The Problem

- **Input**: a sequence of $n$ numbers $\langle a_1 \ldots a_n \rangle$.
- **Output**: a permutation (re-ordering) $\langle a_1' \ldots a_n' \rangle$ of the input sequence s.t. $a_1' \le a_2' \ldots \le a_n'$.
- Numbers to sort are also called **keys**.
Sorting Algorithms

- Bubble sort.
- Selection sort.
- Insertion sort.
- Quick sort.
- Merge sort.
- Heap sort.

Application: priority queues.
Bubble Sort

- Pseudo code in problem 2-2 p38.
- Array indexed from 1 to n.

```
for i = 1 to n-1 do
    for j = n downto i+1 do
        if a[j] < a[j-1] then swap(a[j],a[j-1])
```

- Running time = n-1+n-2+...+1=n(n-1)/2 = Θ(n²).
Bubble Sort

for $i = 1$ to $n-1$ do
  for $j = n$ downto $i+1$ do
    if $a[j] < a[j-1]$ then swap$(a[j], a[j-1])$
Bubble Sort - Correctness

Loop invariant: Sub-array a[1..i] is sorted before the loop on i.

- Initialization: true before 1\textsuperscript{st} iteration.
- Maintenance: If it is true before an iteration, it remains true before the next iteration.
- Termination: The loop terminates and when it does, the invariant gives us a useful property.
Selection Sort

- Corresponds to exercise 2.2-2 p27.

```
for i = 1 to n-1 do
    for j = i+1 to n do
        if a[i] > a[j] then swap(a[i],a[j])
```

- Loop invariant: Sub-array a[1...i] is sorted before the loop on i.
- Running time: $n-1+n-2+\ldots+1 = n(n-1)/2 = \Theta(n^2)$. 
Selection Sort

for $i = 1$ to $n-1$ do
  for $j = i+1$ to $n$ do
    if $a[i] > a[j]$ then swap($a[i], a[j]$)
Insertion Sort

- Idea: Like when you play cards.
- Loop invariant: Sub-array $a[1...j-1]$ is sorted before the loop on $j$.
- Running time: $\Theta(n^2)$?

```
for j = 2 to n do
    key = a[j]
    i = j-1
    while i > 0 and a[i] > key do
        a[i+1] = a[i]
        i = i-1
    done
    a[i+1] = key
done
```
Insertion Sort

\[
\text{for } j = 2 \text{ to } n \text{ do}
\]
\[
\text{key} = a[j]
\]
\[
i = j - 1
\]
\[
\text{while } i > 0 \text{ and } a[i] > \text{key} \text{ do}
\]
\[
a[i+1] = a[i]
\]
\[
i = i - 1
\]
\[
\text{done}
\]
\[
a[i+1] = \text{key}
\]
\[
\text{done}
\]
Example

8  2  4  9  3  6
2  8  4  9  3  6
2  4  8  9  3  6
2  4  8  9  3  6
2  3  4  8  9  6
2  3  4  6  8  9
Analysis of Insertion Sort

Count executions of loops:
- Test n times, execute n-1 times (outer loop).
- Test $t_j$ times, execute $t_j-1$ times (inner loop).
- $c_1 n + c_2 \sum_j t_j$

Analysis:
- Best case: $t_j=1$, $T(n)=\Omega(n)$. Can be good!
- Worst case: $t_j=j$, $T(n)=O(n^2)$.
- Average case: $t_j=j/2$, $E[T(n)]=O(n^2)$. 
Divide-and-Conquer

- *Divide-and-conquer* approach.
  - Recursive algorithms.
  - **Divide** main problem into sub-problems.
  - **Conquer** the sub-problems (solve recursively).
  - **Combine** the solutions of the sub-problems.
- Previous approaches were *incremental*. 
Merge Sort

- Merge sort:
  - Divide array \( n \) into 2 arrays of size \( n/2 \).
  - Sort 2 smaller arrays with merge sort.
  - Combine the smaller arrays.

- Careful: Running time comes from
  - number of divide (= recursive calls)
  - + time of combining solutions.
Merge Sort

- Key of the algorithm: Merge the sorted sub-arrays $A[p...q]$ and $A[q+1...r]$ \textbf{linearly} in time.

