Chapter 1 - Vertex Coloring

Problem 1.1 - Vertex Coloring
Given an undirected graph $G = (V, E)$, assign a color cu to each vertex $u \in V$ such that the following holds: $e = (v, w) \in E \Rightarrow cv \neq cw$.

Definition 1.2 - Node Identifiers
Each node has a unique identifier, e.g., its IP address. We usually assume that each identifier consists of only log n bits if the system has n nodes.

Definition 1.3 - Chromatic Number
Given an undirected Graph $G = (V, E)$, the chromatic number $\chi(G)$ is the minimum number of colors to solve Problem 1.1.

Algorithm 1 - Greedy Sequential *(see page 6)*

- **Vertex Coloring**
- Non-distributed
- Centralized
- **Theorem 1.5:** is correct and terminates in n "steps". The algorithm uses at most $\Delta + 1$ colors.

Definition 1.4 - Degree
The number of neighbors of a vertex v, denoted by $\delta(v)$, is called the degree of v. The maximum degree vertex in a graph G defines the graph degree $\Delta(G) = \Delta$.

Definition 1.6 - Synchronous Distributed Algorithm
n a synchronous al- gorithm, nodes operate in synchronous rounds. In each round, each processor executes the following steps:

1. Do some local computation (of reasonable complexity).
2. Send messages to neighbors in graph (of reasonable size).
3. Receive messages (that were sent by neighbors in step 2 of the same round).

Algorithm 3 - Reduce *(see page 7)*

- **Theorem 1.8:** is correct and has time complexity n. The algorithm uses at most $\Delta + 1$ colors.

Definition 1.7 - Time Complexity
For synchronous algorithms (as defined in 1.6) the time complexity is the number of rounds until the algorithm terminates.

Lemma 1.9
$\chi(Tree) \leq 2$

Algorithm 4 - Slow Tree Coloring *(see page 8)*

- Time Complexity: Height of tree (up to n)
- Does not need to be synchronous

Definition 1.10 - Asynchronous Distributed Algorithm
In the asynchronous model, algorithms are event driven ("upon receiving message ..., do..."). Processors cannot access a global clock. A message sent from one processor to another will arrive in finite but unbounded time.

Definition 1.11 - Time Complexity
For asynchronous algorithms (as defined in 1.6) the time complexity is the number of time units from the start of the execution to its completion in the worst case (every legal input, every execution scenario), assuming that each message has a delay of at most one time unit.

Definition 1.12 - Message Complexity
The message complexity of a syn- chronous or asynchronous algorithm is determined by the number of messages exchanged (again every legal input, every execution scenario).

Theorem 1.13 - Slow Tree Coloring
Algorithm 4 (Slow Tree Coloring) is correct. If each node knows its parent and its children, the (asynchronous) time complexity is the tree height which is bounded by the diameter of the tree; the message complexity is $n - 1$ in a tree with n nodes.

Definition 1.14 - Log-Star
$\forall x \leq 2 : \log^* x := 1 \; \forall x > 2 : \log^* x := 1 + \log^*(\log(x))$

Algorithm 5 - "6-Color" *(see page 10)*

- **Theorem 1.15:** terminates in $log^*(n)$ **time**.
- Time Complexity: $O(log^*(n))$

Algorithm 6 - Shift Down *(see page 11)*

- **Lemma 1.16:** Preserves coloring legality: also siblings are monochromatic

Algorithm 7 - Six-2-Three *(see page 11)*

- **Theorem 1.17:** colors a tree with three colors in $O(log^*(n))$.

Chapter 2 - Leader Election

Setup:

- Ring topology

Problem 2.1 - Leader Election
Each node eventually decides whether it is a leader or not, subject to the constraint that there is exactly one leader.

Definition 2.2 - Anonymous
A system is anonymous if nodes do not have unique identifiers.

Definition 2.3 - Uniform
An algorithm is called uniform if the number of nodes n is not known to the algorithm (to the nodes, if you wish). If n is known, the algorithm is called non-uniform.

Lemma 2.4
After round k of any deterministic algorithm on an anonymous ring, each node is in the same state s_k.

Theorem 2.5 - Anonymous Leader Election
Deterministic leader election in an anonymous ring is impossible.
(Also holds for other symmetric network topologies (e.g., com- plete graphs, complete bipartite graphs), but does not hold for randomized algorithms.)

