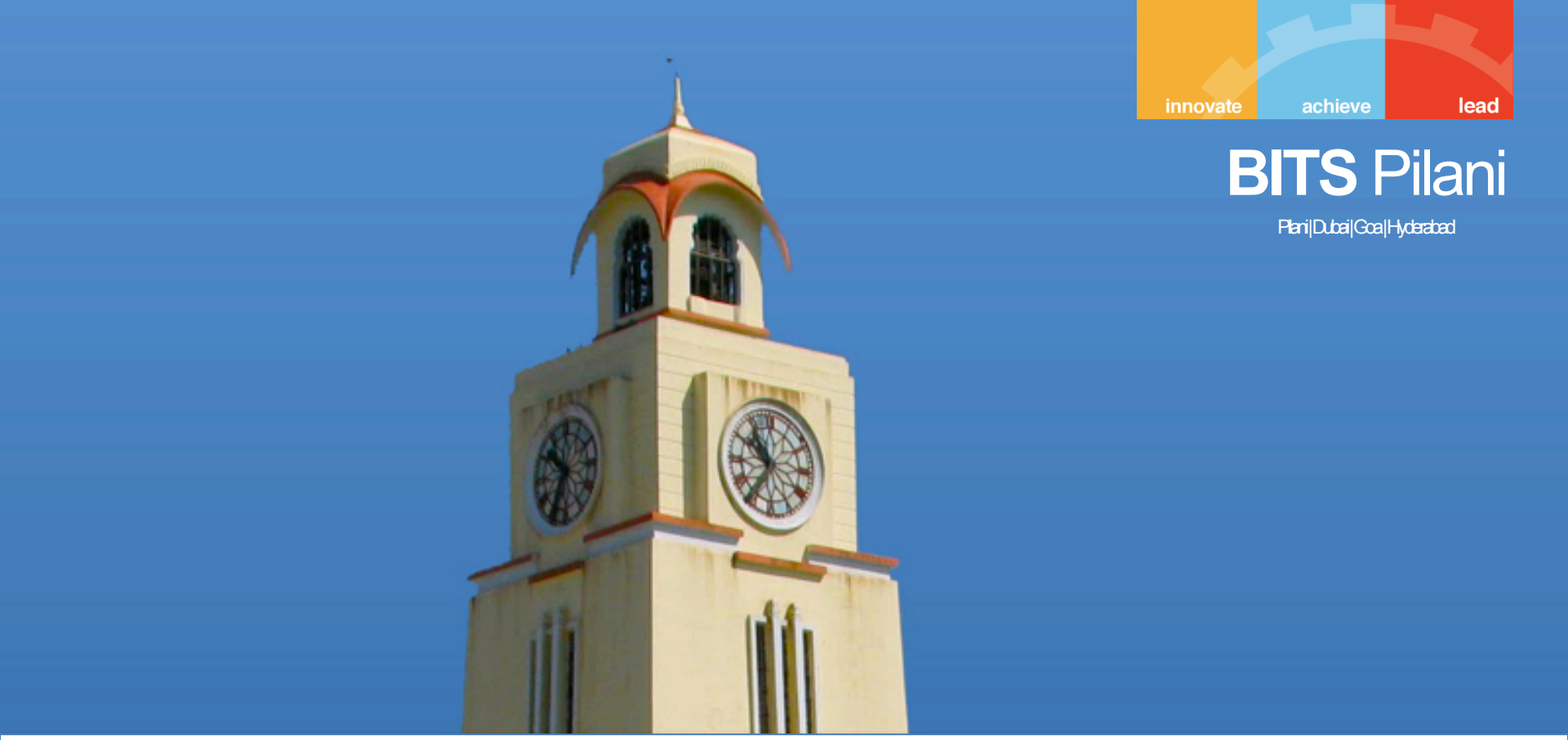
Data Structures and Algorithms Design

DSECLZG519

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Amisha Gupta



innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Webinar #2

Binary Tree Traversals & Selection Sort

Agenda for Webinar

- **Binary Tree Traversals**
- **Selection Sort**

Recap: Tree

A tree is a frequently-used data structure to simulate a hierarchical tree structure.

Each node of the tree will have a root value and a list of references to other nodes which are called child nodes.

A Binary Tree is one of the most typical tree structure. As the name suggests, a binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.

