Chapter 11-3 – Faster Sorting Algorithms

Internal Sort – collection of data fits in memory
External Sort – collection of data does not fit in memory

- Must reside on secondary storage

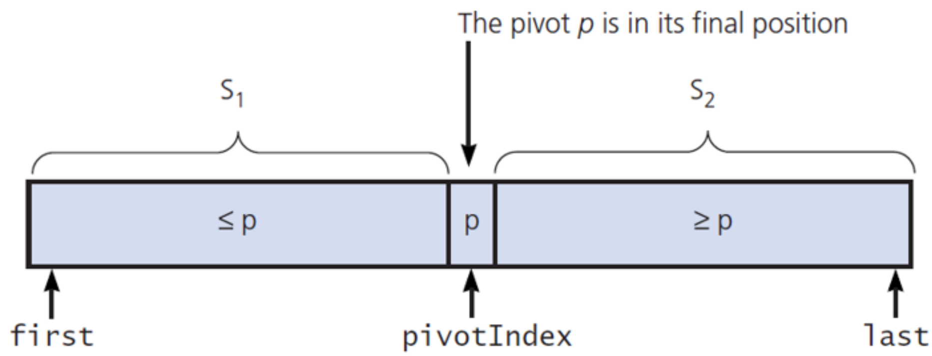
1. Merge Sort – Divide and Conquer sorting algorithm

Divide: divide the n-element sequence to be sorted into two subsequences of n/2 elements each
Conquer: sort the two subsequences recursively using merge sort. IF the length of a sequence is 1, do nothing since it is already in order.
Combine: merge two sorted subsequences to produce the sorted answer.

```
// Sorts theArray[first..last] by
//   1. Sorting the first half of the array
//   2. Sorting the second half of the array
//   3. Merging the two sorted halves
mergeSort(theArray: ItemArray, first: integer, last: integer)
{
   if (first < last)
   {
      mid = (first + last) / 2        // Get midpoint

      // Sort theArray[first..mid]
      mergeSort(theArray, first, mid)

      // Sort theArray[mid+1..last]
      mergeSort(theArray, mid + 1, last)

      // Merge sorted halves theArray[first..mid] and theArray[mid+1..last]
      merge(theArray, first, mid, last)
   }
   // If first >= last, there is nothing to do
}
```

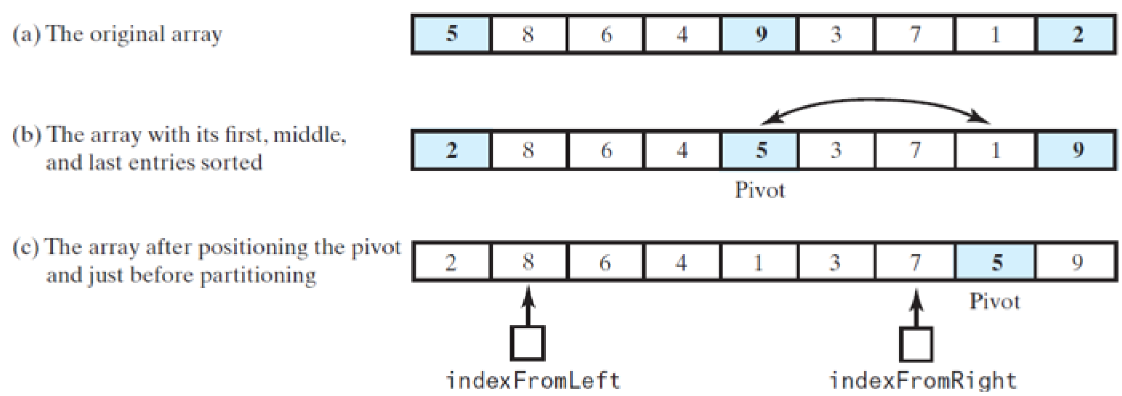
2. The Quick Sort

- Another divide-and-conquer algorithm
- Partitions an array into items that are
  - Less than or equal to the pivot and
  - Those that are greater than or equal to the pivot
- Partitioning places pivot in its correct position within the array
  - Place chosen pivot in theArray[last] before partitioning



Median-of-three pivot selection

```
// Arranges the first, middle, and last entries in an array into ascending order.
sortFirstMiddleLast(theArray: ItemArray, first: integer, mid: integer,
                    last: integer): void
{
   if (theArray[first] > theArray[mid])
      Interchange theArray[first] and theArray[mid]

   if (theArray[mid] > theArray[last])
      Interchange theArray[mid] and theArray[last]

   if (theArray[first] > theArray[mid])
      Interchange theArray[first] and theArray[mid]
}
```




```
// Partitions theArray[first..last].
partition(theArray: ItemArray, first: integer, last: integer): integer
{
   // Choose pivot and reposition it
   mid = first + (last - first) / 2
   sortFirstMiddleLast(theArray, first, mid, last)
   Interchange theArray[mid] and theArray[last - 1]
   pivotIndex = last - 1
   pivot = theArray[pivotIndex]

   // Determine the regions S1 and S2
   indexFromLeft = first + 1
   indexFromRight = last - 2

   done = false
```

```
while (not done)
{
   // Locate first entry on left that is ≥ pivot
   while (theArray[indexFromLeft] < pivot)
      indexFromLeft = indexFromLeft + 1

   // Locate first entry on right that is ≤ pivot
   while (theArray[indexFromRight] > pivot)
      indexFromRight = indexFromRight - 1

   if (indexFromLeft < indexFromRight)
   {
      Interchange theArray[indexFromLeft] and theArray[indexFromRight]
      indexFromLeft = indexFromLeft + 1
      indexFromRight = indexFromRight - 1
   }
   else
      done = true
}
// Place pivot in proper position between S1 and S2, and mark its new location
Interchange theArray[pivotIndex] and theArray[indexFromLeft]
pivotIndex = indexFromLeft

return pivotIndex
}
```

2	8	6	4	1	3	7	5	9

Example A partitioning of an array during a quick sort

3	5	0	7	6	1	2	4

```
// Sorts theArray[first..last].
quickSort(theArray: ItemArray, first: integer, last: integer): void
{
   if (first < last)
   {
      Choose a pivot item p from theArray[first..last]
      Partition the items of theArray[first..last] about p
      // The partition is theArray[first..pivotIndex..last]

      quickSort(theArray, first, pivotIndex - 1) // Sort S1

      quickSort(theArray, pivotIndex + 1, last)  // Sort S2
   }
   // If first >= last, there is nothing to do
}
```

Example Trace the quicksort's partitioning algorithm as it partitions the following array

5	8	6	4	9	3	7	1	2

Example) Trace the quicksort's portioning algorithm as it partitions the following array

38	16	40	39	12	27

Analysis

Partitioning – O(n)

i) Find pivot: median –of-three selection

ii) partitioning

Average case

Worst case

A Comparison of sorting algorithms

	Worst case	Average case
Selection sort	n^2	n^2
Bubble sort	n^2	n^2
Insertion sort	n^2	n^2
Merge sort	$n \times \log n$	$n \times \log n$
Quick sort	n^2	$n \times \log n$

Quiz

(1) Trace the merge sort algorithm as it sorts the following array into ascending order.
21, 83, 45, 20, 66, 31

(2) Trace the quick sort algorithm as it sorts the following array into ascending order.

21, 83, 45, 20, 66, 13, 17