Updated: 2/22/2025

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	> pivot	unprocessed

- Call Quicksort on the two subarrays.

Execution of the Partition function with rightmost item as pivot. (See https://visualgo.net/en/sorting)

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

```
int Partition(int *A,int start,int end){
  pivot = A[start]
  big1idx = start+1; //1st purple (> pivot)
  for(j=start+1; j<=end; j++){
    if (A[j]< pivot){
      A[j] <-> A[big1idx]  //swap
      big1idx++;
    }
  }
  A[big1idx-1] <-> A[start] //swap
  return big1idx-1
}
```

```
void Quicksort(int *A,int start,int 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 current element (at index j) with the leftmost, larger one (at index big1idx). If no purple section, it will swap the same element in place (here 27<->27).

Space complexity: in place (for code) + O(lgN) for frame stack with [Sedgewick's tail recursive version]
(Else, space for frame stack (due to recursion): O(N), (average and best: Θ(lgN), worst: O(N))
Time complexity: **best, average** (when random): **Θ(NlgN), worst** (when sorted): **Θ(N²)**

- 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	

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]
    - See worked-out example for Median-of-three on the next page.
    - Discuss code changes
- **Random Pivot:** element from a random index.
- Call **insertion sort for small problem sizes.**
- **Sedgewick's tail recursive – reduces stack size to O(logN).** See also on Wikipedia: **"**From a bit complexity viewpoint, variables such as *lo* and *hi* do not use constant space; it takes $O(\log n)$ bits to index into a list of n items. Because there are such variables in every stack frame, quicksort using Sedgewick's trick requires $O((\log n)^2)$ bits of space.**"**

Properties:

- not stable – build example
- 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: **50**, 20, 40, 45, 90, 70, 60, 80, 10, 27

Resources:

- https://visualgo.net/en/sorting (or https://visualgo.net/en) – good to visualize the Partition method. Uses first element as the pivot (in this document we use the last element).
- https://www.youtube.com/watch?v=COk73cpQbFQ (Youtube (mycodeschool))
  - Explains recursion tree well
  - 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)
- Wikipedia

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

Selection problem - QuickSelect

Return the k-th element in the array (e.g. the 7-th smallest item)

- Similar to Quicksort, but after partition, keep only the subarray that has the k-th element.
- 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]

```
int QuickSelect (int k, int[] A, int start, int end){
  int pIndex = Partition(A,start,end);
  if ( (k-1)==pIndex )   // k-1 because indexes start at 0
    return A[pIndex];
  else if ( (k-1)<pIndex )
    return QuickSelect(k, A, start, pIndex-1);
  else
    return QuickSelect(k, A, pIndex+1,end);
}
```

Extra materials

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:

  $T(n) = T(n-1) + \Theta(1) + \Theta(n) = 2T(\ (n-1-1)/2\ ) + \Theta(n-1) + \Theta(n) = 2T(\ (n-2)/2\ ) + \Theta(n)$

  (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): $T(n) = T(n-1) + \Theta(1) + \Theta(n) \Rightarrow T(n) = T(n-1) + \Theta(n) \Rightarrow \Theta(n^2)$

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[s]
     continue with the regular partition code
```

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