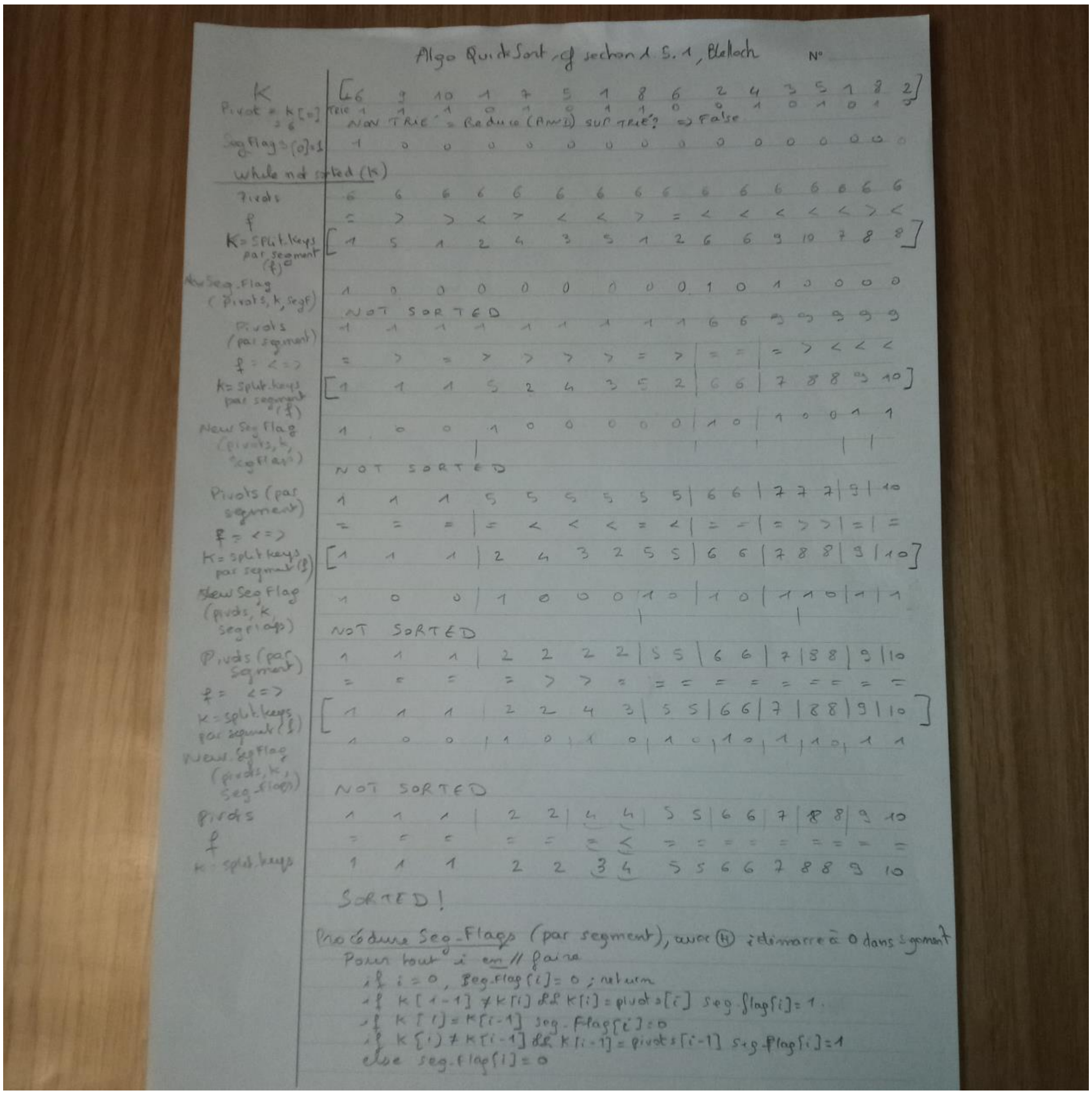
Exercice 1

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. So the total is O(log^2 n) with a probability less than 1.