Tree Terminology

Nodes: the elements in the tree

Root: origin of the tree

Edges: connections between nodes

Empty tree: has no nodes and no edges

Parent: the node which has child/children

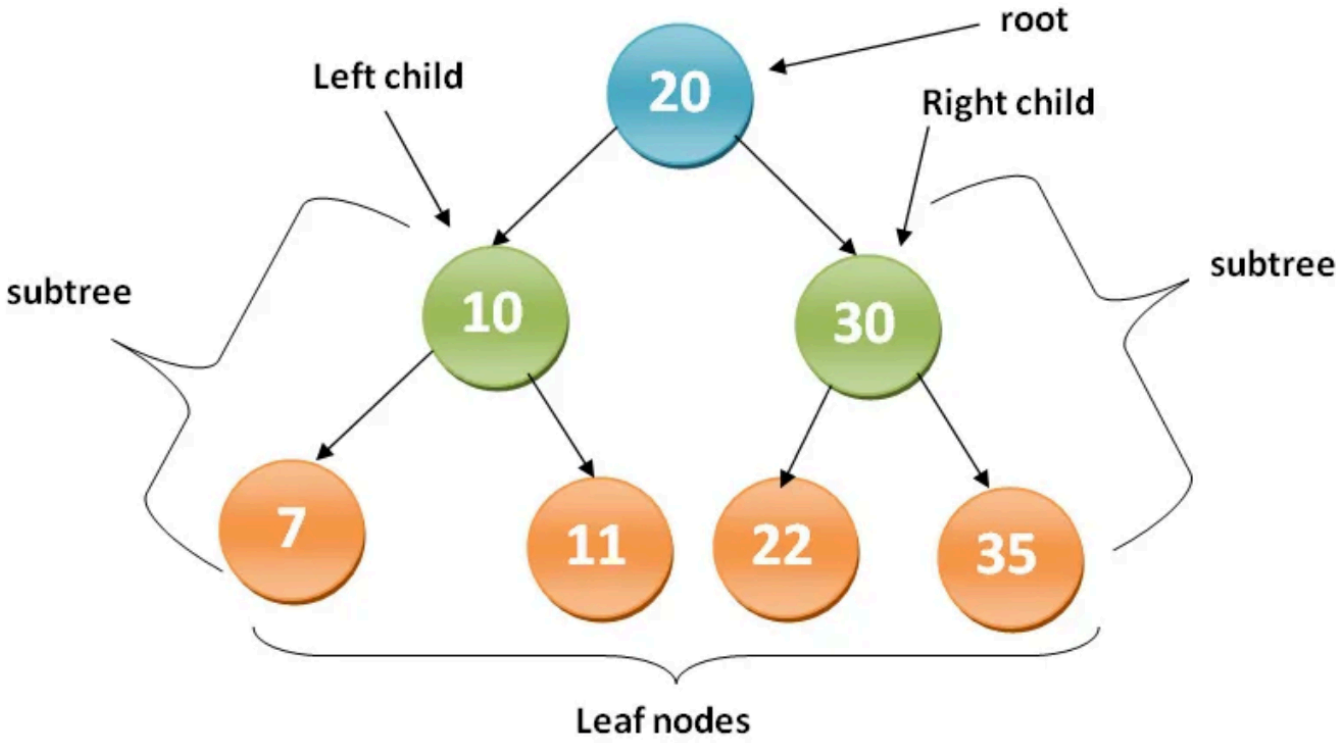
Child: descendant of any node

Siblings: the nodes which share same Parent

Leaf: the nodes which does not have children

Subtree of a node: consists of a child node and all its descendants

Binary Tree Example:



Tree Traversals

A traversal of a tree requires that each node of the tree be **visited** once.

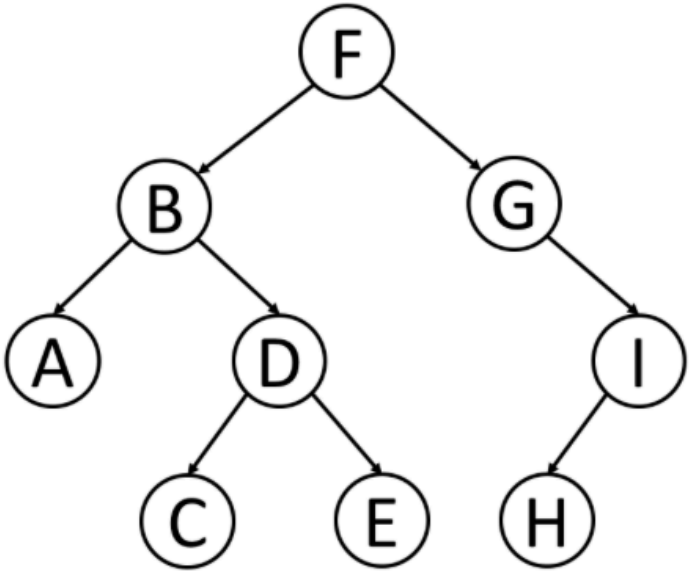
1. Pre-order Traversal - visit the root first, then traverse the left subtree, finally, traverse the right subtree.

2. In-order Traversal - traverse the left subtree first, then visit the root, finally, traverse the right subtree.

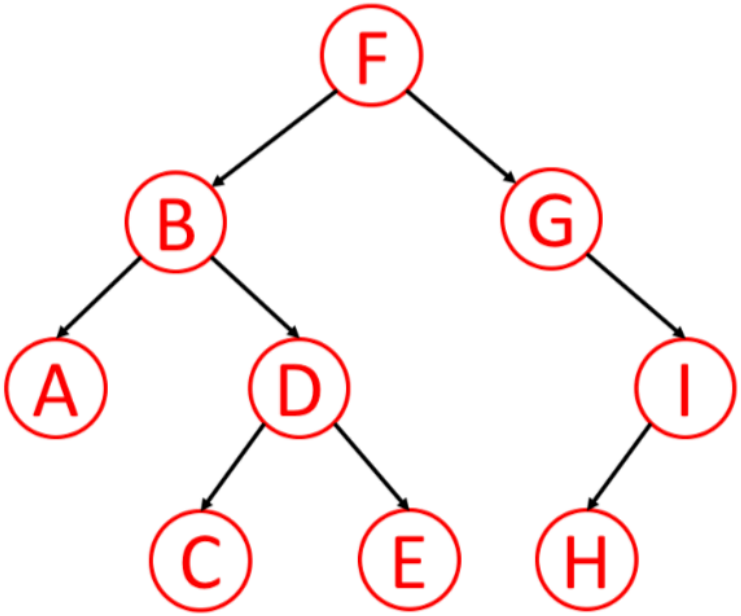
3. Post-order Traversal - traverse the left subtree first, then traverse the right subtree, finally, visit the root.

Pre-order Traversal

Given a binary tree, find its preorder traversal.



