1 Selection in a list

Question 1

a) Let \( L \) be a list containing \( n \) objects colored either in blue or red. Design an efficient EREW algorithm that separates the blue elements from the red elements (i.e. that builds a new list containing only the blue elements).

2 Mystery Procedure

We define the following two operators for a table \( A = [a_0, a_1, \ldots, a_{n-1}] \) of \( n \) integers:

- \( \text{Prescan}(A) \) returns the table: \([0, a_0, a_0 + a_1, a_0 + a_1 + a_2, \ldots, a_0 + a_1 + \ldots + a_{n-2}]\)
- \( \text{Scan}(A) \) returns the table: \([a_0, a_0 + a_1, a_0 + a_1 + a_2, \ldots, a_0 + a_1 + \ldots + a_{n-1}]\)

These two operators can be computed in \( O(\log n) \) time on P-RAM EREW.

Given a table \( \text{Flags} \) we define the following \text{Split} procedure:

**Algorithm 1: Mystery Procedure 1**

```python
def Split(A, Flags):
    Iup ← n - Reverse(Scan(Reverse(Flags)));
    Idown ← Prescan(1 - Flags);
    for i = 1 to n do in parallel
        if Flags(i) then
            Index[i] ← Iup[i]
        else
            Index[i] ← Idown[i]
    Result ← Permute(A, Index);
    return Result
```

The names of the different functions are relatively intuitive. In particular, \( \text{Reverse} \) reverse the table, and \( \text{Permute}(A, Index) \) reorders table \( A \) according the permutation \( Index \) (the element \( A[i] \) goes to the \( Index[i] \)th position).

Question 2

a) Apply the procedure on this input:

\[
A = [5 \ 7 \ 3 \ 1 \ 4 \ 2 \ 7 \ 2 ]
\]

\[
\text{Flags} = [1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 ]
\]

b) What is the purpose of the \text{Split} procedure?

c) What is the computational time of the \text{Split} procedure?
Question 3

a) We consider the following Mystery procedure:

Algorithm 2: Mystery Procedure 2

```python
def Mystery(A, Number_Of_Bits):
    for i = 0 to Number_Of_Bits - 1 do
        bit(i) ← table containing the ith bit of the elements of A;
        A ← Split(A, bit(i));
```

(a) Run the procedure on $A = [5, 7, 3, 1, 4, 2, 7, 2]$ with $Number_Of_Bits = 3$.

(b) What is the purpose of procedure Mystery 2?

(c) Given entries of size $O(\log n)$ bits, what is the complexity with $n$ processors? With $p$ processors?

3 Connected components

We would like to design a CREW algorithm to compute the connected components of a graph $G = (V, E)$ with vertices numbered from to 1 to $n$. In particular, we are looking for an algorithm that returns a table $C$ of size $n$, such that $C(i) = C(j) = k$ if and only if $i$ and $j$ are in the connected component and $k$ is the smallest index among the vertices from this component.

Definition 1 For all iteration of the algorithm, we call the pseudo-vertex labeled by $i$ the set of vertices $j, k, l, \ldots \in V$ such that $C(j) = C(k) = C(l) = \cdots = i$. In other words, we consider the pseudo-vertex labeled by $i$ to be the same as the vertex labeled by $i$.

One of the invariants of the algorithm is that the smallest index of the vertices from the pseudo-vertex labeled by $i$ is $i$ and the vertices belonging to a pseudo-vertex are in the same connected component. This assertion is true if we initialize $C$ by: for all $i \in V = [1, n]$ : $C(i) = i$. This means that at the beginning, each processor considers itself as the pseudo-vertex of its connected component. The goal of the algorithm is to change this egocentric point of view.

Definition 2 A $k$-cyclic tree ($k \geq 0$) is a weakly connected oriented graph such that:

- Each vertex has an out-degree of 1
- There is exactly one circuit of length $k + 1$

We call a star a 0-cyclic tree.

Therefore, the previous invariant is that the oriented graph $(V, \{(i, C(i)) \mid i \in V\})$ consists of stars only. We can identify pseudo-vertex and stars, the center of the star being the index of the pseudo-vertex. Computing the connected components is done by running the following procedures several times:

Question 4

a) We consider the following graph:
Algorithm 3: Procedures to compute the connected components.

```python
def Gather():
    for $i \in S$ do in parallel
        $T(i) \leftarrow \begin{cases} 
        \min \{C(j) \mid \{i, j\} \in E, C(j) \neq C(i)\} & \text{if the set is nonempty} \\
        C(i) & \text{otherwise}
        \end{cases}$
    for $i \in S$ do in parallel
        $T(i) \leftarrow \begin{cases} 
        \min \{T(j) \mid C(j) = i, T(j) \neq i\} & \text{if the set is nonempty} \\
        C(i) & \text{otherwise}
        \end{cases}$

def Jump():
    for $i \in S$ do in parallel
        $B(i) \leftarrow T(i)$
    for $j = 1$ to $\log n$ do
        for $i \in S$ do in parallel
            $T(i) \leftarrow T(T(i))$
    for $i \in S$ do in parallel
        $C(i) \leftarrow \min \{B(T(i)), T(i)\}$
```

Apply the function Gather on this graph, then the function Jump, and the Gather function again, etc.

b) Show that after using the Gather function, connected components containing several pseudo-vertices induce 1-cyclic trees in the oriented graph $(V, \{(i, T(i)) \mid i \in V\})$. Note that the smallest pseudo-vertex of a 1-cyclic tree belongs to the cycle.

c) Show that the function Jump transforms a 1-cyclic tree into a 1-cyclic star (or pseudo-vertex).

d) Show that after $\lceil \log n \rceil$ iterations, the connected components of the graph are represented by pseudo-vertices induced by $C$.

e) What is the overall complexity of the algorithm? (account for the computation of minima)