Algo QuickSort, cf section 1 S.1, Blelloch   N°

```
K                         [6  9  10  1  7  5  1  8  6  2  4  3  5  1  8  2]
Pivot = K[0] = 6     Trie  1  1  1  0  1  0  1  1  0  0  1  0  1  0  1  0
                      NON TRIE = Reduce (AND) sur TRIE ? => False
Seg Flags[0]=1        1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
while not sorted (K)
Pivots                6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6
f                     =  >  >  <  >  <  <  >  =  <  <  <  <  <  >  <
K = Split.Keys       [1  5  1  2  4  3  5  1  2  6  6  9  10 7  8  8]
 par segment (f)
New Seg.Flag          1  0  0  0  0  0  0  0  0  1  0  1  0  0  0  0
 (Pivots, K, SegF)
                      NOT SORTED
Pivots                1  1  1  1  1  1  1  1  1  6  6  9  9  9  9  9
 (par segment)
f = <=>               =  >  =  >  >  >  >  =  > | =  = | =  >  <  <  <
K = Split.keys       [1  1  1  5  2  4  3  5  2 | 6  6 | 7  8  8  9  10]
 par segment (f)
New Seg Flag          1  0  0  1  0  0  0  0  0 | 1  0 | 1  0  0  1  1
 (pivots, K, segFlags)
                      NOT SORTED
Pivots (par segment)  1  1  1  5  5  5  5  5  5 | 6  6 | 7  7  7 | 9 | 10
f = <=>               =  =  = | =  <  <  <  =  < | =  = | =  >  > | = | =
K = split keys       [1  1  1 | 2  4  3  2  5  5 | 6  6 | 7  8  8 | 9 | 10]
 par segment (f)
New Seg Flag          1  0  0 | 1  0  0  0  1  0 | 1  0 | 1  1  0 | 1 | 1
 (pivots, K, segFlags)
                      NOT SORTED
Pivots (par segment)  1  1  1 | 2  2  2  2 | 5  5 | 6  6 | 7 | 8  8 | 9 | 10
f = <=>               =  =  =   =  >  >  =   =  =   =  =   =   =  =   =   =
K = split keys       [1  1  1 | 2  2  4  3 | 5  5 | 6  6 | 7 | 8  8 | 9 | 10]
 par segment (f)
New Seg Flag          1  0  0 | 1  0 | 1  0 | 1  0 | 1  0 | 1 | 1  0 | 1  1
 (pivots, K, seg-flags)
                      NOT SORTED
Pivots                1  1  1 | 2  2 | 4  4 | 5  5 | 6  6 | 7 | 8  8 | 9  10
f                     =  =  =   =  =   =  <   =  =   =  =   =   =  =   =  =
K = split keys        1  1  1   2  2   3  4   5  5   6  6   7   8  8   9  10
                      SORTED !
```

```
Procédure Seg_Flags (par segment), avec (H) i démarre à 0 dans segment
  Pour tout i en // faire
    if i = 0, Seg-Flag[i] = 0 ; return
    if K[i-1] ≠ K[i] && K[i] = pivots[i]  seg_flag[i] = 1
    if K[i] = K[i-1]  seg_Flag[i] = 0
    if K[i] ≠ K[i-1] && K[i-1] = pivots[i-1]  seg_flag[i] = 1
    else seg_flag[i] = 0
```

Below, you find a more appropriate procedure for determining the new seg flag array as opposed to the piece of code that is displayed in the image above. It is used after the split operation that has permuted elements of array K within each segment, and the goal of it is to indicate within each segment what are the up to 3 new segments.

This procedure that can run in parallel on the whole array K, considering each processor has access to the previous array seg_flag (the one that was used to permute elements within each segment) and the pivot values per segment array. The goal of it is to obtain an array named new_seg_flag that will then be considered as seg_flag in the next step.

```
Procedure compute_new_seg_flag:

    For all i in parallel do
        if (seg_flag[i] == 1) {
            new_seg_flag[i]=1;
            // means value at K[i] is at the start of a new segment
```

```
                return; // implication: only processors in charge of values **not** at the start of the (old) segment are continuing to run the tests, so we can test K[i-1] in the sequel, as it belongs to our (old) segment
            }
            if (K[i]==pivots[i]){
                new_seg_flag[i]=(K[i-1]==pivots[i])?0:1; //K[i] is or not the 1st pivot
                return; // implication: only processors in charge of values that are **not** equal to the pivot in their (old and new) segment are continuing to run the tests
            }
            if (seg_flag[i-1]==1) { // K[i] is not the 1st in the old (and in the new) segment
                new_seg_flag[i]==0;
                return; // it works as K[i] not a pivot, and we are sure i is such that i-1 entry exists and is part of the (old) segment
            }
            else {//last case to consider: is K[i] the start of a new seg (notice that it will have with values > pivot)?
                new_seg_flag[i]=(K[i-1]==pivots[i])?1:0;
            }
```

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:

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

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 (small exercise is to adapt the given rules below to the case where indexes of segment start at 0)
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 := < > = < < > <

$$\text{Vect}\ \begin{pmatrix}0\\1\\0\end{pmatrix}\begin{matrix}1&1&1&2&3&3\\1&1&2&2&2&2\\0&1&1&1&1&2\end{matrix}\begin{pmatrix}4\\2\\2\end{pmatrix}$$

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] (that is value 4) is to be moved as first element in the subsegment containing values equal 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, like this:
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 array: 2 1 3 2 4 4 5 7