- Description (book) uses copies and sentinel values (to simplify tests).

```plaintext
merge_sort(a,p,r):
  if p < r then
    q = (p+r)/2
    merge_sort(a, p, q)
    merge_sort(a, q+1, r)
  merge(a, p, q, r)
fi
```
Merge Sort

merge_sort(a, p, r):
  if p < r then
    q = (p+r)/2
    merge_sort(a, p, q)
    merge_sort(a, q+1, r)
    merge(a, p, q, r)
  fi
Key: Merging 2 Sorted Arrays

Time = $\Theta(n)$ to merge a total of $n$ elements (linear time).
Analyzing Merge Sort

\[
T(n) \quad \Theta(1)
\]

(sloppy) \quad 2T(n/2) \quad \Theta(n)

merge_sort(a, p, r):
   if p < r then
      q = (p+r)/2
      merge_sort(a, p, q)
      merge_sort(a, q+1, r)
      merge(a, p, q, r)
   fi
Analyzing Merge Sort

- Running time $T(n)$ given as a recurrence. In general, for divide-and-conquer algorithms:
  - $T(n)=\Theta(1)$ if $n\leq c$ ($c$ small enough) otherwise
  - $T(n)=aT(n/b)+D(n)+C(n)$. (divide-conquer & combine)
- Easy to solve!
- Merge-sort: $c=1$, $D(n)=1$, $C(n)=\Theta(n)$.
  - Master method: $a=2$, $b=2 \Rightarrow T(n)=\Theta(n \log n)$.

BUT: Algorithm is not in-place.
Quick Sort

- Divide-and-conquer algorithm.
  - Worst case $\Theta(n^2)$.
  - Average (robust) case $\Theta(n \lg n)$.
  - Very efficient in practice.

- In-place algorithm with small constants (complexity). Fastest sort in practice except for “almost-sorted” inputs.
Quick Sort

- **Divide:** Partition $A[p..r]$ into $A'[p..q-1]$ and $A'[q+1..r]$ s.t. $a’_{p..q-1} \leq a’_q \leq a’_{q+1..r}$.

- **Conquer:** Sort $A'[p..q-1]$ and $A'[q+1..r]$ by recursive calls to quick-sort.

- **Combine:** $A[p..r]$ is sorted. Nothing to do!

- **Key:** How to partition. We can choose $a’_q$ arbitrarily.
Divide-and-Conquer

1) Divide:
   Choose *pivot* element.

   \[ \leq \quad \text{\textbullet} \quad \geq \]

   Partition the array.

2) Conquer: Recursive calls.
3) Combine: Trivial.

**KEY:** Linear time partitioning sub-routine.
Quick Sort Partitioning

- Equivalent to selection-sort for worst case.

\[
\text{partition}(A,p,r) \\
x = A[r] \\
i = p-1 \\
\text{for } j = p \text{ to } r-1 \text{ do} \\
\quad \text{if } A[j] \leq x \text{ then} \\
\quad \quad i = i+1 \\
\quad \quad \text{swap}(A[i],A[j]) \\
\quad \text{fi} \\
\text{done} \\
\text{swap}(A[i+1],A[r]) \\
\text{return } i+1
\]
Variants

- Hoare’s partitioning more efficient.
  - Problem 7-1 p160.
  - Fewer swaps in practice.

- Randomized quick-sort:
  - Choose the pivot randomly.
  - No worst-case input but unlucky run may take $\Theta(n^2)$.
  - More robust.

- Combine quick-sort with insertion sort.
partition(A, p, r)
\[ x = A[r] \]
i = p - 1
for j = p to r - 1 do
  if \( A[j] \leq x \) then
    i = i + 1
    swap(A[i], A[j])
  fi
done
swap(A[i + 1], A[r])
return i + 1
Worst Case of Quick Sort

- Input sorted or reverse order.
- Partition around \textit{min} or \textit{max} element.
- Always no element for one side.

\[
T(n) = T(0) + T(n-1) + \Theta(n) \\
= \Theta(1) + T(n-1) + \Theta(n) \\
= T(n-1) + \Theta(n) \\
= \Theta(n^2)
\]
Worst Case Recursion Tree

\[ T(n) = T(0) + T(n-1) + cn \]

\[ \Theta(n^2) \]
Best Case Analysis

**Intuition**

- If we are lucky, the partition splits the array evenly. Like merge-sort:
  \[
  T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)
  \]