Pre-order Traversal

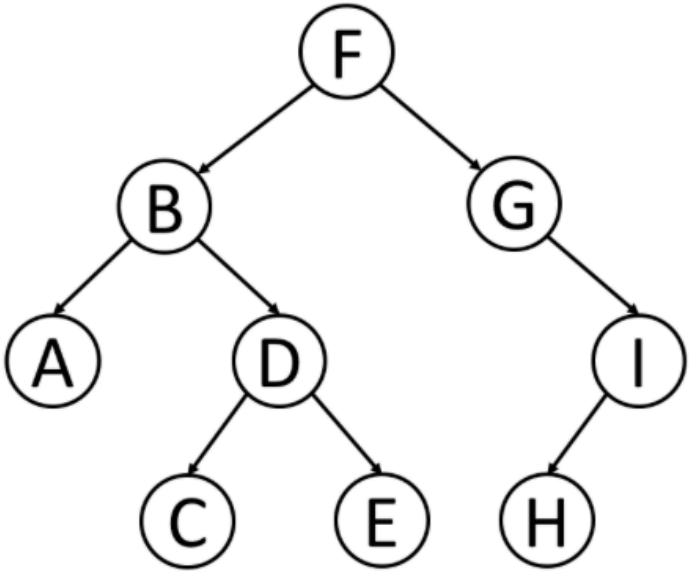


Preorder:

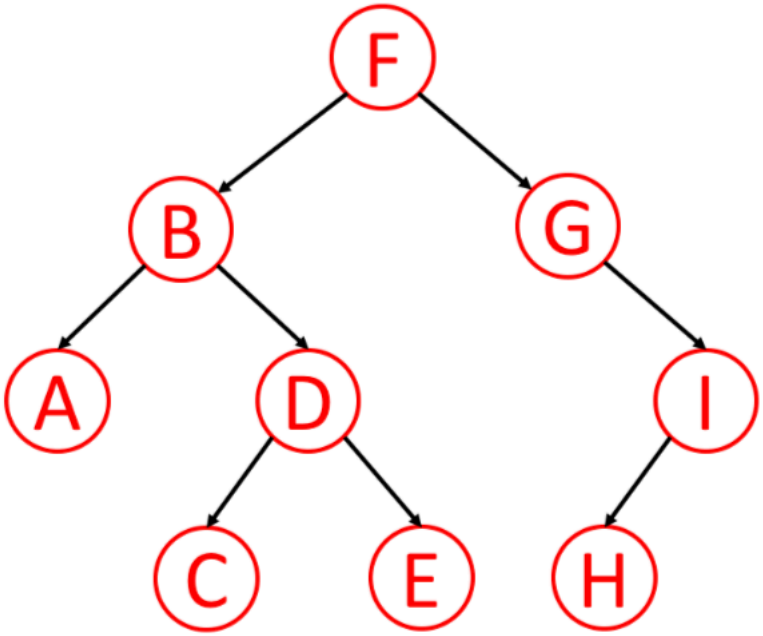
F	B	A	D	C	E	G	I	H

In-order Traversal

Given a binary tree, find its inorder traversal.



In-order Traversal

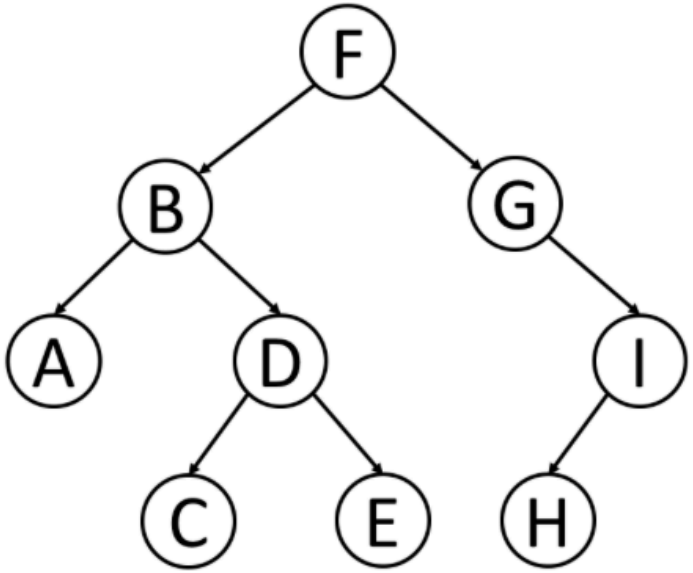


Inorder:

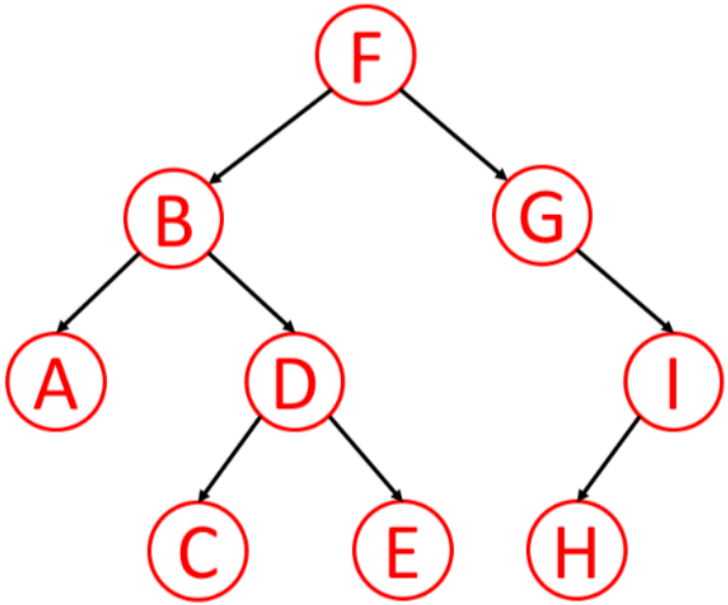
A	B	C	D	E	F	G	H	I

Post-order Traversal

Given a binary tree, find its postorder traversal.



Post-order Traversal



Postorder:

A	C	E	D	B	H	I	G	F

Construct Binary Tree from In-order and Pre-order Traversal

Example:
Pre-order traversal: [A, B, D, E, C, F]
In-order traversal: [D, B, E, A, F, C]

Step 1: Identify the Root Node
The first element of the pre-order traversal is 'A'. Therefore, 'A' is the root of the binary tree.

Step 2: Find the Root in the In-order Traversal
Locate the root node 'A' in the in-order traversal. Elements to the left of 'A' (D, B, E) belong to the left subtree, and elements to the right of 'A' (F, C) belong to the right subtree.

