Sorting Algorithms

- Number of Comparisons
- Number of Data Movements
- Additional Space Requirements
Figure 2.4  The operation of merge sort on the array $A = (5, 2, 4, 7, 1, 3, 2, 6)$. The lengths of the sorted sequences being merged increase as the algorithm progresses from bottom to top.
Figure 2.3 The operation of lines 10–17 in the call \texttt{MERGE(A, 9, 12, 16)}, when the subarray \(A[9..16]\) contains the sequence \(\{2, 4, 5, 7, 1, 2, 3, 6\}\). After copying and inserting sentinels, the array \(L\) contains \(\{2, 4, 5, 7, \infty\}\), and the array \(R\) contains \(\{1, 2, 3, 6, \infty\}\). Lightly shaded positions in \(A\) contain their final values, and lightly shaded positions in \(L\) and \(R\) contain values that have yet to be copied back into \(A\). Taken together, the lightly shaded positions always comprise the values originally in \(A[9..16]\), along with the two sentinels. Heavily shaded positions in \(A\) contain values that will be copied over, and heavily shaded positions in \(L\) and \(R\) contain values that have already been copied back into \(A\). (a)-(h) The arrays \(A\), \(L\), and \(R\), and their respective indices \(k, i,\) and \(j\) prior to each iteration of the loop of lines 12–17. (i) The arrays and indices at termination. At this point, the subarray in \(A[9..16]\) is sorted, and the two sentinels in \(L\) and \(R\) are the only two elements in these arrays that have not been copied into \(A\).
Assumption: Array A is sorted from positions p to q and also from positions q+1 to r.

The merge algorithm is as follows:

```
MERGE(A, p, q, r)
1  n₁ ← q - p + 1
2  n₂ ← r - q
3  create arrays L[1..n₁ + 1] and R[1..n₂ + 1]
4  for i ← 1 to n₁
5      do L[i] ← A[p + i - 1]
6  for j ← 1 to n₂
7      do R[j] ← A[q + j]
8  L[n₁ + 1] ← ∞
9  R[n₂ + 1] ← ∞
10 i ← 1
11 j ← 1
12 for k ← p to r
13      do if L[i] ≤ R[j]
14          then A[k] ← L[i]
15              i ← i + 1
16          else A[k] ← R[j]
17              j ← j + 1
```

O(n)
MERGE-SORT(A, p, r)
1    if p < r
2        then q ← \lfloor (p + r)/2 \rfloor
3    MERGE-SORT(A, p, q)
4    MERGE-SORT(A, q + 1, r)
5    MERGE(A, p, q, r)
Figure 2.5 The construction of a recursion tree for the recurrence $T(n) = 2T(n/2) + cn$. Part (a) shows $T(n)$, which is progressively expanded in (b)–(d) to form the recursion tree. The fully expanded tree in part (d) has $\lg n + 1$ levels (i.e., it has height $\lg n$, as indicated), and each level contributes a total cost of $cn$. The total cost, therefore, is $cn \lg n + vn$, which is $\Theta(n \lg n)$. 
Figure 8.10  Quicksort

Data Structures & Problem Solving using JAVA/2E  Mark Allen Weiss  © 2002  Addison Wesley
Figure A  If 6 is used as pivot, the end result after partitioning is as shown in the Figure B.

```
2 1 4 5 0 3 9 8 7 6
```

Figure B  Result after Partitioning
```
2 1 4 5 0 3 6 8 7 9
```
QuickSort

QuickSort(array A, int p, int r)
1  if (p < r)
2   then q ← Partition(A, p, r)
3   QuickSort(A, p, q − 1)
4   QuickSort(A, q + 1, r)

To sort array call QuickSort(A, 1, length[A]).

Partition(array A, int p, int r)
1  x ← A[r]  ▶ Choose pivot
2  i ← p − 1
3  for j ← p to r − 1
4  do if (A[j] ≤ x)
5   then i ← i + 1
7  exchange A[i + 1] ← A[r]
8  return i + 1

Page 146, CLRS
Problems with QuickSort

• Bad pivot choices can lead to worst-case behavior of $O(n^2)$
• High cost of recursive calls
• Unstable sorting algorithm
• Lots of potential data movements.
Suggestions for improvement

- Improved choice of pivot to improve worst-case time complexity
  - Randomly choose pivot
  - Pick median of three choices (deterministic or random)

- Huge cost incurred by recursive calls especially when the sublists get smaller and smaller
  - When the sublist becomes smaller than some threshold, call InsertionSort or some other non-recursive sorting algorithm
Questions to ask

• Worst-case versus average-case behavior of the sorting algorithms
• Why is QuickSort fast in practice as opposed to others?
• How can we improve on it?
• How is it different from the divide-and-conquer algorithm, MergeSort?
Sorting Algorithms

• SelectionSort
• InsertionSort
• BubbleSort
• ShakerSort
• QuickSort
• MergeSort
• HeapSort
• Bucket & Radix Sort
• Counting Sort
Problems to think about!

• What is the least number of comparisons you need to sort a list of 3 elements? 4 elements? 5 elements?

• How to arrange a tennis tournament in order to find the tournament champion with the least number of matches? How many tennis matches are needed?