Algorithm 8 - Clockwise *(see page 17)*

- **Theorem 2.6:** is correct. The **time complexity** is $O(n)$. The **message complexity** is $O(n2)$.

Algorithm 9 - Radius Growth *(see page 18)*

- **Theorem 2.7:** is correct. The **time complexity** is $O(n)$. The **message complexity** is $O(nlogn)$.
- Asynchronous
- Uniform

Definition 2.8 - Execution
Definition 2.8 (Execution). An execution of a distributed algorithm is a list of events, sorted by time. An event is a record (time, node, type, message), where type is "send" or "receive".

Definition 2.9 - Open Schedule
A schedule is an execution chosen by the scheduler. An open (undirected) edge is an edge where no message traversing the edge has been received so far. A schedule for a ring is open if there is an open edge in the ring.

Lemma 2.10
Given a ring R with two nodes, we can construct an open schedule in which at least one message is received. The nodes cannot distinguish this schedule from one on a larger ring with all other nodes being where the open edge is.

Lemma 2.11
By gluing together two rings of size $n/2$ for which we have open schedules, we can construct an open schedule on a ring of size n. If $M(n/2)$ denotes the number of messages already received in each of these schedules, at least $2M(n/2) + n/4$ messages have to be exchanged in order to solve leader election.

Lemma 2.12
Any uniform leader election algorithm for asynchronous rings has at least message complexity $M(n) \geq n(log(n) + 1)$.

Theorem 2.13 - Asynchronous Leader Election Lower Bound
Any uniform leader election algorithm for asynchronous rings has $\Omega(n * log(n))$ message complexity.

Algorithm 10 - Synchronous Leader Election *(see page 21)*

- Non-uniform (i.e. the ring size n is known)
- Every node starts at same time
- Mimimum identifies (integers) becomes the leader
- **Time Complexity:** $m * n$, where m is the minimum identifier
- **Message Complexity:** n.

Chapter 3 - Tree Algorithms

Definition 3.1 - Broadcast
A broadcast operation is initiated by a single processor, the source. The source wants to send a message to all other nodes in the system.

Definition 3.2 - Distance, Radius, Diameter
The **distance** between two nodes u and v in an undirected graph G is the number of hops of a minimum path between u and v. The **radius of a node** u is the maximum distance between u and any other node in the graph. The **radius of a graph** is the minimum radius of any node in the graph. The **diameter** of a graph is the maximum distance between two arbitrary nodes.
Relation between radius and diameter: $R \leq D \leq 2R$

Theorem 3.3 - Broadcast Lower Bound
The message complexity of broadcast is at least $n - 1$. The source's radius is a lower bound for the time complexity.

Definition 3.4 - Clean
A graph (network) is clean if the nodes do not know the topology of the graph.

Theorem 3.5 - Clean Broadcast Lower Bound
For a clean network, the num- ber of edges is a lower bound for the broadcast message complexity.

Algorithm 11 - Flooding *(see page 24)*

-

Algorithm 12 - Echo *(see page 25)*

- **Message Complexity:** $n - 1$
- Together with flooding the message complexity is $O(m)$, where $m = |E|$ is the number of edges in the graph.
- **Time Complexity:** Depth of the spanning tree.

Algorithm 13 - Dijkstra BFS *(see page 26)*

- **Spanning Tree**
- Always add the closest node.
- **Theorem 3.6:** The **time complexity** is $O(D2)$, the **message complexity** is $O(m + nD)$, where D is the diameter of the graph, n the number of nodes, and m the number of edges.

Algorithm 14 - Bellman-Ford BFS *(see page 26)*

- **Spanning Tree**
- Simply keep the distance to the root accurate.
- Border Gateway Protocol
- **Theorem 3.7:** The **time complexity** is $O(D)$, the **message complexity** is $O(nm)$, where D is the diameter of the graph, n the number of nodes, and m the number of edges.

Definition 3.8 - MST (Minimum spanning tree)
Given a weighted graph $G = (V, E, \omega)$, the MST of G is a spanning tree T minimizing $\omega(T)$, where $\omega(G') = \sum_{e \in G'} \omega_e$ for any subgraph $G' \subseteq G$.

