Sorting by Comparison algorithms

- **Simple**: Selection Sort
  - (Insertion Sort, Bubble Sort, Shell Sort)
- **Good worst case**: HeapSort, AVLSort, MergeSort
- **Quick**: QuickSort
- **Imaginary**: StrawSort (aka, BrianSort)
- Can we do better?

Selection Sort idea

- Find the smallest element, put it first
- Find the next smallest element, put it second
- Find the next smallest, put it next
- etc.

```c
void SelectionSort (Array a[1..n]) {
    for (i=0, i<n; ++i) {
        Find the smallest entry in Array. 
        Let j be the index of that entry.
        Swap(a[i],a[j])
    }
}
while (other people are coding QuickSort/MergeSort) {
    Twiddle thumbs
}
```

HeapSort: sorting with a priority queue ADT (heap)

Shove everything into a queue, take them out smallest to largest.
**MergeSort**

- **MergeSort** (Array [1..n])
  - Split Array in half
  - Recursively sort each half
  - Merge two halves together

```
Merge (a1[1..n], a2[1..n])
1, i2 = 1
While (1 < i1 < n, i2 < n) {
    if (a1[i1] < a2[i2]) {
        Next is a1[i1]
        i1++
    } else {
        Next is a2[i2]
        i2++
    }
}
```

**MergeSort Running Time**

**QuickSort**

- Pick a "pivot". Divide into less-than & greater-than pivot.
- Sort each side recursively.

**QuickSort Partition**

- Pick pivot:
  - 7 2 8 3 5 9 6
- Partition with cursors:
  - 7 2 8 3 5 9 6
- 2 goes to less-than:
  - 7 2 8 3 5 9 6
- <= >>

**QuickSort Partition (cont’d)**

- 6, 8 swap less/greater-than
  - 7 2 6 3 5 9 8
- 3, 5 less-than
  - 7 6 2 3 5 9 8
- 9 greater-than
  - 7 2 6 3 5 9 8
- Partition done. Recursively sort each side.
Dealing with Slow Quicksorts

- Randomly permute input
  - Bad cases more common than simple probability would suggest. So, make it truly random.
- Pick pivot cleverly
  - “Median-of-3” rule takes Median(first, middle, last) element.
- Choose pivot point randomly!

QuickSelect

- What if we want to find the $k^{th}$ biggest element in an array?
- What if $k = N/2$ (i.e., we want to find the median)?

QuickSelect

<table>
<thead>
<tr>
<th>Pick pivot:</th>
<th>7</th>
<th>2</th>
<th>8</th>
<th>3</th>
<th>5</th>
<th>9</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition array:</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>8</td>
</tr>
</tbody>
</table>

If ($k == partition_index + 1$), we are done!
else recursively call QuickSelect on one subarray.

Could we do better?*

* (no. sorry.)

Worst case time Lower Bound

- How many comparisons does it take before we can be sure of the order?
- This is the minimum # of comparisons that any algorithm could do.

StrawSort (aka BrianSort)
A decision tree to sort list A,B,C

Max depth of the decision tree

- What’s the most leaves a binary tree of height \( h \) could have?
- What’s the shallowest tree with \( L \) leaves?

- A decision tree to sort \( N \) elements must have \( N! \) leaves.
- Any sorting algorithm that uses only comparisons between elements requires at least \( \log(N!) \) comparisons in the worst case!