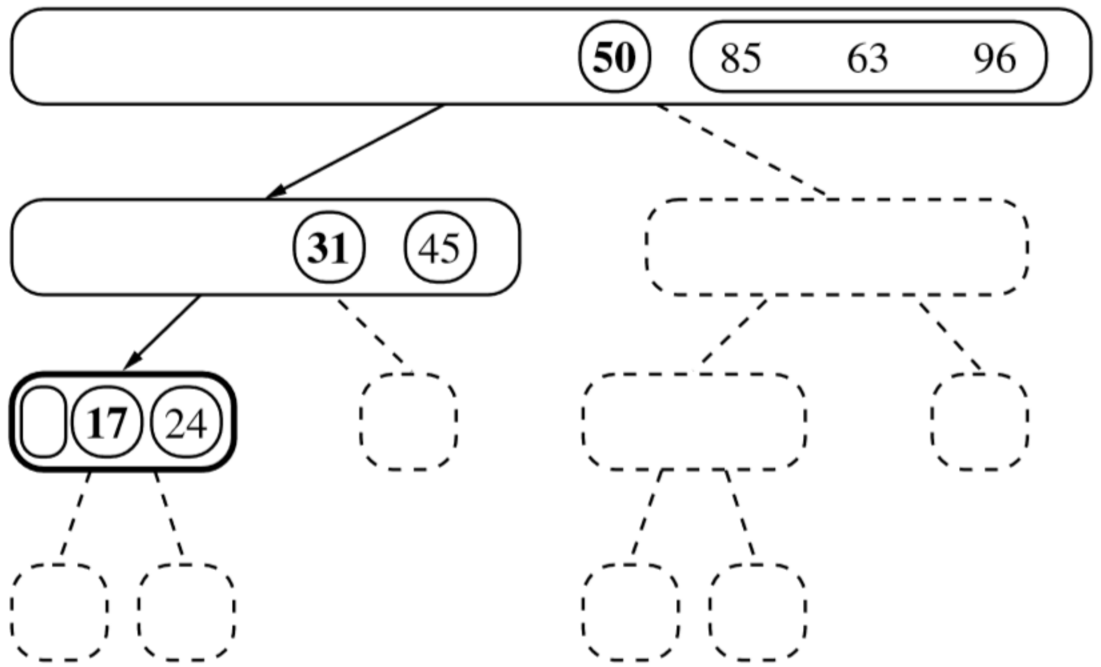


(d)

Tutorial 8: Quick Sort Illustration [1] (Cont.)

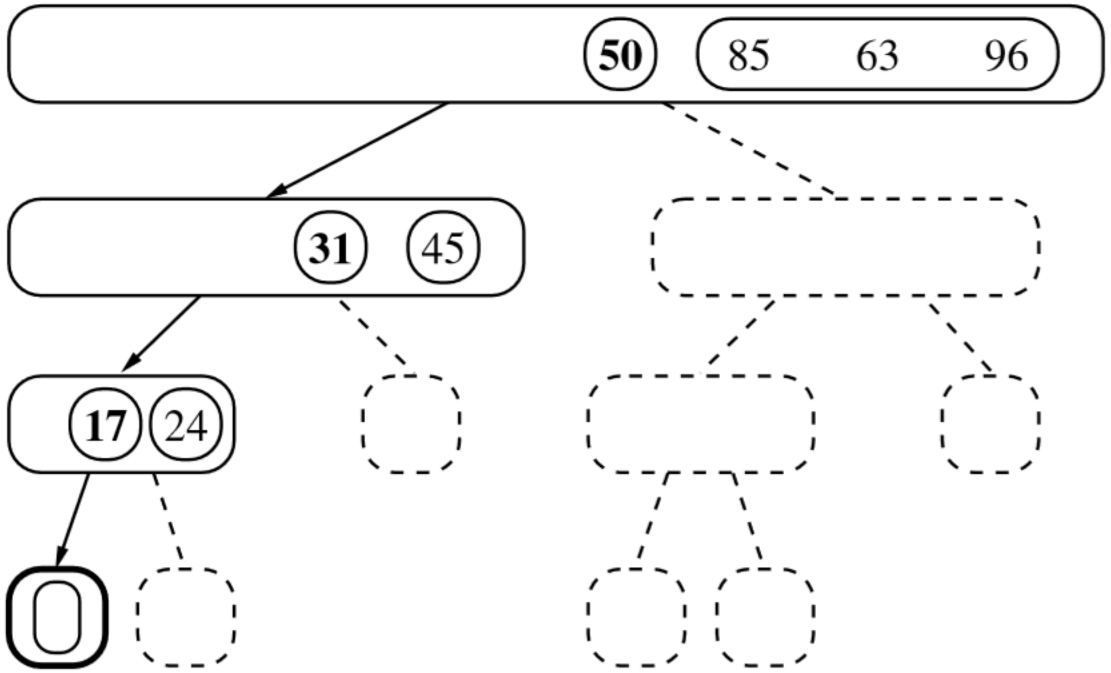


(e)