Definition 3.9 - Blue Edge
Let T be a spanning tree of the weighted graph G and $T' \subseteq T$ a subgraph of T (also called a fragment). Edge $e = (u, v)$ is an outgoing edge of T' if $u \in T'$ and $v \notin T'$ (or vice versa). The minimum weight outgoing edge $b(T')$ is the so-called blue edge of T'.

Lemma 3.10
For a given weighted graph G (such that no two weights are the same), let T denote the MST, and T' be a fragment of T. Then the blue edge of T' is also part of T, i.e., $T' \cup b(T') \subseteq T$.

Algorithm 15 - GHS (Gallager-Humblet-Spira) *(see page 28)*

- **Theorem 3.11:** The **time complexity** is $O(nlogn)$, the **message complexity** is $O(mlogn)$.
- Directly solves leader election in general graphs: The leader is simply the last surviving root.

Chapter 4 - Distributed Sorting

Definition 4.1 - Sorting
We choose a graph with n nodes v1, ..., vn. Initially each node stores a value. After applying a sorting algorithm, node vk stores the kth smallest value.
Star Topology: Sorting takes $O(1)$.

Definition 4.2 - Node Contention
n each step of a synchronous algorithm, each node can only send and receive $O(1)$ messages containing $O(1)$ values, no matter how many neighbors the node has.
Star Topology: Sorting takes $O(n)$.

Algorithm 16 - Odd/Even Sort *(see page 32)*

- **Theorem 4.4:** sorts correctly in n steps.

Lemma 4.3
0-1 Sorting Lemma: If an oblivious comparison-exchange algorithm sorts all inputs of $0's$ and $1's$, then it sorts arbitrary inputs.

Algorithm 17 - Shearsort *(see page 33)*

- **Theorem 4.5:** sorts n values in $\sqrt(n)(logn + 1)$ time in snake-like order.

Definition 4.6 - Sorting Network
A comparator is a device with two inputs x, y and two outputs x', y' such that $x' = min(x, y)$ and $y' = max(x, y)$. We construct so-called comparison networks that consist of wires that connect comparators (the output port of a comparator is sent to an input port of another comparator). Some wires are not connected to comparator outputs, and some are not connected to comparator inputs. The first are called input wires of the comparison network, the second output wires. Given n values on the input wires, a sorting network ensures that the values are sorted on the output wires. We will also use the term width to indicate the number of wires in the sorting network.

Definition 4.7 - Depth
he depth of an input wire is 0. The depth of a comparator is the maximum depth of its input wires plus one. The depth of an output wire of a comparator is the depth of the comparator. The depth of a comparison network is the maximum depth (of an output wire).

Definition 4.8 - Bitonic Sequence
A bitonic sequence is a sequence of numbers that first monotonically increases, and then monotonically decreases, or vice versa.

Algorithm 18 - Half Cleaner *(see page 35)*

- **Lemma 4.9:** Feeding a bitonic sequence into a half cleaner, the half cleaner cleans (makes all-0 or all-1) either the upper or the lower half of the n wires. The other half is bitonic.

Algorithm 19 - Bitonic Sequnence Sorter *(see page 35)*

- **Lemma 4.10:** A bitonic sequence sorter of width n sorts bitonic sequences. It has depth $logn$.

Algorithm 20 - Merging Network *(see page 36)*

- **Lemma 4.11:** A merging network of width n merges two sorted input sequences of length $n/2$ each into one sorted sequence of length n.

Algorithm 21 - Batcher's "Bitonic" Sorting Network *(see page 36)*

- **Theorem 4.12:** A sorting network sorts an arbitrary sequence of n values. It has depth $O(log^2 n)$.
- Can be simulated by a Butterfly network and other hypercubic networks.

Definition 4.13 - Distributed Counting
A distributed counter is a variable that is common to all processors in a system and that supports an atomic test-and-increment operation. The operation delivers the system's counter value to the requesting processor and increments it.

Definition 4.14 - Balancer
A balancer is an asynchronous flip-flop which forwards messages that arrive on the left side to the wires on the right, the first to the upper, the second to the lower, the third to the upper, and so on.

