Today’s topic: Quicksort – fastest known sorting algorithm in practice
✦ Algorithm description
✦ Example
✦ Partitioning in place during Quicksort
✦ Performance analysis

Covered in Chapter 7 of the textbook

Quicksort Description
✦ Quicksort Algorithm:
1. Partition array into left and right sub-arrays such that:
   ◦ Elements in left sub-array < elements in right sub-array
2. Recursively sort left and right sub-arrays
3. Concatenate left and right sub-arrays with pivot in middle
✦ How to Partition the Array:
1. Choose an element from the array as the pivot
2. Move all elements < pivot into left sub-array and all elements > pivot into right sub-array
✦ Pivot? One choice ⇒ use first element in array

QuickSort Example
✦ Sort the array containing:

<table>
<thead>
<tr>
<th>9</th>
<th>1</th>
<th>6</th>
<th>4</th>
<th>1</th>
<th>5</th>
<th>2</th>
<th>5</th>
<th>1</th>
<th>7</th>
</tr>
</thead>
</table>

Partition 4

<table>
<thead>
<tr>
<th>2</th>
<th>5</th>
<th>1</th>
</tr>
</thead>
</table>

Concatenate 9

<table>
<thead>
<tr>
<th>2</th>
<th>16</th>
<th>4</th>
<th>15</th>
<th>2</th>
<th>5</th>
<th>17</th>
<th>1</th>
</tr>
</thead>
</table>

QuickSort Description

Hmmm...seems like we need an extra array for partitioning and concatenating left/right sub-arrays
✦ No!

Algorithm for In Place Partitioning:
1. Swap pivot with last element ⇒ swap pivot and A[N-1]
2. Set pointers i and j to beginning and end of array
3. Increment i until you hit an element A[i] > pivot
4. Decrement j until you hit an element A[j] < pivot
5. Swap A[i] and A[j]
6. Repeat until i and j cross (i exceeds or equals j)
7. Swap pivot (= A[N-1]) with A[i]

On-Board Example: Partition in place:

<table>
<thead>
<tr>
<th>2</th>
<th>16</th>
<th>4</th>
<th>15</th>
<th>2</th>
<th>5</th>
<th>17</th>
<th>1</th>
</tr>
</thead>
</table>

(pivot = A[0] = 9)
Choosing the Pivot

- First Idea: Pick the first element in (sub-)array as pivot
  - What if it is the smallest or largest?
  - What if the array is sorted? How many recursive calls does quicksort make?
- 2nd Idea: Pick a random element
  - Gets rid of asymmetry in left/right sizes
  - But... requires calls to pseudo-random number generator – expensive/error-prone
- Third idea: Pick median (N/2th largest element)
  - Hard to compute without sorting!
  - Compromise: Pick median of three elements

Median-of-Three Pivot

- Find the median of the first, middle and last element
- Takes only O(1) time and not error-prone like the pseudo-random pivot choice
- Less chance of poor performance as compared to looking at only 1 element
- For sorted inputs, splits array nicely in half each recursion
  - Good performance

Quicksort Performance Analysis

- Best Case Performance: Algorithm always chooses best pivot and keeps splitting sub-arrays in half at each recursion
  - $T(0) = T(1) = O(1)$ (constant time if 0 or 1 element)
  - For $N > 1$, 2 recursive calls plus linear time for partitioning
  - $T(N) = 2T(N/2) + O(N)$ (Same recurrence relation as Mergesort)
  - $T(N) = ?$

Quicksort Performance Analysis

- Best Case Performance: Algorithm always chooses best pivot and keeps splitting sub-arrays in half at each recursion
  - $T(0) = T(1) = O(1)$ (constant time if 0 or 1 element)
  - For $N > 1$, 2 recursive calls plus linear time for partitioning
  - $T(N) = 2T(N/2) + O(N)$ (Same recurrence relation as Mergesort)
  - $T(N) = O(N \log N)$
- Worst Case Performance: What is the worst case?
**QuickSort Performance Analysis**

- **Best Case Performance**: Algorithm always chooses best pivot and keeps splitting sub-arrays in half at each recursion
  - $T(0) = T(1) = O(1)$ and $T(N) = 2T(N/2) + O(N)$
  - $T(N) = O(N \log N)$

- **Worst Case Performance**: Algorithm keeps picking the worst pivot – one sub-array empty at each recursion
  - $T(0) = T(1) = O(1)$
  - $T(N) = T(N-1) + O(N)$
  - $T(N) = \ldots$

- Fortunately, average case performance is $O(N \log N)$ (see text for proof)

**Can We Sort Any Faster?**

- Heapsort, Mergesort, and QuickSort all run in $O(N \log N)$ best case running time
- Can we do any better?
- Can Joe Smartypants from Softwareville, USA come up with an $O(N \log \log N)$ sorting algorithm?

**Answer in next class…**

**To do:**
- Finish reading chapter 7
- Start reading chapter 8