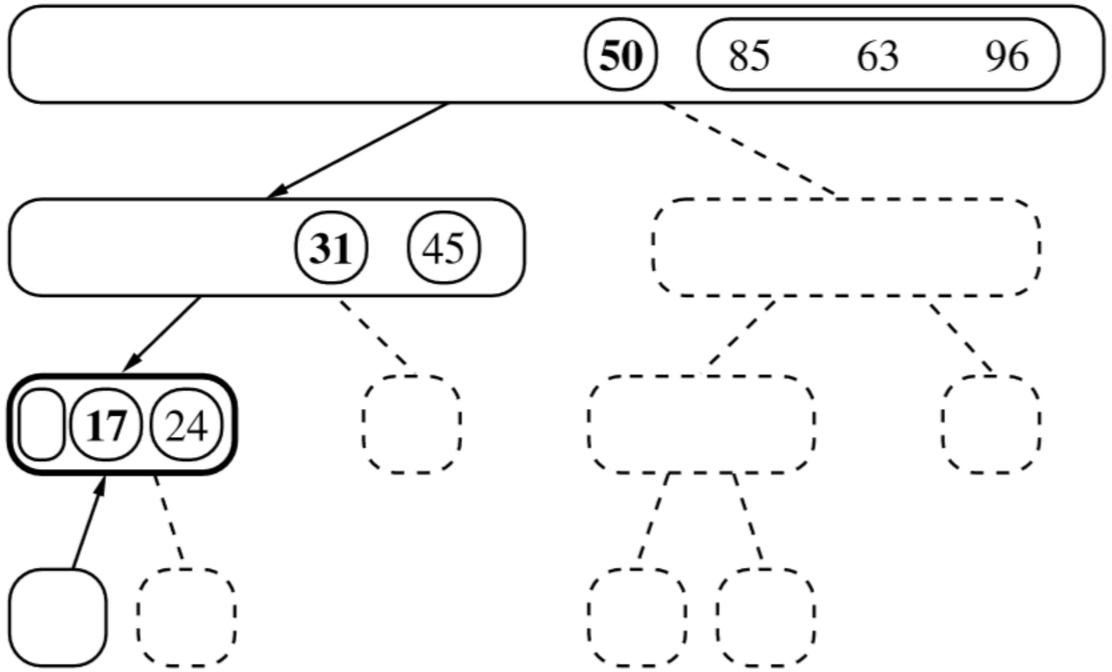


(f)

Tutorial 8: Quick Sort Illustration [1] (Cont.)