Algorithm 22 - Bitonic Counting Network *(see page 38)*

- **Theorem 4.20:** In a quiescent state, the w output wires of a bitonic counting network of width w have the step property.
- **Lemma 4.23:** The bitonic counting network is not linearizable.

Definition 4.15 - Step Property
A sequence y_0,y_1,...,y_{w-1} is said to have the step property,if $0 \leq y_i - y_j \leq 1$, for any $i < j$.

Fact 4.16
For a balancer, we denote the number of consumed messages on the i^{th} input wire with x_i, $i = 0, 1$. Similarly, we denote the number of sent messages on the ith output wire with y_i , $i = 0, 1$. A balancer has these properties:

1. A balancer does not generate output-messages; that is, $x_0 + x_1 \geq y_0 + y_1$ in any state.
2. Every incoming message is eventually forwarded. In other words, if we are in a quiescent state (no message in transit), then $x_0 + x_1 = y_0 + y_1$.
3. The number of messages sent to the upper output wire is at most one higher than the number of messages sent to the lower output wire: in any state $y_0 = \lceil (y_0 + y_1)/2 \rceil$ (thus $y_1 = \lfloor (y_0 + y_1)/2 \rfloor$).

Fact 4.17
If a sequence $y_0, y_1, \ldots, y_{w-1}$ has the step property,

1. then all its subsequences have the step property.
2. then its even and odd subsequences satisfy
   $\sum_{i=0}^{w/2-1} y_{2i} = \lceil \frac{1}{2} \sum_{i=0}^{w-1} y_i \rceil$ and $\sum_{i=0}^{w/2-1} y_{2i+1} = \lfloor \frac{1}{2} \sum_{i=0}^{w-1} y_i \rfloor$

Fact 4.18
If two sequences $x_0, x_1,\ldots,x_{w-1}$ and $y_0, y_1, \ldots, y_{w-1}$ have the step property,

1. and $\sum_{i=0}^{w-1} x_i = \sum_{i=0}^{w-1} y_i$, then $x_i = y_i$ for $i = 0,\ldots,w-1$.

2. and $\sum_{i=0}^{w-1} x_i = \sum_{i=0}^{w-1} y_i + 1$ then there exists a unique j (j = 0,1,...,$w-1$) such that $x_j = y_j + 1$, and $x_i = y_i$ for i = 0,...,$w-1$, $i \neq j$.

Lemma 4.19
Let M[w] be a Merger of width w. In a quiescent state (no mes- sage in transit), if the inputs x_0, x_1, ..., $x_{w/2-1}$ resp. $x_{w/2}$, $x_{w/2+1}$, ..., x_{w-1} have the step property, then the output y_0, y_1, ..., y_{w-1} has the step property.

Theorem 4.21 - Counting vs. Sorting
If a network is a counting network then it is also a sorting network, but not vice versa.

Definition 4.22 - Linearizable
A system is linearizable if the order of the values assigned reflects the real-time order in which they were requested. More formally, if there is a pair of operations o_1, o_2, where operation o_1 terminates be- fore operation o_2 starts, and the logical order is "o_2 before o_1", then a distributed system is not linearizable.

Chapter 5 - Maximal Independent Set

Setup:

- (First) randomized algorithm.

Definition 5.1 - Independent Set
Given an undirected Graph $G = (V, E)$ an independent set is a subset of nodes $U \subseteq V$, such that no two nodes in U are adjacent. An **independent set is maximal** if no node can be added without violating independence. An independent set of maximum cardinality is called maximum.

Algorithm 23 - Slow MIS *(see page 46)*

- **Theorem 5.2:** features a **time complexity** of $O(n)$ and a **message complexity** of $O(m)$.

Corollary 5.3
Given a coloring algorithm that needs C colors and runs in time T, we can construct a MIS in time C + T. *(We first choose all nodes of the first color. Then, for each additional color we add "in parallel" (without conflict) as many nodes as possible.)*

Algorithm 24 - Fast MIS *(see page 47)*

- **Lemma 5.4 - Joining MIS:** A node v joins the MIS in Step 2 of the Fast MIS with probability $p \geq \frac{1}{4d(v)}$.
- **Lemma 5.5 - Good Nodes:** A node v is called good if
  $\sum_{w \in N(v)} \frac{1}{2d(w) \geq \frac{1}{6}}$
  Otherwise we call v a bad node. A good node will be removed in Step 3 with probability $p \geq \frac{1}{36}$.
- **Lemma 5.6 - Good Edges:** An edge $e = (u, v)$ is called bad if both u and v are bad; else the edge is called good. The following holds: At any time at least half of the edges are good.
- **Theorem 5.8:** terminates in expected **time** $O(logn)$ (, even with high probability).

