Show a trace of mergesort operating on the following array (length = 20)

\[52, 28, 6, 9, 31, 15, 2, 45, 49, 24, 32, 7, 3, 8, 19, 16, 41, 11, 20, 30\]

Use the recursive version of mergesort and produce a trace in the style of example 1, lecture notes 15. Do not give an internal trace of "merge". Assume that the recursion goes down to array length 3.

Problem 2

Show a trace of quicksort operating on the same array as in problem 1.

Use the in-place version of quicksort and produce a trace in the style of example 2, lecture notes 16. Do not give an internal trace of "partition". Assume that the recursion goes down to array length 3.

Problem 3

Consider the problem of finding the kth smallest element in an array.

This problem can be solved by an algorithm of a similar structure to quicksort as follows. We will use a recursive method findKth(a,l,u,k) which looks between indices l and u for the kth smallest element of a.

We can write findKth as follows:

```
findKth(a,l,u,k) {
    if (k < l || k > u) raise error; // invalid value of k
    if (l==u) return a[l]; // base case
    m = partition(a,l,u);
    if (???) return a[m];
    else if (???)
        return findKth(a,???,???,??);
    else
        return findKth(a,???,???,??);
}
```

The partition function here is the same one as in the usual quicksort. It returns the index in a where the pivot ends up.

Fill in the question marks in the code above.
Honors Problem

Design an efficient algorithm to determine whether two unsorted sets of $m$ and $n$ integers are disjoint. Assume that $m < n$. Full credit will be given for an algorithm that runs in time $O(n \log m)$; half credit will be given for an algorithm that runs in time $O(n \log n + m \log m)$ (which is the same thing as $O(n \log n)$).