THINK BIG WE DO

CSC 212
Data Structures & Algorithms

Fall 2022 | Jonathan Schrader

Quick Sort

Housekeeping

Assignment 3 Due

Review Project [MEC]

- Due October 28, 11:59pm

Quick Sort

Divide the array into **two** partitions (subarrays)

- need to pick a *pivot* and rearrange the elements into two partitions

Conquer *Recursively* each half

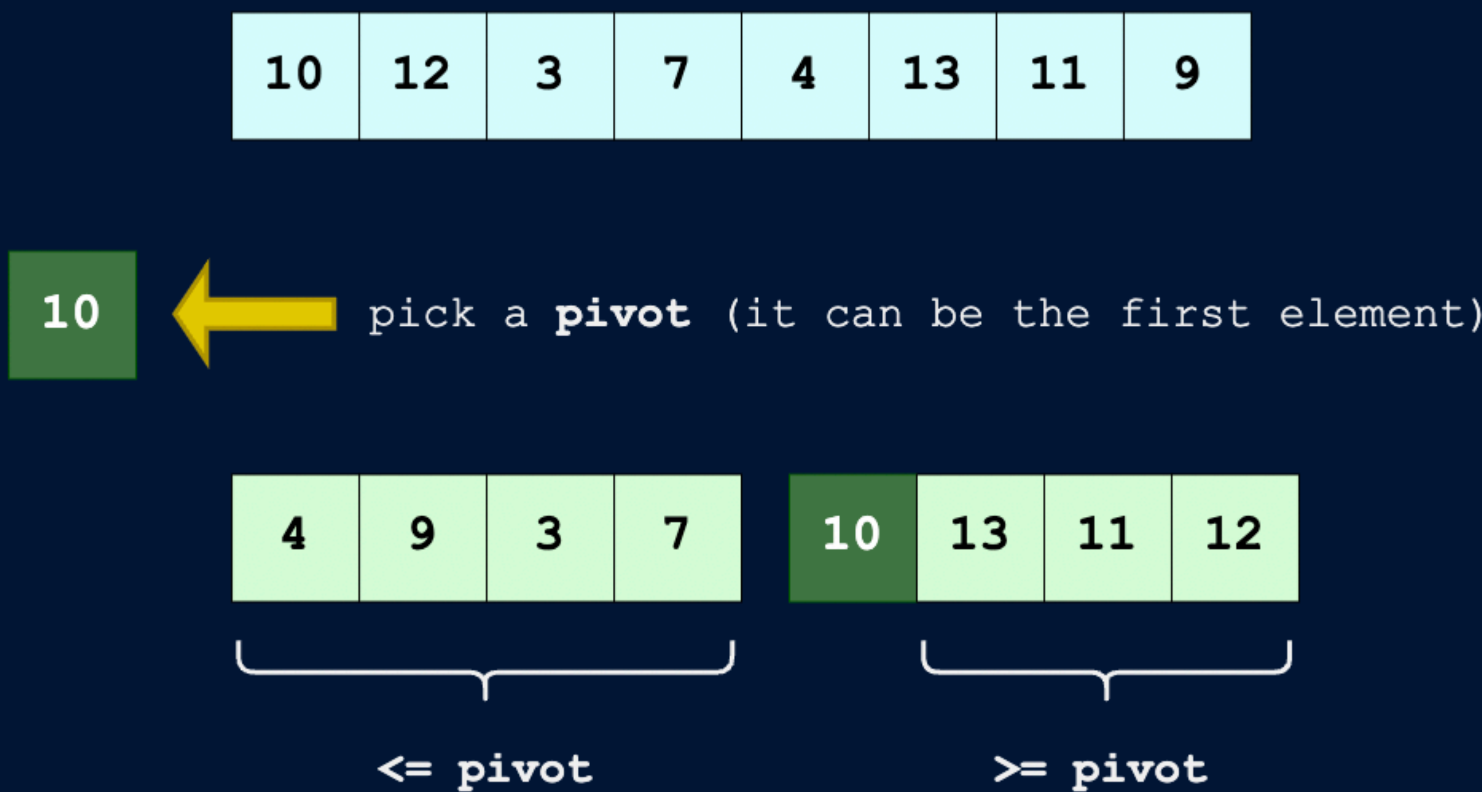
- call Quick Sort on each partition (i.e. solve 2 smaller problems)

Combine Solutions

- there is no need to combine the solutions

```
// pseudocode

if (hi <= lo)
  return;

int p = partition(A, lo, hi);

quicksort(A, lo, p - 1);

quicksort(A, p + 1, hi);
```

Partition


10 12 3 7 4 13 11 9
10
pick a **pivot** (it can be the first element)
4 9 3 7
10 13 11 12
<= pivot
>= pivot

Partition: algorithm


```
while (true)
  scan i left-to-right (while a[i] < a[lo])
  scan j right-to-left (while a[j] > a[lo])
  if i and j crossed then
    break
  swap a[i] with a[j]
swap a[lo] with a[j]
```

