Quick Sort
And Trees!
12
Quick Sort

• Another divide-and conqueror sort
  ▫ divide: pick a random element $x$ (pivot) and partition into
    ◆ $L: < x$
    ◆ $E: = x$
    ◆ $G: > x$
  ▫ conquer: sort $L$ and $G$
  ▫ combine: join $L$, $E$ and $G$
Pseudo Code

quickSort(S):
    if (S.size() < 2) return
    p = S.last()  // first as pivot
    L = E = G = new list()
    partition(S, p)
    quickSort(L)
    quickSort(G)
    S = L+E+G
QS in action

{7, 2, 9, 4, 3, 7, 1, 6}
Partition on 6
{2, 4, 3, 1} 6  {7, 7, 9}
Partition on 1
{} 1 {2, 4, 3}
Partition on 9
{7, 7} 9 {}
Partition on 3
{2} 3 {4}
Partition on 7
{7} 7 {}

{2,3,4}  {7,7}
{1,2,3,4}  {7,7,9}
{1,2,3,4,6,7,7,9}

Partition (pink)
Assemble (aqua)
Divide
Conquer
MergeSort, Quicksort, etc

- Quicksort does work on way down in recursion
  - Mergesort does work on way up
- Insertion sort does work on way down
  - Selection sort on way up
- Which one is faster Quick or Merge?
Worst-case Running Time

- When the pivot is the min or max
  - one of $L$ or $G$ has size $n - 1$
  - $T(n) = n + (n - 1) + \ldots + 2 + 1 = O(n^2)$
In-place Quick Sort

- instead of three lists partition rearranges the input list
  - $L: [0, l - 1]$
  - $E: [l, r]$
  - $G: [r + 1, n - 1]$
- Recursive calls on $[0, l - 1]$ and $[r + 1, n - 1]$
public int partition(int arr[], int begin, int end) {
    int pivot = arr[end];
    int insertLoc = (begin-1);
    for (int j = begin; j < end; j++) {
        if (arr[j] <= pivot) {
            insertLoc++;
            int swapTemp = arr[insertLoc];
            arr[insertLoc] = arr[j];
            arr[j] = swapTemp;
        }
    }
    int swapTemp = arr[insertLoc+1];
    arr[insertLoc+1] = arr[end];
    arr[end] = swapTemp;
    return insertLoc+1;
}
QuickSort

```java
private void quickSort(int arr[], int begin, int end)
{
    if (begin < end) {
        int partitionIndex = partition(arr, begin, end);
        quickSort(arr, begin, partitionIndex-1);
        quickSort(arr, partitionIndex+1, end);
    }
}
```
# Speed

**Table 1**

<table>
<thead>
<tr>
<th>size</th>
<th>Insertion</th>
<th>Heap</th>
<th>merge (improved)</th>
<th>Quick</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>11</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2000</td>
<td>26</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4000</td>
<td>20</td>
<td>5</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8000</td>
<td>81</td>
<td>10</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>16000</td>
<td>315</td>
<td>17</td>
<td>16</td>
<td>13</td>
</tr>
<tr>
<td>32000</td>
<td>1218</td>
<td>36</td>
<td>32</td>
<td>30</td>
</tr>
<tr>
<td>64000</td>
<td>4605</td>
<td>77</td>
<td>69</td>
<td>59</td>
</tr>
<tr>
<td>128000</td>
<td>19849</td>
<td>161</td>
<td>143</td>
<td>108</td>
</tr>
<tr>
<td>256000</td>
<td>345</td>
<td>294</td>
<td>219</td>
<td></td>
</tr>
<tr>
<td>512000</td>
<td>1128</td>
<td>563</td>
<td>464</td>
<td></td>
</tr>
<tr>
<td>1024000</td>
<td>1973</td>
<td>1191</td>
<td>955</td>
<td></td>
</tr>
<tr>
<td>2048000</td>
<td>3225</td>
<td>2412</td>
<td>1989</td>
<td></td>
</tr>
<tr>
<td>4096000</td>
<td>7577</td>
<td>5191</td>
<td>4148</td>
<td></td>
</tr>
<tr>
<td>8192000</td>
<td>18586</td>
<td>10282</td>
<td>10101</td>
<td></td>
</tr>
<tr>
<td>16384000</td>
<td></td>
<td></td>
<td></td>
<td>17614</td>
</tr>
<tr>
<td>32768000</td>
<td></td>
<td></td>
<td></td>
<td>37291</td>
</tr>
</tbody>
</table>
Quick and Merge

- Quicksort is reliably quicker than merge
- Quicksort does not need extra memory for auxiliary array
Mini Homework

14, 6, 18, 2, 13, 7, 8, 9, 3, 17, 5, 10, 11, 12, 15, 19, 16, 0, 1, 4

For the data above, show all the steps of a quick sort, following the pattern of slide 19. Always choose the last element as the partitioning element.