Exercise Sheet, Week 8

Question 1. Use *merge sort* to sort the following sequences:

$$\langle 2, 7, 2, 3, 8, 5, 4, 1 \rangle \qquad \langle 1, 2, 3, 4, 5, 6, 7 \rangle$$

Question 2. Use *quicksort* to sort the first sequence of Question 1 with the following pivot-selection strategies:

(a) The leftmost element.

(b) The middle element.

(c) A random element given by the following "random" sequence of positions:

$$6, 0, 2, 0, 1, 4, 1, 3, 1, 2, 4, 1, 0, 2, 3, 5, 1, 2, 0, 0, 0, 0, 0, 0, \ldots$$

(Use `mod` if the position is out of scope of the part of the array which you are sorting.)

Question 3. What would be the time complexity of the modified *merge sort* which calls itself recursively only as long as the size is n or $\frac{n}{2}$ and, for smaller inputs, it calls *selection sort* instead?

Question 4. Instead of recursion, use stacks to implement *quicksort*. You can use function `partition` as in the lecture. *Hint:* It might help to use the following class to remember which partitions are to be sorted:

```
class Interval {
    int left;
    int right;
}
```

```
void quicksort(int[] arr) {
    s = new Stack<Interval>();
    s.push(new Interval(0, arr.length-1));

    while (s.notEmpty()) {








    }
}
```

Question 5. Write code for `int partition(arr, left, right)` as we described in in the lecture, that is, `partition` rearranges the array so that

- the small entries are stored on positions `left, left+1, left+2, ..., pivot_index-1`,
- pivot is stored on position `pivot_index` and
- the large entries are stored on `pivot_index+1, pivot_index+2, ..., right`.

The return value is the position of the pivot in `arr`, that is, `pivot_index`.

```java
int partition(int[] arr, int left, int right) {
    old_pivot_index = choosePivot(arr, left, right);
    tmpLE = new int[right-left+1];
    tmpG  = new int[right-left+1];

    pivot_index = -1;
    a = 0;
    b = 0;

    for (int i=0; i<right-left+1; i++) {
        if (arr[left+i] <= arr[old_pivot_index]) {






        } else {


        }
    }

    // Copy tmpLE and tmpG into arr:






    // Return the location of pivot in arr:
    return ???;
}
```

(Bonus) Question 6. Instead of recursion, use stacks/queues to implement *merge sort*. (You can use function `merge` as in the lecture.)

(Bonus) Question 7. Write partitioning for quicksort which "in-place", that is, it does not need to allocate any extra memory. *Remark:* This can break stability of quicksort.