CS 171: Introduction to Computer Science II

Quicksort
Welcome back from Spring break!

• Quiz 1 distributed
• Hw3 with 2 late credit is due 3/20 (today)
• Hw4 with 1 late credit is due 3/21, with 2 late credits is due 3/24
Outline

• Quicksort algorithm (review)
• Quicksort Analysis (cont.)
  – Best case
  – Worst case
  – Average case
• Quicksort improvements
• Sorting summary
• Java sorting methods
Quicksort

Basic plan.

- **Shuffle** the array.
- **Partition** so that, for some \( j \)
  - entry \( a[j] \) is in place
  - no larger entry to the left of \( j \)
  - no smaller entry to the right of \( j \)
- **Sort** each piece recursively.

![Quicksort diagram](image-url)
Quicksort partitioning

Basic plan.
- Scan $i$ from left for an item that belongs on the right.
- Scan $j$ from right for an item that belongs on the left.
- Exchange $a[i]$ and $a[j]$.
- Repeat until pointers cross.

Partitioning trace (array contents before and after each exchange)
Quicksort: Java code for partitioning

```java
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo, j = hi + 1;
    while (true) {
        while (less(a[++i], a[lo]))
            if (i == hi) break;

        while (less(a[lo], a[--j]))
            if (j == lo) break;

        if (i >= j) break;
        exch(a, i, j);
    }
    exch(a, lo, j);
    return j;
}
```

- **before**
  - lo
  - hi

- **during**
  - \( 
    \begin{array}{c}
    \leq \\
    \geq
    \end{array}
  \)  
  - \( i \)  
  - j

- **after**
  - \( \leq \)  
  - v  
  - \( \geq \)  
  - hi
public class Quick
{
    private static int partition(Comparable[] a, int lo, int hi)
    {
        /* see previous slide */
    }

    public static void sort(Comparable[] a)
    {
        StdRandom.shuffle(a);
        sort(a, 0, a.length - 1);
    }

    private static void sort(Comparable[] a, int lo, int hi)
    {
        if (hi <= lo) return;
        int j = partition(a, lo, hi);
        sort(a, lo, j-1);
        sort(a, j+1, hi);
    }
}
Quicksort Cost Analysis

• Depends on the partitioning
  – What’s the best case?
  – What’s the worst case?
  – What’s the average case?
Quicksort: best-case analysis

<table>
<thead>
<tr>
<th>lo</th>
<th>j</th>
<th>hi</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>initial values</strong></td>
<td>H</td>
<td>A</td>
<td>C</td>
<td>B</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>D</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>random shuffle</strong></td>
<td>H</td>
<td>A</td>
<td>C</td>
<td>B</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>D</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>14</td>
<td>D</td>
<td>A</td>
<td>C</td>
<td>B</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>6</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>H</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>6</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>L</td>
<td>I</td>
<td>K</td>
<td>J</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>J</td>
<td>I</td>
<td>K</td>
<td>L</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>N</td>
<td>M</td>
<td>O</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
</tbody>
</table>

A B C D E F G H I J K L M N O
Quicksort Cost Analysis – Best case

• The best case is when each partition splits the array into two equal halves

• Overall cost for sorting N items
  – Partitioning cost for N items: N+1 comparisons
  – Cost for recursively sorting two half-size arrays

• Recurrence relations
  – $C(N) = 2 \cdot C(N/2) + N + 1$
  – $C(1) = 0$
Quicksort Cost Analysis – Best case

• Simplified recurrence relations
  – C(N) = 2 C(N/2) + N
  – C(1) = 0

• Solving the recurrence relations
  – N = 2^k
  – C(N) = 2 C(2^{k-1}) + 2^k
    = 2 (2 C(2^{k-2}) + 2^{k-1}) + 2^k
    = 2^2 C(2^{k-2}) + 2^k + 2^k
    = ...
    = 2^k C(2^{k-k}) + 2^k + ... 2^k + 2^k
    = 2^k + ... 2^k + 2^k
    = k * 2^k
    = O(N \log N)
<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>hi</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial values</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>random shuffle</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>14</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
</tbody>
</table>
Quicksort Cost Analysis – Worst case

- The worst case is when the partition does not split the array (one set has no elements)
- Ironically, this happens when the array is sorted!
- Overall cost for sorting N items
  - Partitioning cost for N items: N+1 comparisons
  - Cost for recursively sorting the remaining (N-1) items
- Recurrence relations
  - C(N) = C(N-1) + N + 1
  - C(1) = 0
Quicksort Cost Analysis – Worst case

• Simplified Recurrence relations
  \[ C(N) = C(N-1) + N \]
  \[ C(1) = 0 \]

• Solving the recurrence relations
  \[ C(N) = C(N-1) + N \]
  \[ = C(N-2) + N -1 + N \]
  \[ = C(N-3) + N-2 + N-1 + N \]
  \[ = ... \]
  \[ = C(1) + 2 + ... + N-2 + N-1 + N \]
  \[ = O(N^2) \]
Quicksort Cost Analysis – Average case

• Suppose the partition split the array into 2 sets containing k and N-k-1 items respectively (0<=k<=N-1)

• Recurrence relations
  - \( C(N) = C(k) + C(N-k-1) + N + 1 \)

• On average,
  - \( C(k) = C(0) + C(1) + \ldots + C(N-1) / N \)
  - \( C(N-k-1) = C(N-1) + C(N-2) + \ldots + C(0) / N \)

• Solving the recurrence relations (not required for the course)
  - Approximately, \( C(N) = 2N\log N \)
Quicksort: summary of performance characteristics

Worst case. Number of compares is quadratic.
- \( N + (N - 1) + (N - 2) + \ldots + 1 \sim \frac{1}{2} N^2. \)
- More likely that your computer is struck by lightning bolt.