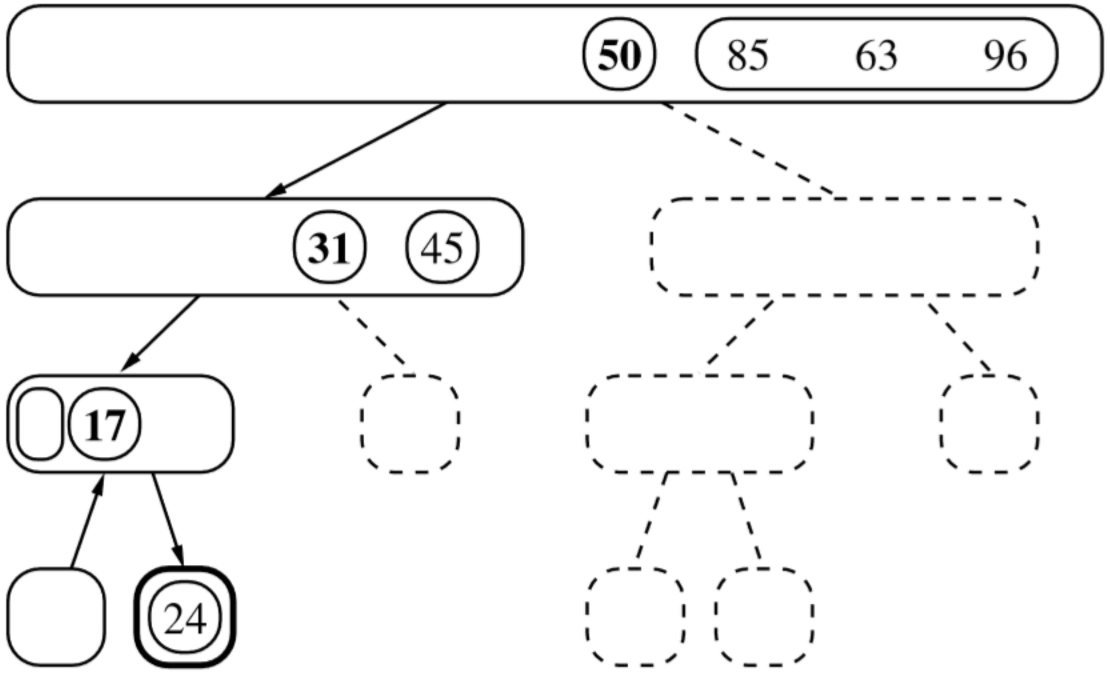


(g)

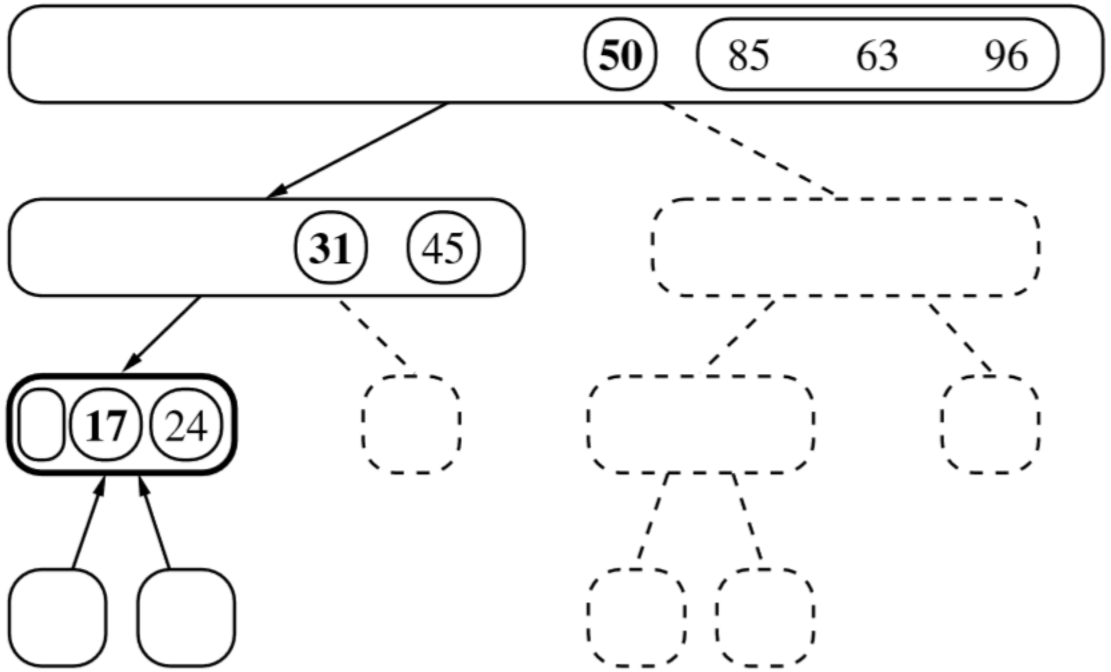


(h)

Tutorial 8: Quick Sort Illustration [1] (Cont.)

