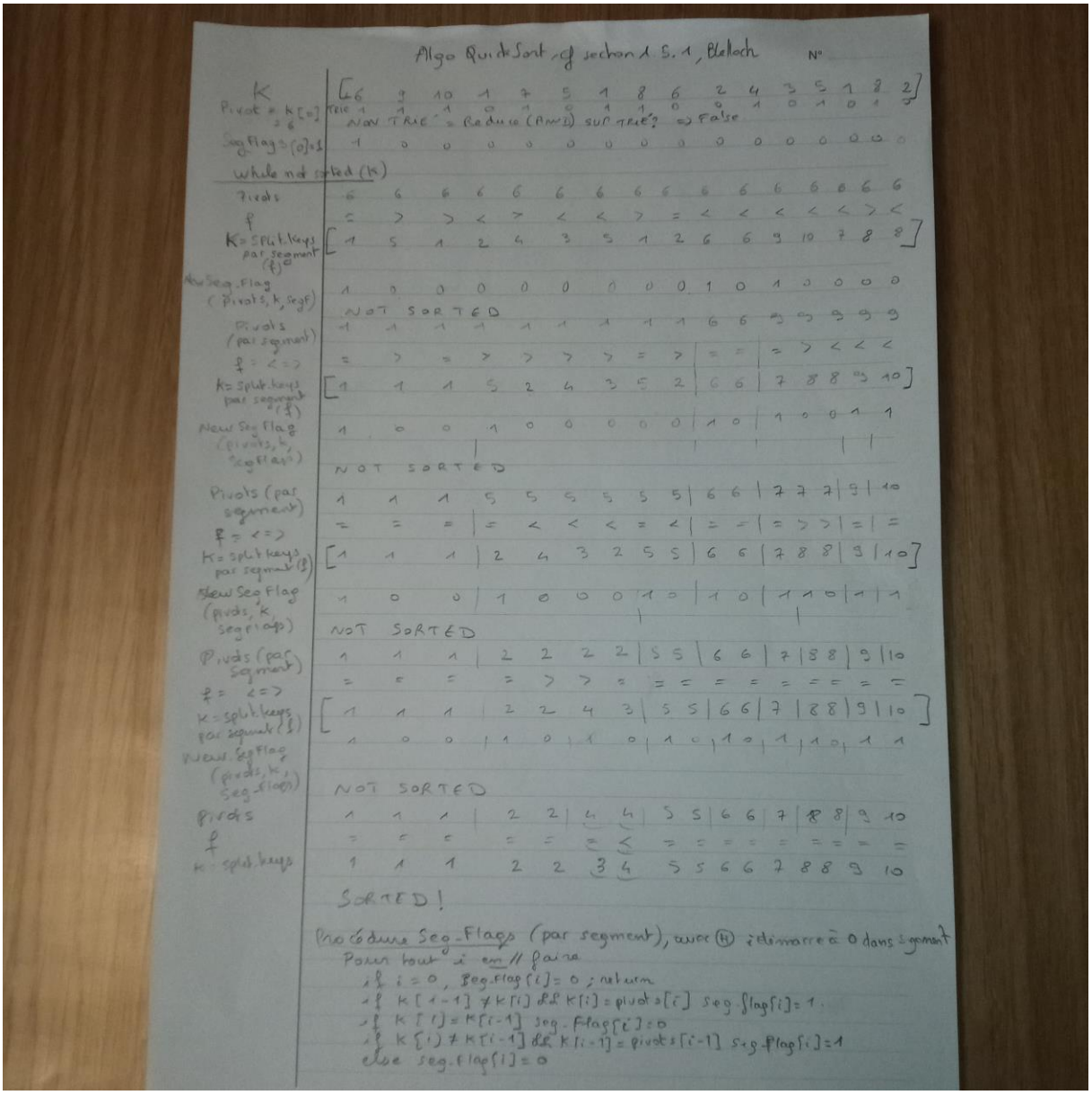
Exercice 1

Application of the Prefix, or Scan, on segments. Usage for quick sort. Read carefully PDF of Blelloch' chapter, section 1.5

Apply the proposed quick sort algorithm, without trying to implement the various scan/prefix segmented primitives, on an array of 16 entries and sort them in increasing order.

Evaluate its complexity in time: expected number of steps: O(log n); each step has a series of scan/prefix operations, each costs O(logn) // time.



Exercice 2

Now, try to devise how to understand scan/prefix on segments. For this, study carefully equation 1.5 of that same PDF.

The segmented scans satisfy the recurrence:

$$x_i = \begin{cases} a_0 & i = 0 \\ \begin{cases} a_i & f_i = 1 \\ (x_{i-1} \oplus a_i) & f_i = 0 \end{cases} & 0 < i < n \end{cases} \tag{1.15}$$

So, it is easy to understand that depending on the value stoded in seg flag (corresponds to f in 1.15 above), the scan/prefix operation will start again with x_i=a_i when a new segment is encountered. Otherwise, the current accumulated value in x_{i-1} will be combined with the binary operator to the current a_i value.
We can for instance apply segmented scan with the copy(a,b)=a operator. It is used in the step

2 of quicksort; in each segment, the value x_i==b, a=first element of the current segment as pivot, and the goal is to propagate the pivot to the end of the segment.

The segmented scan will also be very useful for the split operation see below Exercice3.

Exercice 3

Propose an implementation for the SPLIT on segments, that for each segment split in three sub parts, on the left hand side, elements whose value is less than the pivot of that segment, in the middle, elements that are equal to the pivots, then, on the right, elements that are greater than the pivot

The operator will apply on data in the form of a vector of 3 integers associated to each array element. In each vector, it corresponds to the number of elements including the current one, that are < than the pivot, that are == to the pivot, and that are > to the pivot.

Eg, on such a segment, indexes start at 1
Position
1 2 3 4 5 6 7 8
Keys : 4 2 5 4 1 3 7 2
Pivots:4 4 4 4 4 4 4 4
F := < > = < < > <
Vect :0 1 1 1 2 3 3 4
1 1 1 2 2 2 2 2
0 0 1 1 1 1 2 2
What is missing is the number of elements in total that are < to the pivot, = to the pivot, > than the pivot, so to know in advance the size of each sub segment (each new segment).
We can copy the vector of the last element (4,2,2), name it OFFSET from right to the left.
Knowing this, for instance, the first element Keys[0] (4) is to be moved as first element in the subsegment containing values = to the pivot, so at position 4 in the current segment.
We can use the indexes to move elements to the right place according to each Vect stored in position i,
If F[i] = "<", then use Vect[0] as index in segment for K[i]
If F[i] = "=", then use Vect[1] as index + offset of elements < to the pivot (OFFSET[0]) in segment for K[i]
If F[i] = ">", then use Vect[2] as index + offset of elements < to the pivot (OFFSET[0])+ offset of elements = to the pivot (OFFSET[1])in segment for K[i]

From Keys: 4 2 5 4 1 3 7 2
Keys : 4 moves to position in current segment: Offset[0]=4+1 = 5
2 moves to position in current segment: 1
5 moves to position in current segment Offset[0]+Offset[1]+1 = 4+2+1=7
4 moves to position in current segment: Offset[0]=4+2=6
1 moves to position in current segment: 2
3 moves to position in current segment: 3
7 moves to position in current segment: Offset[0]+Offset[1]+2 = 4+2+2=8
2 moves to position in current segment: 4
At the end, we indeed get
New Keys: 2 1 3 2 4 4 5 7