Step 3: Divide the In-order Traversal
Divide the in-order traversal into two parts based on the root node 'A':
Left subtree (in-order): [D, B, E]
Right subtree (in-order): [F, C]

Step 4: Divide the Pre-order Traversal
Divide the pre-order traversal into two parts based on the number of nodes in the left and right subtrees:
Left subtree (pre-order): [B, D, E]
Right subtree (pre-order): [C, F]

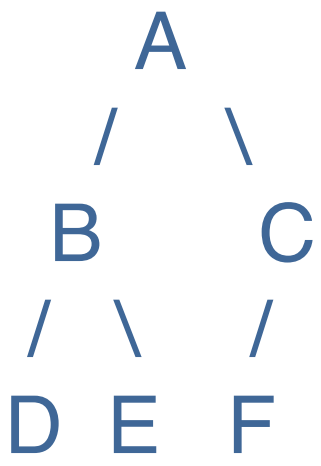
Construct Binary Tree from In-order and Pre-order Traversal

Step 5: Recurse for the Left Subtree
Using the divided pre-order and in-order traversals of the left subtree, repeat steps 1-4 to construct the left subtree recursively.
Pre-order traversal (left subtree): [B, D, E]
In-order traversal (left subtree): [D, B, E]

Step 6: Recurse for the Right Subtree
Using the divided pre-order and in-order traversals of the right subtree, repeat steps 1-4 to construct the right subtree recursively.
Pre-order traversal (right subtree): [C, F]
In-order traversal (right subtree): [F, C]

Step 7: Connect the Subtrees

Construct Binary Tree from In-order and Pre-order Traversal



Problem 1:

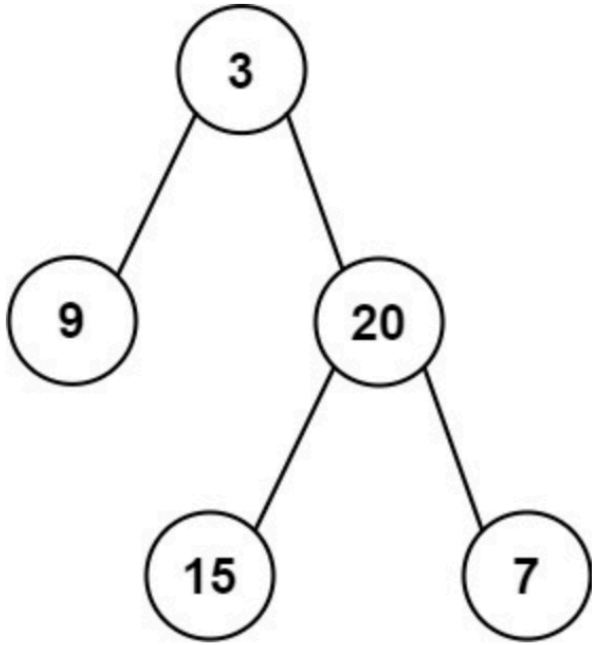
Given two integer arrays preorder and inorder where preorder is the pre-order traversal of a binary tree and inorder is the in-order traversal of the same tree, construct and return the binary tree.

preorder = [3, 9, 20, 15, 7]

inorder = [9, 3, 15, 20, 7]

Solution:

```
[3, 9, 20, null, null, 15, 7]
```



Construct Binary Tree from In-order and Post-order Traversal

Example:
Post-order traversal: [D, E, B, F, C, A]
In-order traversal: [D, B, E, A, F, C]

Step 1: Identify the Root Node
The last element of the post-order traversal is 'A'. Therefore, 'A' is the root of the binary tree.

Step 2: Find the Root in the In-order Traversal
Locate the root node 'A' in the in-order traversal. Elements to the left of 'A' (D, B, E) belong to the left subtree, and elements to the right of 'A' (F, C) belong to the right subtree.

Step 3: Divide the In-order Traversal
Divide the in-order traversal into two parts based on the root node 'A':
Left subtree (in-order): [D, B, E]
Right subtree (in-order): [F, C]

Step 4: Divide the Post-order Traversal
Divide the post-order traversal into two parts based on the number of nodes in the left and right subtrees:
Left subtree (post-order): [D, E, B]
Right subtree (post-order): [F, C]

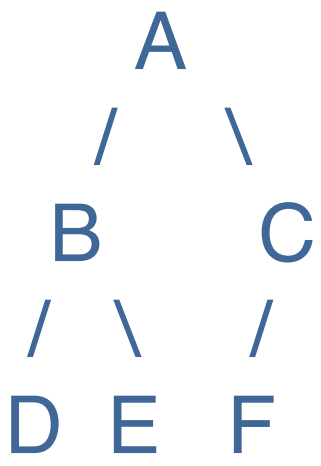
Construct Binary Tree from In-order and Post-order Traversal

Step 5: Recurse for the Left Subtree
Using the divided post-order and in-order traversals of the left subtree, repeat steps 1-4 to construct the left subtree recursively.
Post-order traversal (left subtree): [D, E, B]
In-order traversal (left subtree): [D, B, E]

Step 6: Recurse for the Right Subtree
Using the divided post-order and in-order traversals of the right subtree, repeat steps 1-4 to construct the right subtree recursively.
Post-order traversal (right subtree): [F, C]
In-order traversal (right subtree): [F, C]

Step 7: Connect the Subtrees

Construct Binary Tree from In-order and Pre-order Traversal



Problem 2:

Given two integer arrays postorder and inorder where postorder is the post-order traversal of a binary tree and inorder is the in-order traversal of the same tree, construct and return the binary tree.