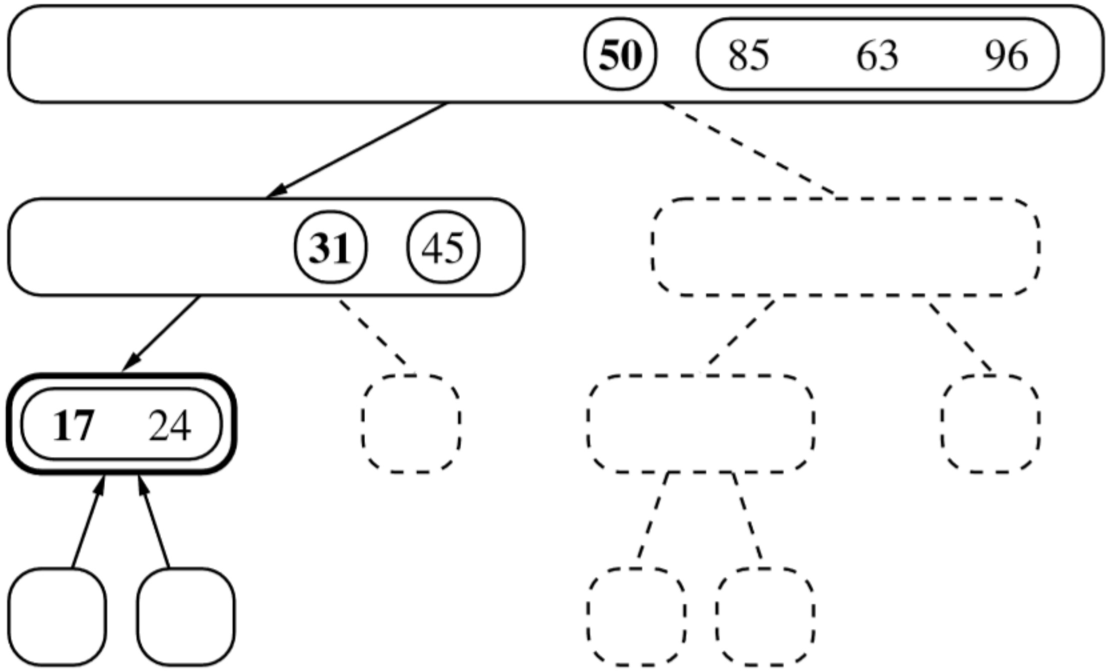


(i)

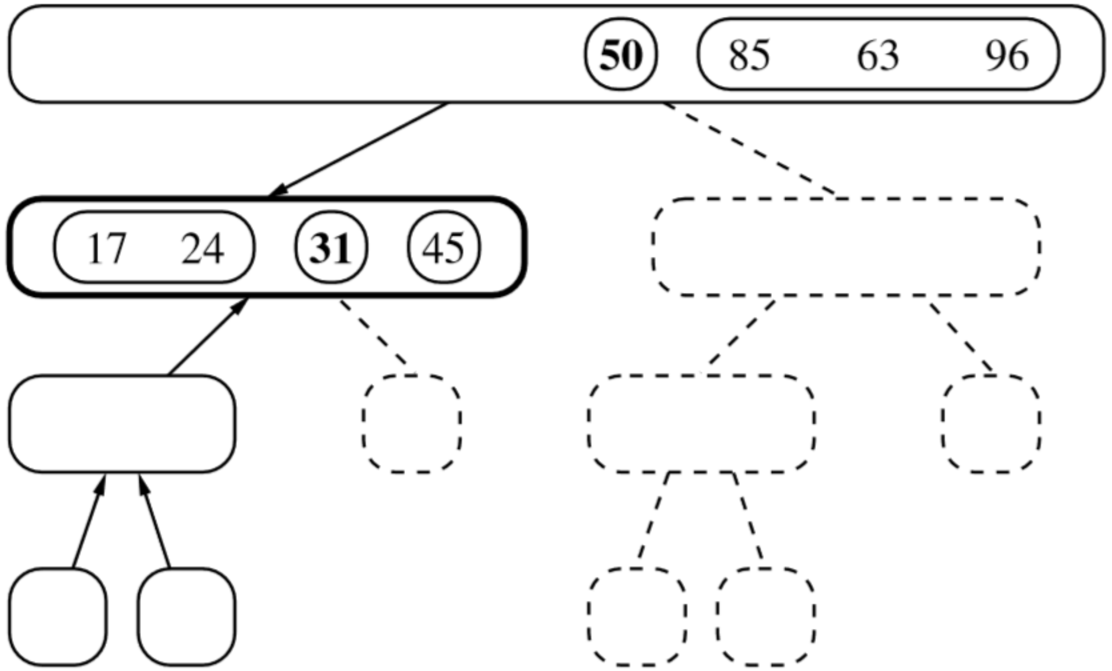


(j)

Tutorial 8: Quick Sort Illustration [1] (Cont.)