- What if the split is always 1/10 : 9/10?
  \[
  T(n) = T(n/10) + T(9n/10) + \Theta(n)
  \]

Solution?
Recursion Tree 1/10 : 9/10

\[ \log_{10} n \quad \text{cn/10} \quad \text{cn/100} \quad \ldots \quad \Theta(1) \]

\[ \Theta(n) \text{ leaves} \]

\[ \Theta(n) \leq T(n) \leq cn \log_{10/9} n + \Theta(n) \rightarrow \Theta(n \log n) \quad \text{lucky} \]
Performance of Quick Sort

- Depends on the input (balanced array or not) and the choice of the pivot.
- Worst-case (sorted, reversed, equal elements): $\Theta(n^2)$.
- Best-case: (balanced): $T(n)=\Theta(n \log n)$.
- Average case?
  - Remark: 1/10 : 9/10 still in $n \log n$.
  - Average is $\Theta(n \log n)$.
  - Analysis: Randomized quick-sort...
Randomized Quick Sort

- Pick up a random pivot for the partitioning.
- Running time unpredictable.
- No assumption on the input distribution.
- No specific input gives the worst case.
- Used for analysis of quick sort for the average case.
Randomized Quick Sort Analysis

- $T(n)$=random variable for the running time of randomized quick sort (input of size $n$), assuming independent random numbers.

- For $k=0\ldots n-1$, define the indicator random variable

$$X_k = \begin{cases} 1 & \text{if partition generates a } k : n-k-1 \text{ split} \\ 0 & \text{otherwise} \end{cases}$$

$$E[X_k] = \Pr\{X_k = 1\} = \frac{1}{n} \text{ since all splits are equally likely, assuming distinct elements.}$$
$T(n)$ with Indicator Variable

$$T(n) = \begin{cases} 
T(0) + T(n-1) + \Theta(n) & \text{if } 0 : n-1 \text{ split,} \\
T(1) + T(n-2) + \Theta(n) & \text{if } 1 : n-2 \text{ split,} \\
& \quad \vdots \\
T(n-1) + T(0) + \Theta(n) & \text{if } n-1 : 0 \text{ split.}
\end{cases}$$

$$T(n) = \sum_{k=0}^{n-1} X_k \left( T(k) + T(n-k-1) + \Theta(n) \right)$$
Expected value

\[ E[T(n)] = E\left[ \sum_{k=0}^{n-1} X_k (T(k) + T(n-k-1) + \Theta(n)) \right] \]

\[ = \sum_{k=0}^{n-1} E[X_k (T(k) + T(n-k-1) + \Theta(n))] \]

\[ = \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n-k-1) + \Theta(n)] \]

\[ = \frac{1}{n} \sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n} \sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n} \sum_{k=0}^{n-1} \Theta(n) \]

\[ = \frac{2}{n} \sum_{k=0}^{n-1} E[T(k)] + \Theta(n) \]

**Tough recurrence!**
Solving the Recurrence

\[ E[T(n)] = \frac{2}{n} \sum_{k=2}^{n-1} E[T(k)] + \Theta(n) \]

(k=0 and k=1 can be absorbed in \( \Theta(n) \))

- **Prove** \( E[T(n)] \leq an \lg n \) for a constant \( a > 0 \). Choose a large enough so that \( an \lg n \) dominates \( E[T(n)] \) for sufficiently small \( n \geq 2 \).
- Use fact: \( \sum_{k=2}^{n-1} k \lg k \leq \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 \) (exercise)
Substitution Method...

\[ E[T(n)] \leq \frac{2}{n} \sum_{k=2}^{n-1} ak \log k + \Theta(n) \]  

(use substitution)

\[ = \frac{2a}{n} \left( \frac{1}{2} n^2 \log n - \frac{1}{8} n^2 \right) + \Theta(n) \]  

(use fact)

\[ = an \log n - \left( \frac{an}{4} - \Theta(n) \right) \]

\[ \leq an \log n \quad \text{with } a \text{ large enough} \]

\[ E[T(n)] = O(n \log n) \]