Partition: do it yourself

12	1	31	20	10	11	8	2	23	1

- Step 1 | Set the `pivot`…
- Step 2 | Find and evaluate `i` and `j` initial values
- Step 3 | Evaluate all `i` and `j` values, until…
- Step 4 | `i` and `j` cross pathes…

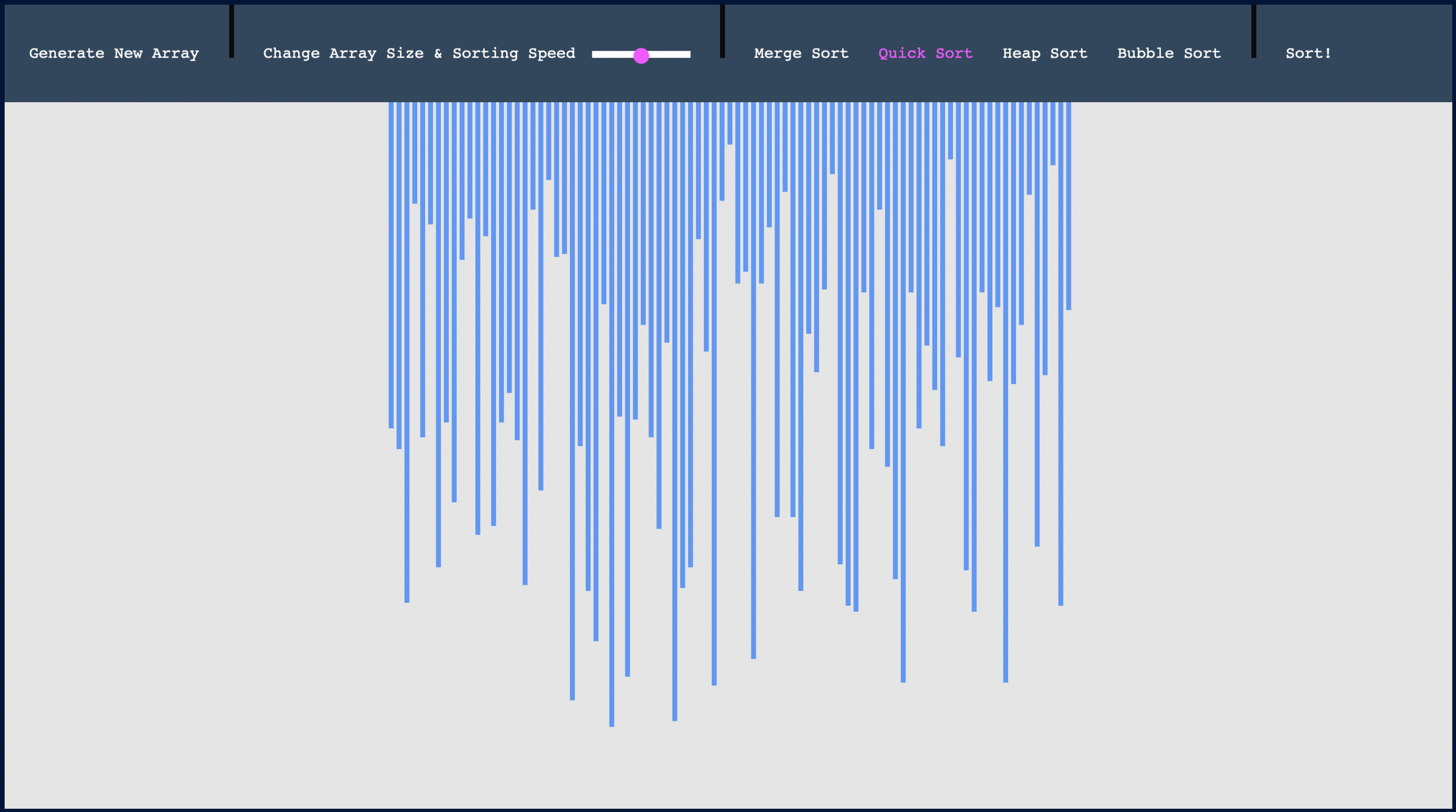
Implementaton

```
int partition (int *A, int lo, int hi)
{
  int i = lo;
  int j = hi + 1;
  while (1) {
    // while A[i] < pivot, increase i
    while (A[++i] < A[lo])
      if (i == hi)
        break;
    // while A[i] > pivot, decrease j
    while (A[lo] < A[--j])
      if (j == lo)
        break;
      // if i and j cross exit theloop
      if(i >= j)
        break;
      // swap A[i] and A[j]
      std::swap(A[i], A[j]);
  }
  // swap the pivot with A[j]
  std::swap(A[lo], A[j]);
  //return pivot's position
  return j;
}
```

```
void r_quicksort(int *A,int lo, int  hi)
{
  if (hi <= lo)
    return;
  int p = partition(A, lo, hi);
  r_quicksort(A, lo, p - 1);
  r_quicksort(A, p + 1, hi);
}
```

```
void quicksort(int *A, int n, int m)
{
  // shuffle the array
  std::random_shuffle(A, A + n);
  // call recursive quicksort
  r_quicksort(A, 0, n - 1);
}
```