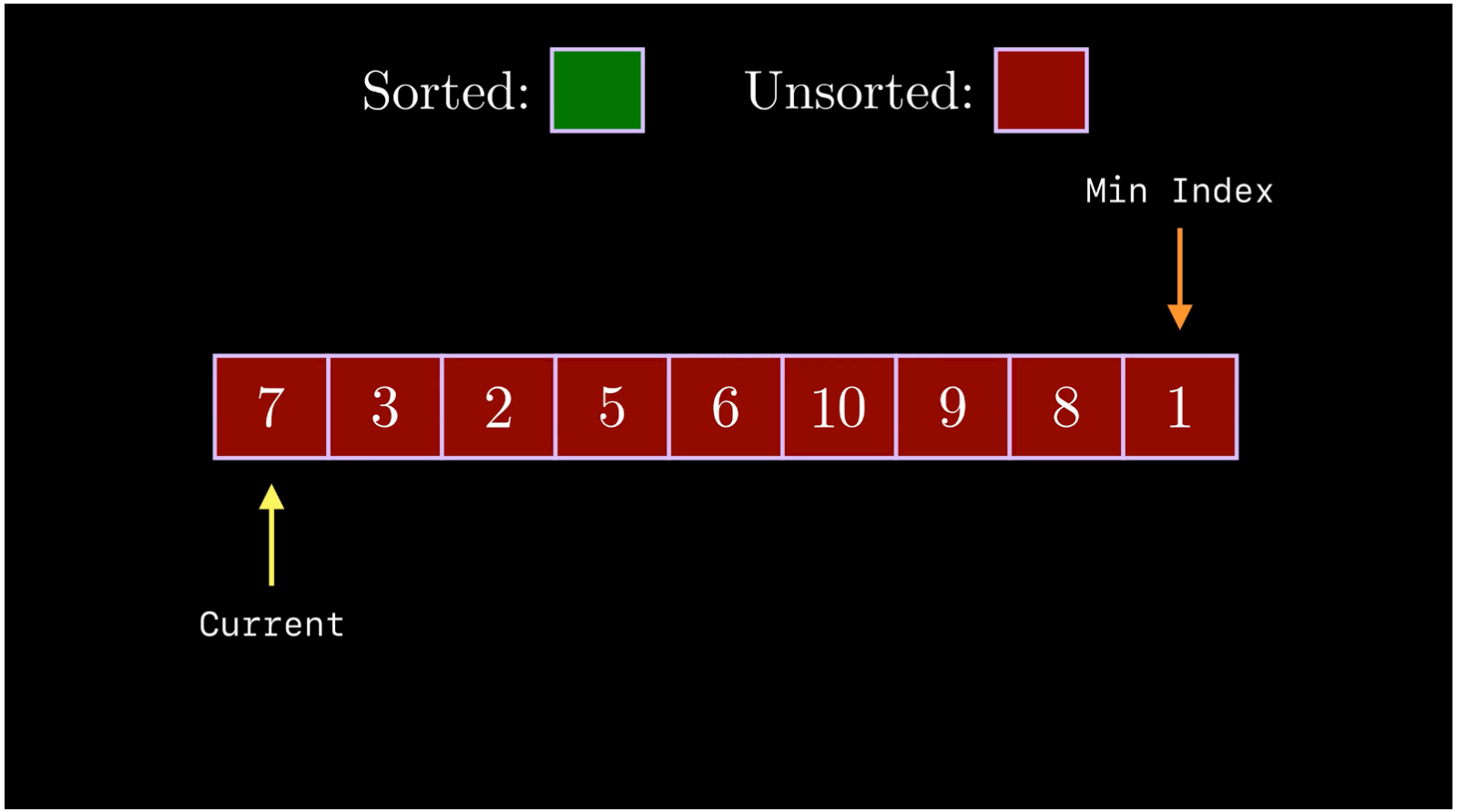
postorder = [8, 4, 5, 2, 6, 7, 3, 1]

inorder = [4, 8, 2, 5, 1, 6, 3, 7]

