How to make mergesort quicker?
Parallelize mergesort(L) and mergesort(R)!
We can even parallelize merge itself. How?

A = [1, 2, 3, 4, 9, 13], N = 6 (length of A)
B = [5, 6, 7, 8, 10, 11], M = 6 (length of B)
Let's assume we are using 2 threads. N+M = 12
Each thread should merge 6 elements.

What should thread 1 do?
What should thread 2 do?

How to define such partition?
A = [1, 2, 3, 4 // 9, 13]
B = [5, 6 // 7, 8, 10, 11

How can we determine such partition efficiently? Use _____.

3-way partitioning quicksort

```
qsort (List S) {

   if (|S| <= 1) return S;                    // Note that S could be
empty

   v = element of S;                          // Choose pivot

   // Using set notation for lists

   List L = { x in  S - {v} | x < v }

   List R = { x in  S - {v} | x > v }

   return (qsort(L) + E {all elements equal to v} + qsort (R));

}
```

How to implement above algorithm?

We maintain 3 partitions (L: strictly smaller than v, E: same as v, R: strictly greater than v) in an array. Thus, we need to maintain 3 pointers:

- lt:
- i:
- gt:

Complete the code below.

```
public static void quicksort(int[] arr) {
    quicksort(arr, 0, arr.length - 1);
}

private static void quicksort(int[] arr, int low, int high) {
    if (low >= high) return;
    int pivot = arr[low]; // choose first element as pivot (this can be optimized)

    int lt = low;    // arr[low..lt-1] < pivot
    int gt = high;   // arr[gt+1..high] > pivot
    int i = low;     // arr[lt..i-1] == pivot
    while (i <= gt) {
        if (arr[i] < pivot) { // i should be part of L (less than pivot region)
            // implement me


        } else if (arr[i] > pivot) { // i should be part of R (greater than pivot region)
            // implement me


        } else { // arr[i] == pivot
            // implement me

        }
    }
```

```
        // Recursively sort left and right partitions
        quicksort(arr, low, lt - 1);
        quicksort(arr, gt + 1, high);
    }

    private static void swap(int[] arr, int i, int j) {
        int tmp = arr[i];
        arr[i] = arr[j];
        arr[j] = tmp;
    }
```

Is this in-place?
Is this stable?

Time complexity?

- `n` = size of array
- `m` = size of `< pivot` region
- `k` = size of `> pivot` region
- `p` = size of `== pivot` region

T(n) = _____

3-way partitioning performance also depends on _____.