Sorting Visualizer



Analysis of Quick Sort

Worst-Case

- input sorted, reverse order, equal elements

$$T(n) = T(n-1) + T(0) + \Theta(n)$$

$$\begin{aligned} T(n) &= T(n-1) + \Theta(1) + \Theta(n) \\ &= T(n-1) + \Theta(n) \\ &= \dots \\ &= \Theta(n^2) \end{aligned}$$

- can shuffle or randomized the array (to avoid the worst-case)

THINK BIG WE DO

Analysis of Quick Sort

Best-Case

- pivot partitions array evenly (almost never happens)

$$
\begin{aligned}
T(n) &= 2T(\frac{n}{2}) + \Theta(n) \\
&= \ldots \\
&= \Theta(n\ log\ n)
\end{aligned}
$$

Analysis of Quick Sort

Average-Case

- analysis is more complex

- Consider a 9-to-1 proportional split
- Even a 99-to-1 split yields same running time
- Faster than merge sort in practice (less data movement)

$$
\begin{aligned}
T(n) &= T(\frac{n}{10}) + T(\frac{9n}{10}) + \Theta(n) \\
&= \dots \\
&= \Theta(n \; log \; n)
\end{aligned}
$$

THINK BIG WE DO

Comments on Quick Sort

Properties

- it is **in-place** but not **stable**
- benefits substantially from code tuning

Improvements

- use insertion sort for small arrays
  - avoid overhead on small instances (~10 elements)
- median of 3 elements
  - estimate true median by inspecting 3 random elements
- three-way partitioning
  - create three partitions < pivot, == pivot, >pivot

DL 1263	Atlanta	6:00 am	Departed
DL 2225	Atlanta	12:48 pm	On time
WN 1138	Baltimore	6:05 am	Departed
WN 846	Baltimore	9:20 am	Departed
WN 3020	Baltimore	11:20 am	On time
WN 6296	Baltimore	12:35 pm	On time
AA 1703	Charlotte	6:17 am	Departed
AA 632	Charlotte	8:07 am	At 9:45 am
AA 1981	Charlotte	11:01 am	On time
AA 5550	Charlotte	11:54 am	On time
WN 6240	Chicago - MDW	5:55 am	Departed
WN 3420	Chicago - MDW	8:45 am	Departed
AA 3410	Chicago - ORD	7:02 am	Departed
UA 3615	Chicago - ORD	7:30 am	Departed
DL 2273	Detroit	5:30 am	Departed
DL 305	Detroit	10:40 am	On time
DL 5090	Detroit	12:32 pm	On time
WN 6247	Fort Lauderdale	8:30 am	Departed
UA 4894	New York/Newark	6:15 am	Departed
B6 475	Orlando	6:15 am	Departed
WN 28	Orlando	6:55 am	Departed
AA 489	Philadelphia	6:00 am	Departed
AA 1735	Philadelphia	8:02 am	Departed
AA 774	Philadelphia	10:51 am	On time
WN 6235	Tampa	7:05 am	Departed
AC 7379	Toronto	11:50 am	On time
AA 5202	Washington - DCA	6:14 am	Departed
WN 2640	Washington - DCA	8:45 am	Departed
AA 4280	Washington - DCA	8:49 am	At 10:20 am
AA 5524	Washington - DCA	11:46 am	At 2:35 pm
AA 4424	Washington - DCA	1:38 pm	On time
UA 6208	Washington - IAD	6:00 am	Departed

Sorting Algorithms

	Best-Case	Average-Case	Worst-Case	Stable	In-place
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	No	Yes
Insertion Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	Yes	Yes
Merge Sort	$\Theta(n\ log\ n)$	$\Theta(n\ log\ n)$	$\Theta(n\ log\ n)$	Yes	No
Quick Sort	$\Theta(n\ log\ n)$	$\Theta(n\ log\ n)$	$\Theta(n^2)$	No	Yes

Empirical Analysis

Running time estimates:

- Home PC executes 10^8 compares/second.
- Supercomputer executes 10^{12} compares/second.

	insertion sort (N^2)			mergesort (N log N)			quicksort (N log N)		
computer	thousand	million	billion	thousand	million	billion	thousand	million	billion
home	instant	2.8 hours	317 years	instant	1 second	18 min	instant	0.6 sec	12 min
super	instant	1 second	1 week	instant	instant	instant	instant	instant	instant

Lesson 1. Good algorithms are better than supercomputers.

Lesson 2. Great algorithms are better than good ones.

https://www.cs.princeton.edu/courses/archive/spring18/cos226/lectures/23Quicksort.