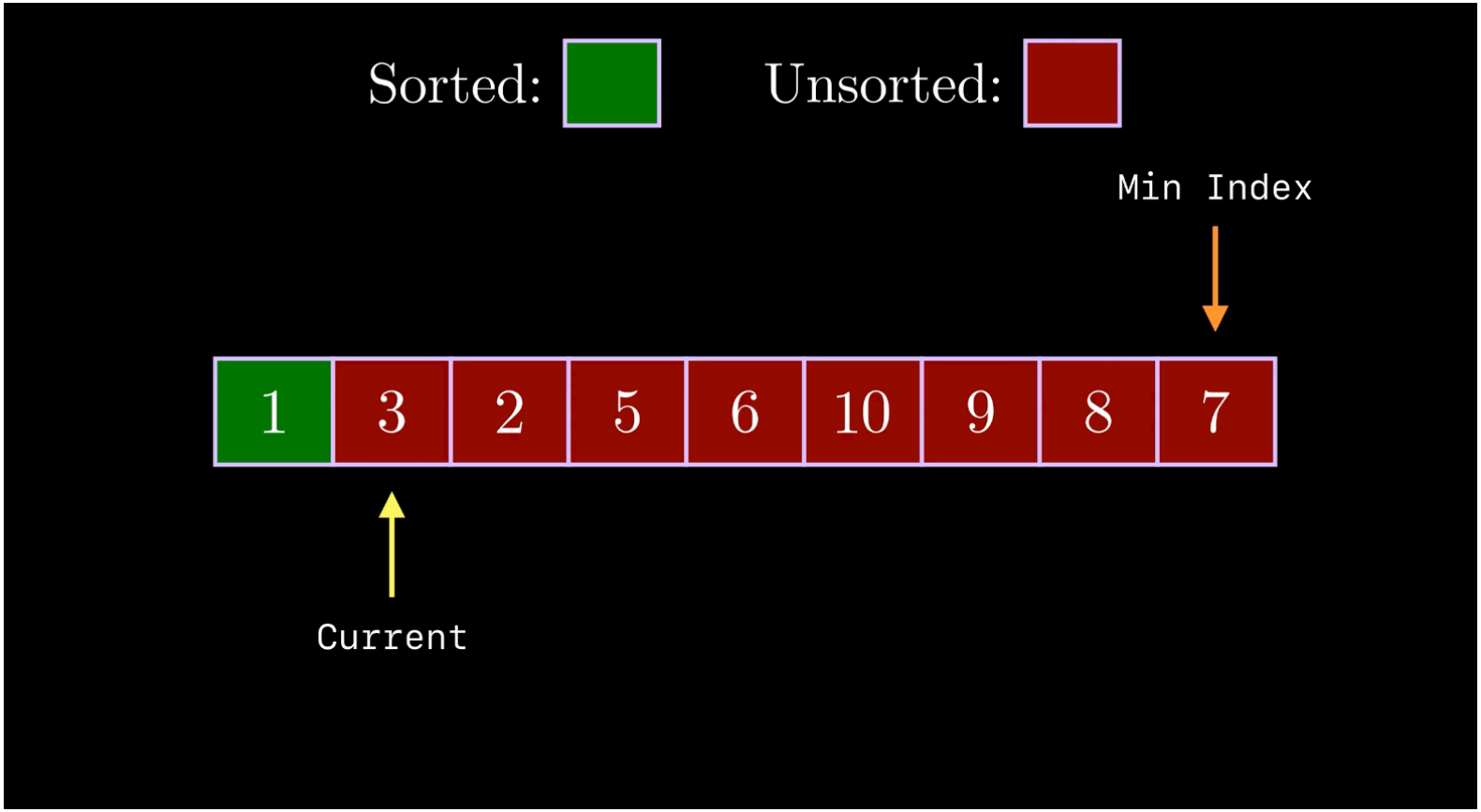
Selection Sort

- Selection sort is a simple comparison-based sorting algorithm.
- It works by dividing the input into two portions: a sorted portion and an unsorted portion.
- The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion and swaps it with the first element of the unsorted portion, thereby expanding the sorted portion.
- This process continues until the entire input is sorted.
- Selection sort has a time complexity of O(n^2) and is generally less efficient than more advanced sorting algorithms for large datasets.
- However, it is easy to understand and implement, making it useful for small or nearly sorted lists.