Average case. Number of compares is \( \sim 1.39 \ N \lg N. \)
- 39% more compares than mergesort.
- But faster than mergesort in practice because of less data movement.

Random shuffle.
- Probabilistic guarantee against worst case.
- Basis for math model that can be validated with experiments.

Caveat emptor. Many textbook implementations go \textit{quadratic} if array
- Is sorted or reverse sorted.
- Has many duplicates (even if randomized!)
Quicksort: practical improvements

Insertion sort small subarrays.
• Even quicksort has too much overhead for tiny subarrays.
• Cutoff to insertion sort for \( \approx 10 \) items.
• Note: could delay insertion sort until one pass at end.

```java
private static void sort(Comparable[] a, int lo, int hi)
{
    if (hi <= lo + CUTOFF - 1)
    {
        Insertion.sort(a, lo, hi);
        return;
    }
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```
QuickSort: practical improvement

• The basic QuickSort uses the first (or the last element) as the pivot value
• What’s the best choice of the pivot value?
• Ideally the pivot should partition the array into two equal halves
Median-of-Three Partitioning

- We don’t know the median, but let’s approximate it by the median of three elements in the array: the first, last, and the center.
- This is fast, and has a good chance of giving us something close to the real median.
QuickSort: practical improvements

Median of sample.

- Best choice of pivot item = median.
- Estimate true median by taking median of sample.
- Median-of-3 (random) items.

\[ \sim \frac{12}{7} N \ln N \text{ compares (slightly fewer)} \]
\[ \sim \frac{12}{35} N \ln N \text{ exchanges (slightly more)} \]

```java
private static void sort(Comparable[] a, int lo, int hi)
{
    if (hi <= lo) return;

    int m = medianOf3(a, lo, lo + (hi - lo)/2, hi);
    swap(a, lo, m);

    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```
Quicksort with median-of-3 and cutoff to insertion sort: visualization
Quicksort Summary

- Quicksort partition the input array to two sub-arrays, then sort each subarray recursively.
- It sorts in-place.
- \(O(N \cdot \log N)\) cost, but faster than mergesort in practice.
- These features make it the most popular sorting algorithm.
Outline

• Quicksort algorithm (review)
• Quicksort Analysis (cont.)
  – Best case
  – Worst case
  – Average case
• Quicksort improvements
• Sorting summary
• Java sorting methods
Sorting Summary

• Elementary sorting algorithms
  – Bubble sort
  – Selection sort
  – Insertion sort

• Advanced sorting algorithms
  – Merge sort
  – Quicksort

• Performance characteristics
  – Runtime
  – Space requirement
  – Stability
Stability

• A sorting algorithm is stable if it preserves the relative order of equal keys in the array
• Stable: insertion sort and mergesort
• Unstable:: selection sort, quicksort
### Sorting summary

<table>
<thead>
<tr>
<th></th>
<th>inplace?</th>
<th>stable?</th>
<th>worst</th>
<th>average</th>
<th>best</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>selection</td>
<td>x</td>
<td></td>
<td>$N^2/2$</td>
<td>$N^2/2$</td>
<td>$N^2/2$</td>
<td>$N$ exchanges</td>
</tr>
<tr>
<td>insertion</td>
<td>x</td>
<td>x</td>
<td>$N^2/2$</td>
<td>$N^2/4$</td>
<td>$N$</td>
<td>use for small $N$ or partially ordered</td>
</tr>
<tr>
<td>shell</td>
<td>x</td>
<td></td>
<td>?</td>
<td>?</td>
<td>$N$</td>
<td>tight code, subquadratic</td>
</tr>
<tr>
<td>merge</td>
<td>x</td>
<td></td>
<td>$N \lg N$</td>
<td>$N \lg N$</td>
<td>$N \lg N$</td>
<td>$N \log N$ guarantee, stable</td>
</tr>
<tr>
<td>quick</td>
<td>x</td>
<td></td>
<td>$N^2/2$</td>
<td>$2N \ln N$</td>
<td>$N \lg N$</td>
<td>$N \log N$ probabilistic guarantee fastest in practice</td>
</tr>
<tr>
<td>3-way quick</td>
<td>x</td>
<td></td>
<td>$N^2/2$</td>
<td>$2N \ln N$</td>
<td>$N$</td>
<td>improves quicksort in presence of duplicate keys</td>
</tr>
<tr>
<td>???</td>
<td>x</td>
<td>x</td>
<td>$N \lg N$</td>
<td>$N \lg N$</td>
<td>$N \lg N$</td>
<td>holy sorting grail</td>
</tr>
</tbody>
</table>
Java system sort method

• `Arrays.sort()` in the `java.util` library represents a collection of overloaded methods:
  – Methods for each primitive type
    • e.g. `sort(int[] a)`
  – Methods for data types that implement `Comparable`
    • `sort(Object[] a)`
  – Method that use a Comparator
    • `sort(T[] a, Comparator<? super T> c)`

• Implementation
  – Quicksort (with 3-way partitioning) to implement the primitive-type methods (speed and memory usage)
  – Mergesort for reference-type methods (stability)
Example

• Sorting transactions
  – Who, when, transaction amount

• Use Arrays.sort() methods

• Implement Comparable interface for a transaction

• Define multiple comparators to allow sorting by multiple keys

• Transaction.java