Lemma 5.7
A bad node has outdegree (number of edges pointing away from bad node) at least twice its indegree (number of edges pointing towards bad node).

Algorithm 25 - Fast MIS 2 *(see page 50)*

- **Lemma 5.10:** In a single phase, we remove at least half of the edges in expectation.
- **Theorem 5.11:** terminates after **time** at most $3log_{4/3}m + 1 \in O(logn)$ phases in expectation.
  Corollary 5.14: terminates w.h.p. in O(log n) time.

Theorem 5.9 - Linearity of Expectation
Let X_i, $i = 1, \ldots, k$ denote random variables, then
$E[\sum_i X_i] = \sum_i E[X_i]$.

Definition 5.12 - With High Probability (W.h.p.)
We say that an algorithm terminates w.h.p. (with high probability) within $O(t)$ time if it does so with probability at least $1 - 1/n^c$ for any choice of $c \geq 1$. Here c may affect the constants in the Big-O notation because it is considered a "tunable constant" and usually kept small.

Definition 5.13 - Chernoff's Bound
Let $X = \sum_{i=1}^{k} X_i$ be the sum of k independent 0-1 random variables. Then Chernoff's bound states that w.h.p.
$|X - E[X]| \in O(logn + \sqrt{E[X]logn})$

Definition 5.15 - Matching
Given a graph $G = (V, E)$ a matching is a subset of edges $M \subseteq E$, such that no two edges in M are adjacent (i.e., where no node is adjacent to two edges in the matching). A matching is maximal if no edge can be added without violating the above constraint. A matching of maximum cardinality is called maximum. A matching is called perfect if each node is adjacent to an edge in the matching.

Definition 5.16 - Approximation
An approximation algorithm A for a maximization problem Π has an approximation factor of r if the following condition holds for all instances $I \in \Pi$:
$\frac{OPT(I)}{A(I)} \leq r$.

Algorithm 26 - General Graph Coloring *(see page 54)*

- **Theorem 5.17:** $(\Delta + 1)$-colors an arbitrary graph in $O(logn)$ time, with high probability, Δ being the largest degree in the graph.

Definition 5.18 - Bounded Independence
$G = (V, E)$ is of bounded independence, if each neighborhood contains at most a constant number of independent (i.e., mutually non-adjacent) nodes.

Definition 5.19 - (Minimum) Dominating Set
A dominating set is a subset of the nodes such that each node is in the set or adjacent to a node in the set. A minimum dominating set is a dominating set containing the least possible number of nodes.

Corollary 5.20
On graphs of bounded independence, a constant-factor approximation to a minimum dominating set can be found in time $O(logn)$ w.h.p.

Chapter 6 - Locality Lower Bounds

Algorithm 27 - Synchronous Algorithm: Canonical Form *(see page 60)*

- Propagate all initial states in r-neighborhood
- Compute output based on all received states
- **Lemma 6.1**: Every deterministic, synchronous r-round algorithm can be transformed into an algorithm of this form if message size and computation is not bounded.

Definition 6.2 - r-hop view
We call the collection of the initial states of all nodes in the r-neighborhood of a node v, the r-hop view of v.

Definition 6.4 - Neighborhood Graph
For a given family of network graphs $\mathcal{G}$, the r-neighborhood graph $\mathcal{N}_r(\mathcal{G})$ is defined as follows. The node set of $\mathcal{N}_r(\mathcal{G})$ is the set of all possible labeled r-neighborhoods (i.e., all possible r-hop views). There is an edge between two labeled r-neighborhoods $\mathcal{V}_r$ and $\mathcal{V}'_r$ if $\mathcal{V}_r$ and $\mathcal{V}'_r$ can be the r-hop views of two adjacent nodes.