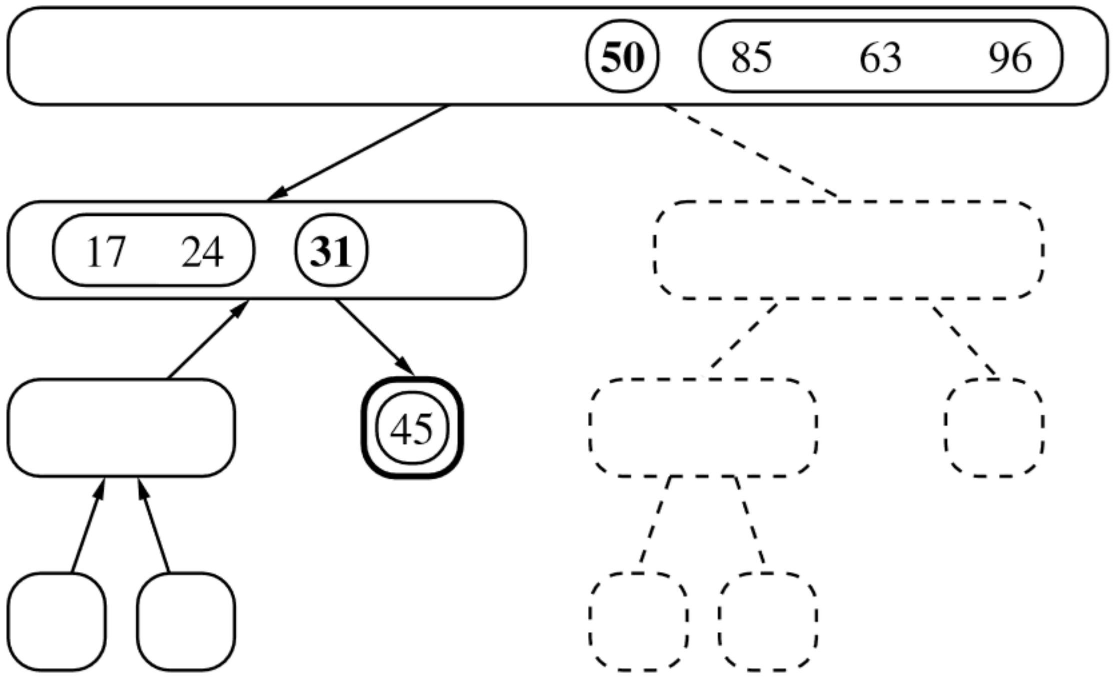


(k)

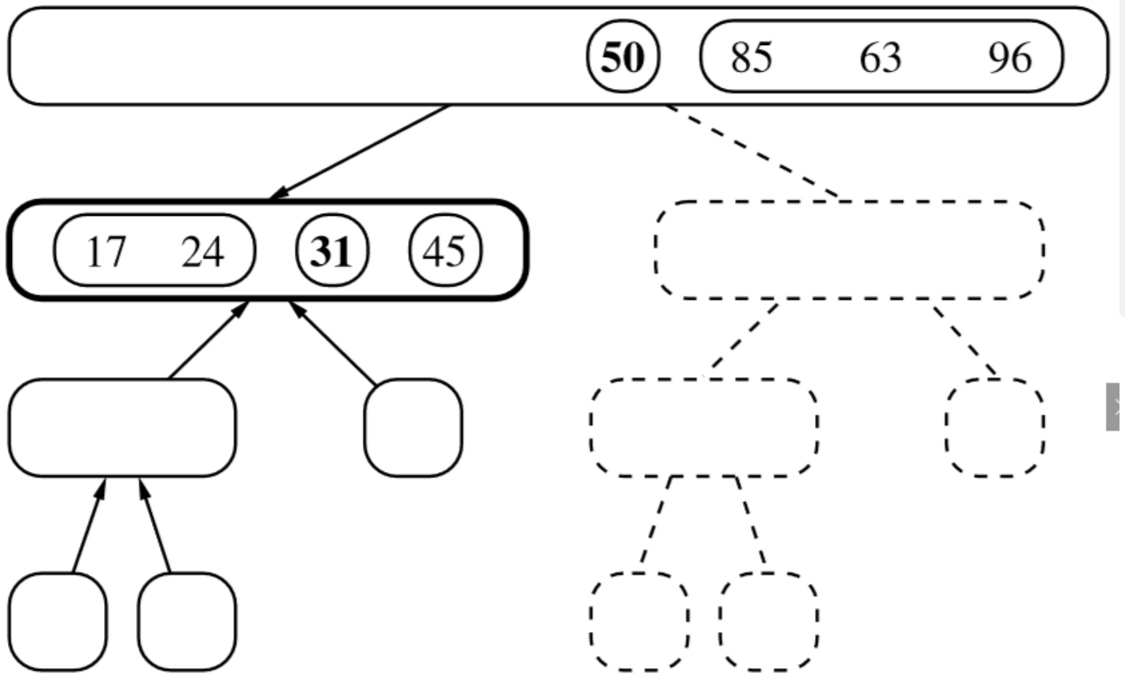


(l)

Tutorial 8: Quick Sort Illustration [1] (Cont.)