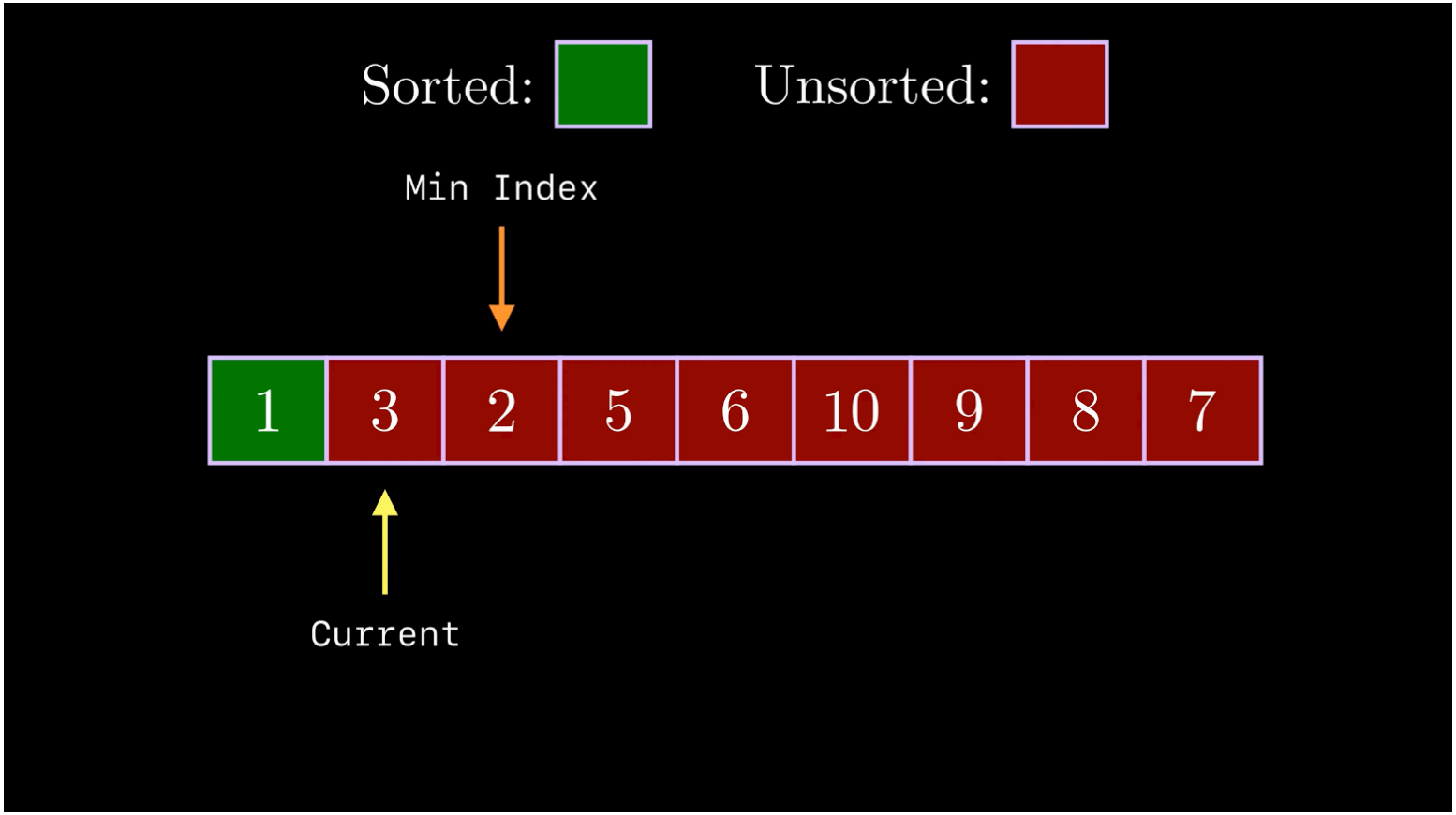
Selection Sort



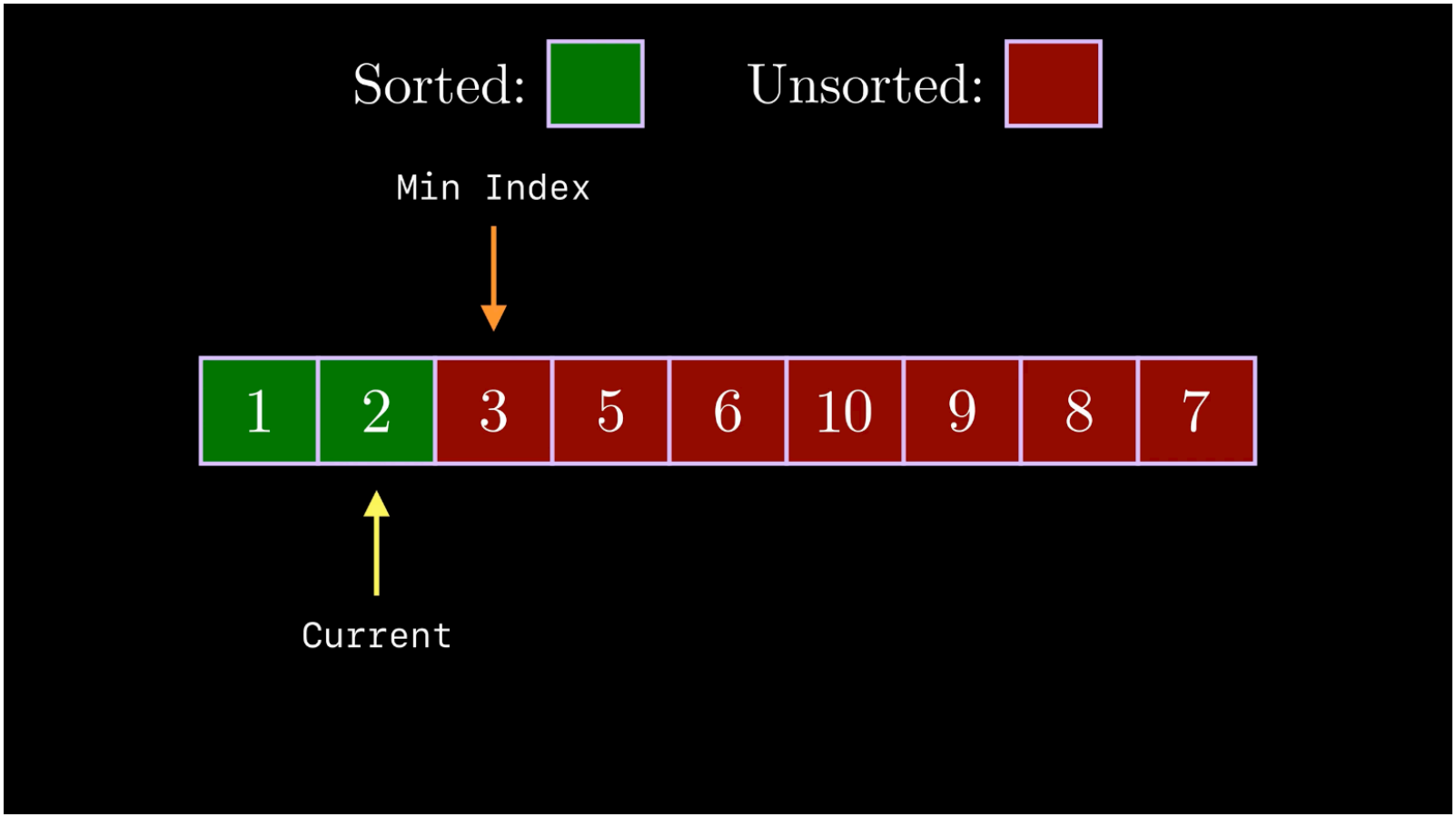
Selection Sort