Definition 6.4 - Deline Graph
The directed line graph (diline graph) $\mathcal{DL}(G)$ of a directed graph $G = (V, E)$ is defined as follows. The node set of $\mathcal{DL}(G)$ is $V[\mathcal{DL}(G)] = E$. There is a directed edge $\big((w, x); (y, z)\big)$ between $(w, x) \in E$ and $(y, z) \in E$ iff $x = y$, i.e., if the first edge ends where the second one starts.

Theorem 6.11 - Directed Ring Coloring
Every deterministic, distributed algorithm to color a directed ring with 3 or less colors needs at least $(\log^* n)/2 - 1$ rounds.

Chapter 7 - All-to-All Communication

Definition 7.1 - Minimum Spanning Tree (MST)
Given a weighted graph $G = (V, E, \omega)$. The MST of G is a spanning tree T minimizing $\omega(T)$, where $\omega(H) = \sum_{e \in H} \omega_e$ for any subgraph $H \subseteq G$.

Algorithm 28 - Simple MST Construction (at node v) *(see page 69)*

- sequentially grow fragments by adding blue edges to MST
- **Theorem 7.3:** On a complete graph, Algorithm 28 computes an MST in time $\mathcal{O}(\log n)$.
- essentially equivalent to the GHS algorithm 15 in chapter 3.

Algorithm 29 - Fast MST construction (at node v) *(see page 70)*

- add multiple blue edges per round to the spanning tree
- **Theorem 7.5:** Algorithm 29 computes an MST in time $\mathcal{O}(\log \log n)$.

Algorithm 30 - AddEdges(E') *(see page 71)*

- Given the set of edges E', determine which edges are added to the MST in Algorithm 29
- Ensures only safe edges are added.

Consult Table 7.1 on page 72 for known time complexity upper and lower bounds for MST construction.

Chapter 8 - Social Networks

Definition 8.1 - Cluster Coefficient
The cluster coefficient of a network is defined by the probability that two friends of a node are likely to be friends as well, averaged over all the nodes.

Definition 8.2 - Augmented Grid
We take $n = m^2$ nodes $(i, j) \in V = 1, \ldots, m^2$ that are identified with the lattice points on an $m \times m$ grid. We define the distance between two nodes (i, j) and (k, ℓ) as $d\big((i, j), (k, \ell)\big) = |k - i| + |\ell - j|$ as the distance between them on the $m \times m$ lattice. The network is modeled using a parameter $\alpha \geq 0$. Each node u has a directed edge to every lattice neighbor. These are the local contacts of a node. In addition, each node also has an additional random link (the long-range contact). For all u and v, the long-range contact of u points to node v with probability proportional to $d(u, v)^{-\alpha}$, i.e., with probability $d(u, v)^{-\alpha} / \sum_{w \in V \setminus \{u\}} d(u, w)^{-\alpha}$. Figure 8.2 (page 78) illustrates the model.

Definition 8.3 - With High Probability
Some probabilistic event is said to occur with high probability (w.h.p.), if it happens with a probability $p \leq 1 - 1/n^c$, where c is a constant. The constant c may be chosen arbitrarily, but it is considered constant with respect to Big-O notation.

Theorem 8.4 - Diameter of Augmented Grid
The diameter of the augmented grid with $\alpha = 0$ is $\mathcal{O}(\log n)$ with high probability.

Algorithm 31 - Greedy Routing *(see page 79)*

- While not at destination go the a neighbor which is closest to the destination
- **Lemma 8.5:** In the augmented grid, Algorithm 31 finds a routing path of length at most $2(m - 1) \in \mathcal{O}(\sqrt{n})$

Definition 8.7 - Phase
Consider routing from source s to target t and assume that we are at some intermediate node w. We say that we are in phase j at node w if the lattice distance $d(w, t)$ to the target node t is between $2^j < d(w, t) \leq 2^{j+1}$.

Lemma 8.8
Assume that we are in phase j at node w when routing from s to t. The probability for getting (at least) to phase $j - 1$ in one step is at least $\Omega(1/\log n)$.

Theorem 8.9 - Expected Lenght of Greedy Routing
Consider the greedy routing path from a node s to a node t on an augmented grid with parameter $\alpha = 2$. The expected length of the path is $\mathcal{O}(\log^2 n)$.

Theorem 8.10 - Two Player Rumor Game
In a two player rumor game where both players select one node to initiate their rumor in the graph, the first player does not always win.