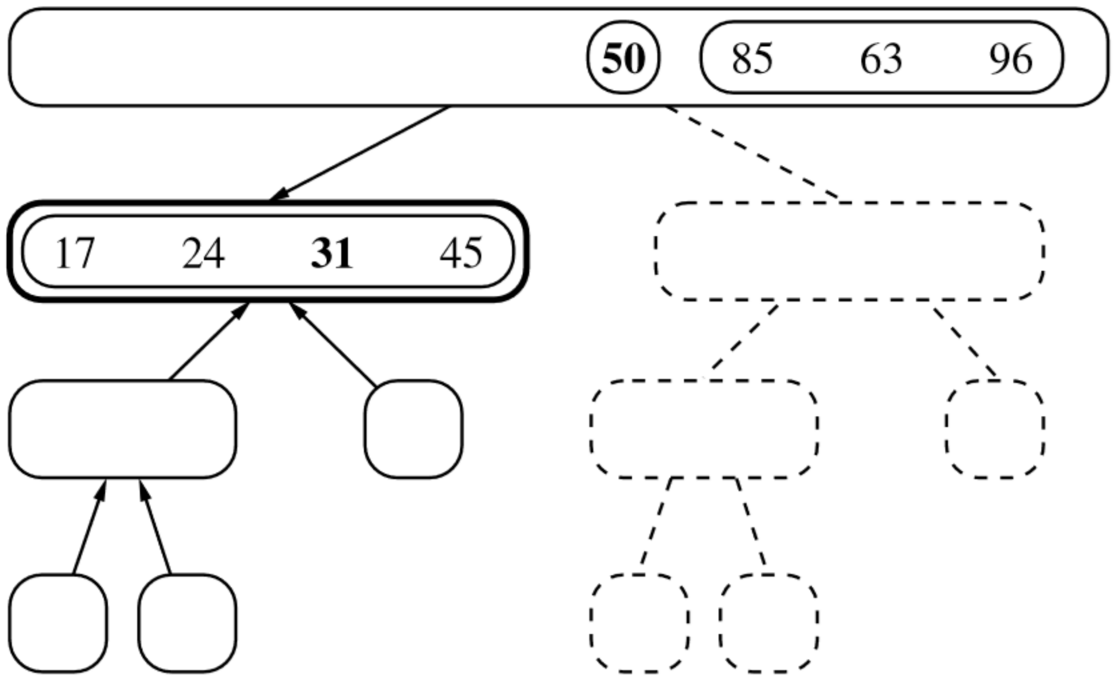


(m)

