Quicksort

Technique: Divide-and-conquer (split, solve, combine)

Idea:

- Partition:
  - pick an element called 'pivot'
  - move elements around s.t. at the end:

_ ≤ pivot	pivot	_ > pivot

    - Intermediate state:

_ ≤ pivot	_ > pivot	unprocessed	pivot

- Call Quicksort on the two subarrays.

Execution of the Partition function with rightmost item as pivot.

0	1	2	3	4	5	6	7	8	9
27	90	70	60	20	40	45	80	10	**50**
27	90	70	60	20	40	45	80	10	**50**
27	90	70	60	20	40	45	80	10	**50**
27	90	70	60	20	40	45	80	10	**50**
27	90	70	60	20	40	45	80	10	**50**
27	**20**	70	60	**90**	40	45	80	10	**50**
27	20	**40**	60	90	**70**	45	80	10	**50**
27	20	40	**45**	90	70	**60**	80	10	**50**
27	20	40	45	90	70	60	80	10	**50**
27	20	40	45	**10**	70	60	80	**90**	**50**
27	20	40	45	10	**50**	60	80	90	**70**

```
Partition(A, start, end){
  pivot = A[end]
  t = start;
  for(j=start; j<=end-1; j++){
    if (A[j]<=pivot){
      A[j] <-> A[t]
      t++;
    }
  }
  A[t]<->A[end];
  return t;
}
```

```
Quicksort(A, start, end){
  if (start >= end) return;
  pIndex = Partition(A,start,end);
  Quicksort(A, start, pIndex-1);
  Quicksort(A, pIndex+1,end);
}
```

The pivot is in its final place (in the sorted array).

The green (≤) section increases by swapping the element with the leftmost, larger one. If no purple section, it will swap the same element in place (here 27<->27).

Space complexity: $\Theta(1)$

Time complexity: **best, average** (when random): **$\Theta(N\lg N)$, worst** (when sorted)**: $\Theta(N^2)$**

- Recurrence formulas: General: $T(n) = T(x) + T(n-1-x) + \Theta(n)$
  - Best (balanced partition): $T(n) = 2\,T(n/2) + \Theta(n) \Rightarrow \Theta(n\lg n)$ (Simplified formula.)
  - Worst (unbalanced): $T(n) = T(n-1) + \Theta(1) + \Theta(n) \Rightarrow T(n) = T(n-1) + \Theta(n) \Rightarrow \Theta(n^2)$

Array after each call to the Partition function (shows the sorting):

0	1	2	3	4	5	6	7	8	9	Quicksort(A,start,end)	Partition returns
27	90	70	60	20	40	45	80	10	50	Original array	
27	20	40	45	10	**50**	60	80	90	70	Quicksort(A,0,9)	5
10	20	40	45	27	50	60	80	90	70	Quicksort(A,0,4)	0
10	20	40	45	27	50	60	80	90	70	Quicksort(A,0,-1)	Not called (basecase)
10	20	**27**	45	40	50	60	80	90	70	Quicksort(A,1,4)	2
10	20	27	45	40	50	60	80	90	70	Quicksort(A,1,1)	Not called (basecase)
10	20	27	**40**	45	50	60	80	90	70	Quicksort(A,3,4)	3
10	20	27	40	45	50	60	80	90	70	Quicksort(A,3,2)	Not called (basecase)
10	20	27	40	45	50	60	80	90	70	Quicksort(A,4,4)	Not called (basecase)
10	20	27	40	45	50	60	**70**	90	80	Quicksort(A,6,9)	7
10	20	27	40	45	50	60	70	90	80	Quicksort(A,6,6)	Not called (basecase)
10	20	27	40	45	50	60	70	**80**	90	Quicksort(A,8,9)	8
10	20	27	40	45	50	60	70	80	90	Quicksort(A,8,7)	Not called (basecase)
10	20	27	40	45	50	60	70	80	90	Quicksort(A,9,9)	Not called (basecase)

Variations (improve performance)

- Pick pivot as **median of three**: first, middle, last – this fixes the worst case of a sorted array.
    - Work-out an example: [27, 90, 70, 60, 20, 40, 45, 80, 10, 50]
    - Discuss code changes
- **Random Pivot:** element from a random index.
- Call **insertion sort for small problem sizes.**

Properties:

- not stable – build example – self-check exercise
- not adaptive

Background needed: Θ, O, recurrences, recursion,

Terminology, notations, conventions:

- pivot, divide-and-conquer
- Show the ≤ and > subarrays by marking the last element in the subarray:
    - Example: 27, 20, 40, 45, 90, 70, 60, 80, 10, **50**

Resources:

- https://www.youtube.com/watch?v=COk73cpQbFQ (Youtube (mycodeschool))
    - Subtitles, code at the end, shows recursive calls order and arguments.

- CLRS method (different index names, and updates the last index of the smaller elements after the swap)

Practice:

1. Build extreme cases: give an array and work-out the algorithm:
   a. All elements are smaller than the pivot
   b. All elements are larger than the pivot
   c. All elements are equal to the pivot
   d. There is no pivot

<u>Selection problem</u>: Return <u>the k-th element in the array</u> (e.g. the 7-th smallest item)

- Similar to Quicksort, but after partition, <u>keep only the subarray that has the k-th element</u>.
- Best and average O(N) , worst $O(N^2)$
  - Simplistic (k is not factored-in):
    - Best: T(n,k) = T((n-1)/2, k)+ Θ(n) => Θ(n)
    - Worst: T(n,k) = T((n-1), k)+ Θ(n) => $\Theta(n^2)$
- Workout example: find 1st, and 7th in array: [17, 90, 70, 30, 60, 40, 45, 80, 10, 35]

Time complexity

Quicksort recurrence formula:

- General: T(n) = T(x) + T(n-1-x) + Θ(n)
- Best (balanced partition): T(n) = 2 T((n-1)/2) + Θ(n) => Θ(nlgn)
- Average - Intuition: Alternate worst with best:

  T(n) = T(n-1) + Θ(1) + Θ(n) = 2T((n-1-1)/2) + Θ(n-1) + Θ(n) = 2T((n-2)/2) + Θ(n)

  (This derivation is not identical to CLRS, but the logic is the same. We have -1 because the pivot is not part of the subproblems.)
- Worst (unbalanced): T(n) = T(n-1) + Θ(1) + Θ(n) => T(n) = T(n-1) + Θ(n) => $\Theta(n^2)$

See worked-out example for Median-of-three on the next page.

Median-of-three

```
Med_3_Partition(A, s,t)
     med_idx <- index of median of A[s], A[t], A[(s+t)/2]
     A[med_idx] <-> A[t]
     continue with the regular partition code
```

Median-of-three example shows data behavior in the new partition method:

0	1	2	3	4	5	6	7	8	Description
5	1	3	7	**9**	2	1	8	**0**	Find median of 5,9,0: 5
0	1	3	7	9	2	1	8	**5**	swap 5<->0 (put median last)
0	1	3	2	1	**5**	9	8	7	Normal partition
0	1	**3**	2	**1**					Find median of 0,3,1 : 1
0	1	3	2	**1**					swap 1 <-> 1
0	1	**1**	2	3					Normal partition
0	**1**								Find median of 0,0,1 : 0
1	**0**								Swap 0 <-> 0
0	1								Normal partition
			2	**3**					Find median of 2,2,3: 2
			3	**2**					Swap 3 <->2
			2	3					Normal partition
						9	**8**	**7**	Find median of 9,8,7: 8
						9	**7**	**8**	Swap 8 <-> 7
						7	**8**	9	Normal partition

Last update: 4/5/18