Chapter 9 - Shared Memory

Definition 9.1 - Shared Memory
A shared memory system is a system that consists of asynchronous processes that access a common (shared) memory. A process can atomically access a register in the shared memory through a set of predefined operations. An atomic modification appears to the rest of the system instantaneously. Apart from this shared memory, processes can also have some local (private) memory.

Definition 9.2 - Mutual Exclusion
We are given a number of processes, each executing the following code sections:
$< Entry > \rightarrow < CriticalSection > \rightarrow < Exit > \rightarrow < RemainingCode >$
A mutual exclusion algorithm consists of code for entry and exit sections, such that the following holds

- **Mutual Exclusion:** At all times at most one process is in the critical section.
- **No deadlock:** If some process manages to get to the entry section, later some (possibly different) process will get to the critical section.

Sometimes we in addition ask for

- **No lockout:** If some process manages to get to the entry section, later the same process will get to the critical section.
- **Unobstructed exit:** No process can get stuck in the exit section.

Algorithm 32 - Mutual Exclusion: Test-and-Set *(see page 87)*

- Solution of the mutual exclusion problem using test-and-set.
- **Theorem 9.3** contains the proof of correctness.

Algorithm 44 - Mutual Exclusion: Peterson's Algorithm *(see page 88)*

- Solution of the mutual exclusion problem without using any Read-Modify-Write Operations but using a spinlock (busy-wait).
- **Theorem 9.4** contains the proof of correctness.
- Basic version only works for 2 processes but can be extended to n by using a tournament tree.

Definition 9.5 - Collect
There are two operations: A STORE(val) by process p_i sets val to be the latest value of its register R_i. A COLLECT operation returns a view, a partial function V from the set of processes to a set of values, where $V(p_i)$ is the latest value stored by p_i, for each process p_i. For a COLLECT operation *cop*, the following validity properties must hold for every process p_i:

- If $V(p_i) = \perp$, then no STORE operation by p_i precedes *cop*.
- If $V(p_i) = v \neq \perp$, then v is the value of a STORE operation *sop* of p_i that does not follow *cop*, and there is no STORE operation by p_i that follows *sop* and precedes *cop*

Algorithm 34 - Collect: Simple (Non-Adaptive) Solution *(see page 89)*

- directly do STORE and read every register on COLLECT
- STORE has step complexity 1 and the step complexity of a COLLECT operation is n.
- Not optimal in COLLECT because it always reads all register even if they were not written.

Definition 9.6 - Splitter
A splitter is a synchronization primitive with the following characteristic. A process entering a splitter exits with either STOP, LEFT, or RIGHT. If k processes enter a splitter, at most one process exits with STOP and at most $k-1$ processes exit with LEFT and RIGHT, respectively.

Algorithm 35 - Splitter Code *(see page 90)*

- Implementation of Definition 9.6 with correctness proof in **Lemma 9.7**.

Algorithm 36 - Adaptive Collect: Binary Tree Algorithm *(see page 92)*

- Implementation of STORE and COLLECT using a splitter and optimizing the COLLECT operation.
- **Lemma 9.8** proofs correctness and shows that the step complexity of the first STORE for a process p_i is $\mathcal{O}(k)$, the step complexity of every additional STORE of p_i is $\mathcal{O}(1)$, and the step complexity of COLLECT is $\mathcal{O}(k)$.
- The space complexity of this algorithm is exponential in n because we have to allocate memory for the complete binary tree of depth $n-1$.
- It is however possible to make the algorithm more space efficient by using a randomized splitter leading to a binary tree of depth only $\mathcal{O}(\log n)$ with high probability.

Theorem 9.9 - Complexity with Splitter Matrix
With the in chapter 9.3.4 introduced algorithm using a **Splitter Matrix** instead of a binary tree to optimize space complexity, we get:
Let k be the number of participating processes. The step complexity of the first store of a process p_i is $\mathcal{O}(k)$, the step complexity of every additional store of p_i is $\mathcal{O}(1)$, and the step complexity of collect is $\mathcal{O}(k^2)$.

Chapter 10 - Shared Objects

Chapter 11 - Wireless Protocols

Chapter 12 - Synchronization

Chapter 13 - Peer-to-Peer Computing