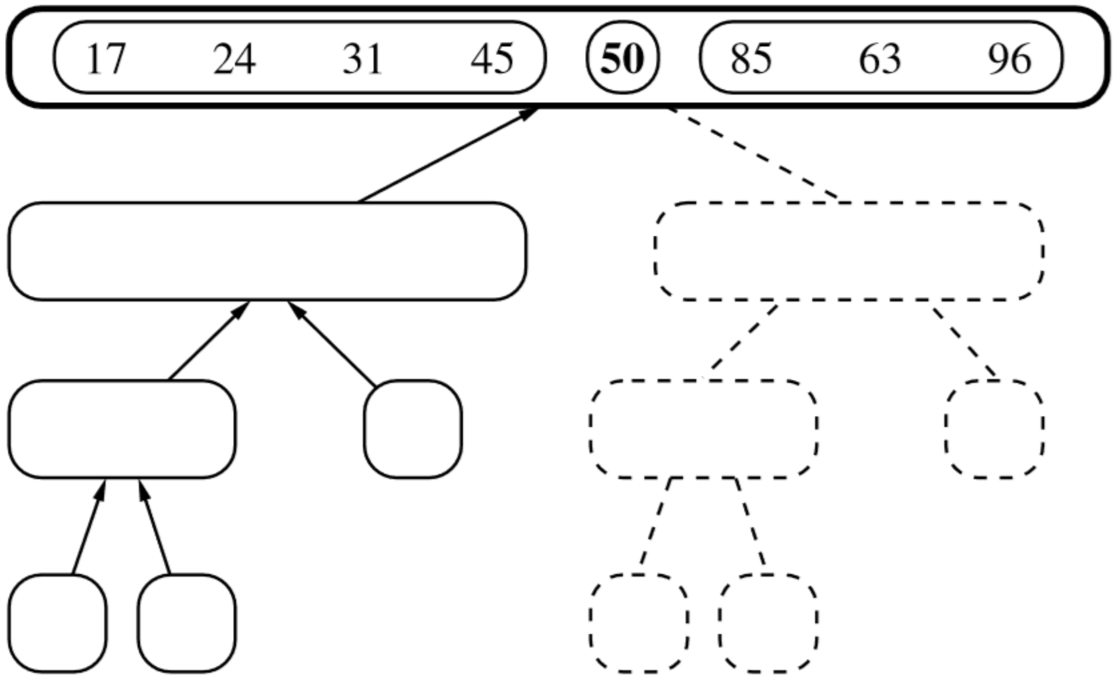


(n)

Tutorial 8: Quick Sort Illustration [1] (Cont.)

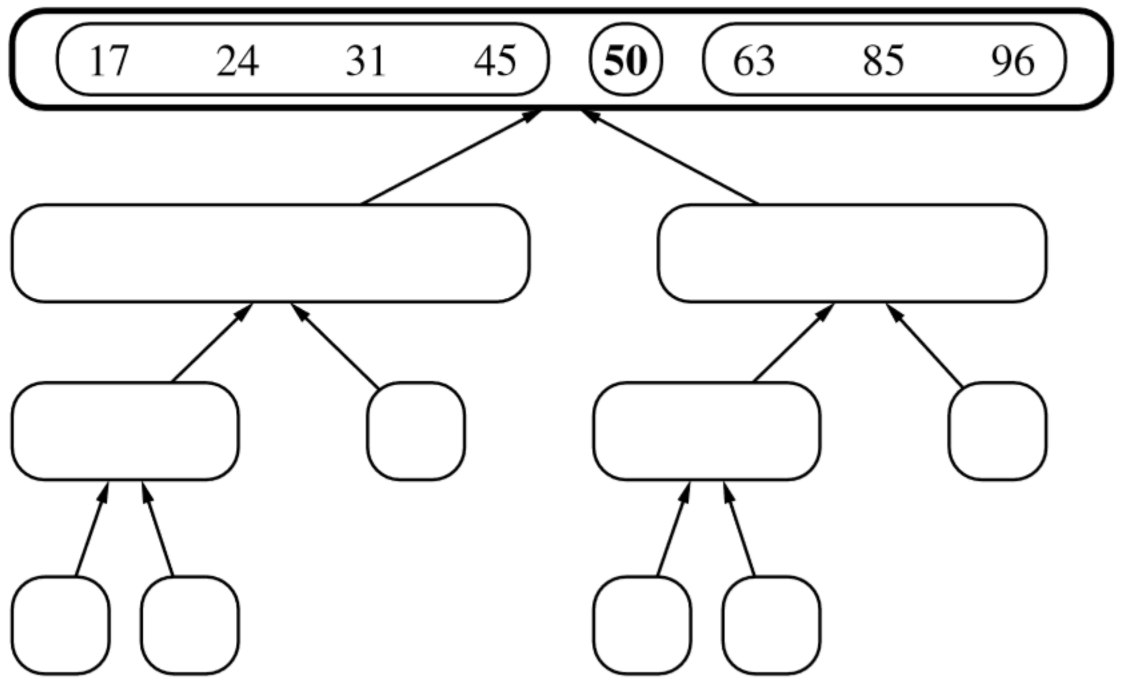


(o)

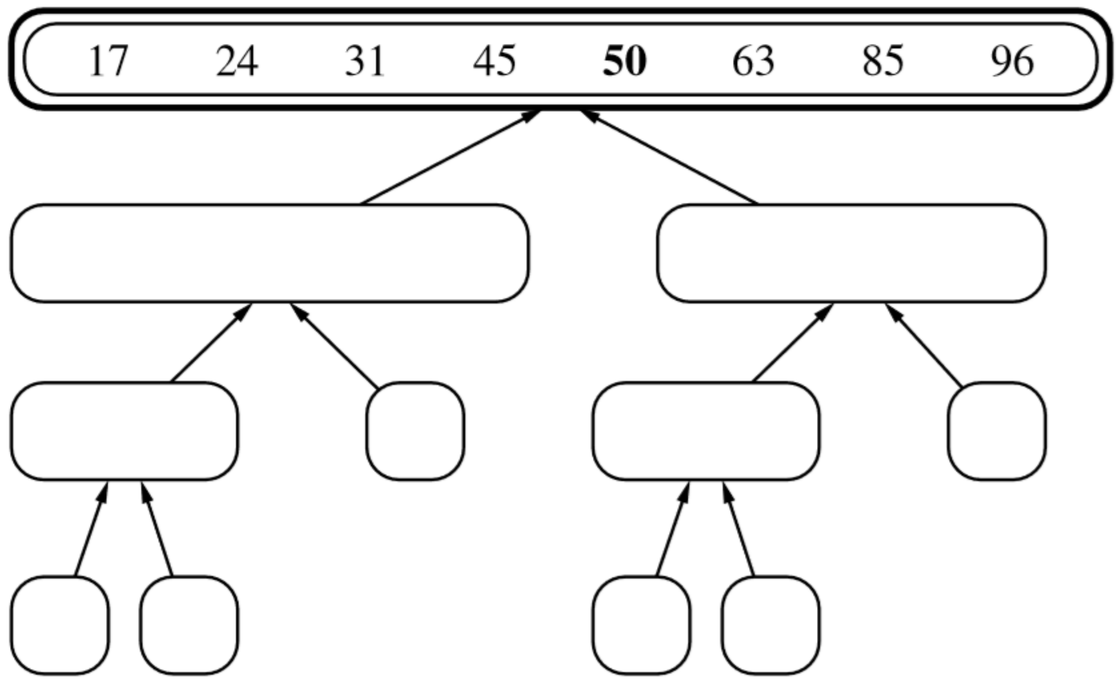


(p)

Tutorial 8: Quick Sort Illustration [1] (Cont.)



(q)



(r)

Tutorial 9: Quick Sort Implementation [2]

1.6

```
def quick_sort(S):
  """Sort the elements of queue S using the quick-sort algorithm."""
  n = len(S)
  if n < 2:
    return                              # list is already sorted
  # divide
  p = S.first( )                        # using first as arbitrary pivot
  L = LinkedQueue( )
  E = LinkedQueue( )
  G = LinkedQueue( )
  while not S.is_empty( ):              # divide S into L, E, and G
    if S.first( ) < p:
      L.enqueue(S.dequeue( ))
    elif p < S.first( ):
      G.enqueue(S.dequeue( ))
    else:                               # S.first() must equal pivot
      E.enqueue(S.dequeue( ))
  # conquer (with recursion)
  quick_sort(L)                         # sort elements less than p
  quick_sort(G)                         # sort elements greater than p
  # concatenate results
  while not L.is_empty( ):
    S.enqueue(L.dequeue( ))
  while not E.is_empty( ):
    S.enqueue(E.dequeue( ))
  while not G.is_empty( ):
    S.enqueue(G.dequeue( ))
```

Quick Sort

- Knuth suggested that when the sort partition becomes small (around 16 elements), a straight insertion (or selection) sort should be used to complete the sorting of the partition !
- The quick sort efficiency is $O(n\log_2 n)$

Quick Sort

n	Number of loops		
	Straight Insertion Sort Straight Selection Sort Bubble Sort	Shell Sort	Heap sort
25	625	55	116
100	10,000	316	664
500	250,000	2,364	4,482
1000	1,000,000	5,623	9,965
2000	4,000,000	13,374	10,965

Table 15-3 Six sort algorithm performances

References

Texts | Integrated Development Environment (IDE)

[1] Data Structures and Algorithms in Python, Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser, Willy & Sons Inc., 2013.

[2] Data Structures and Algorithms Using Python, Rance D. Necaise, John Wiley & Sons, Inc., 2011

[3] Data Structures: A Pseudocode Approach with C++, Richard F. Gilberg and Behrouz A. Forouzan, Brooks/Cole, 2001.

[4] Problem Solving in Data Structures & Algorithms Using Python: Programming Interview Guide, 1st Edition, Hermant Jain, Thiftbooks, March 2017.

[5] https://trinket.io/features/python3

[6] https://colab.research.google.com/

[7] Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein,, Fourth Edition, The MIT Press, 2022.