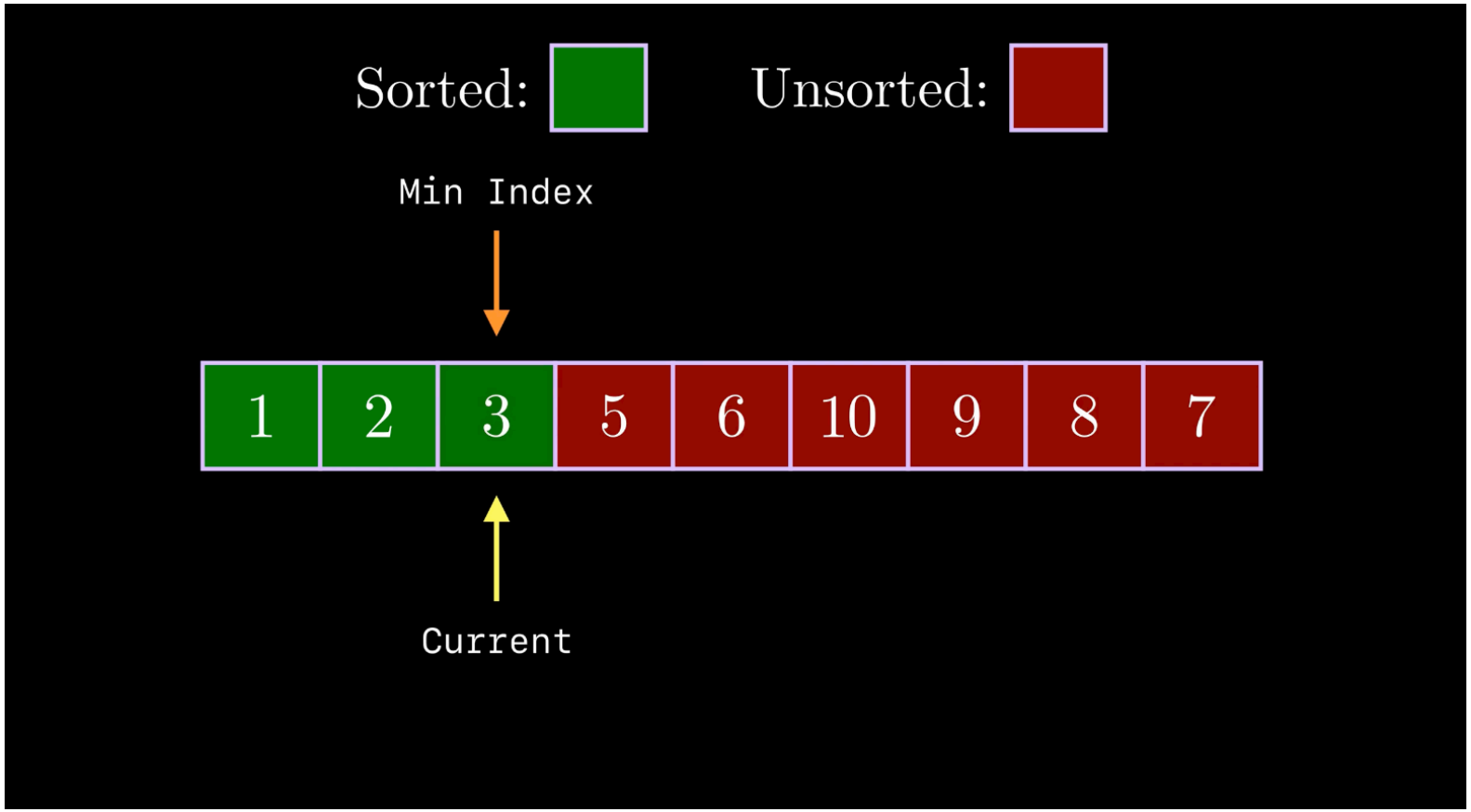
Selection Sort



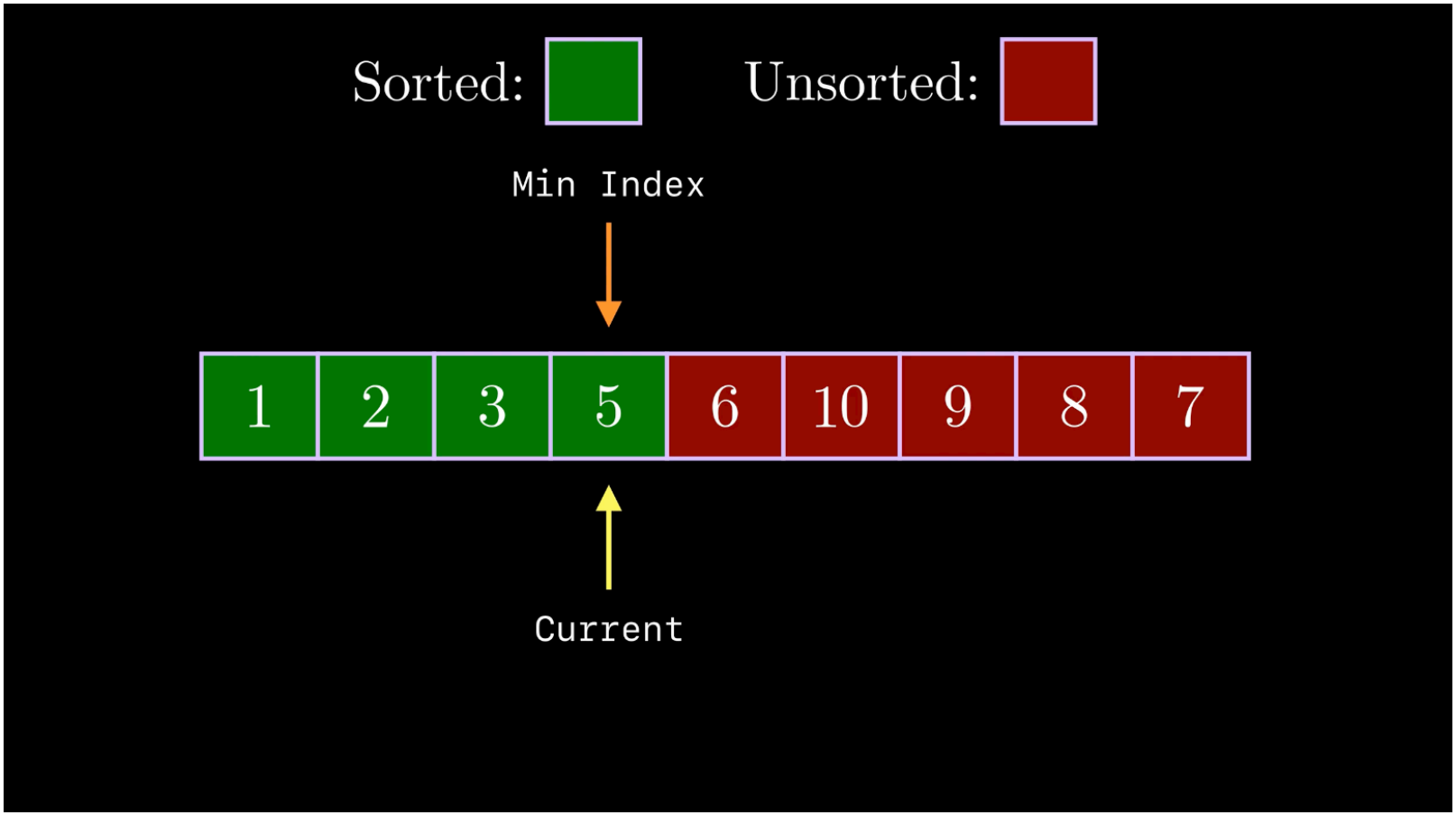
Selection Sort



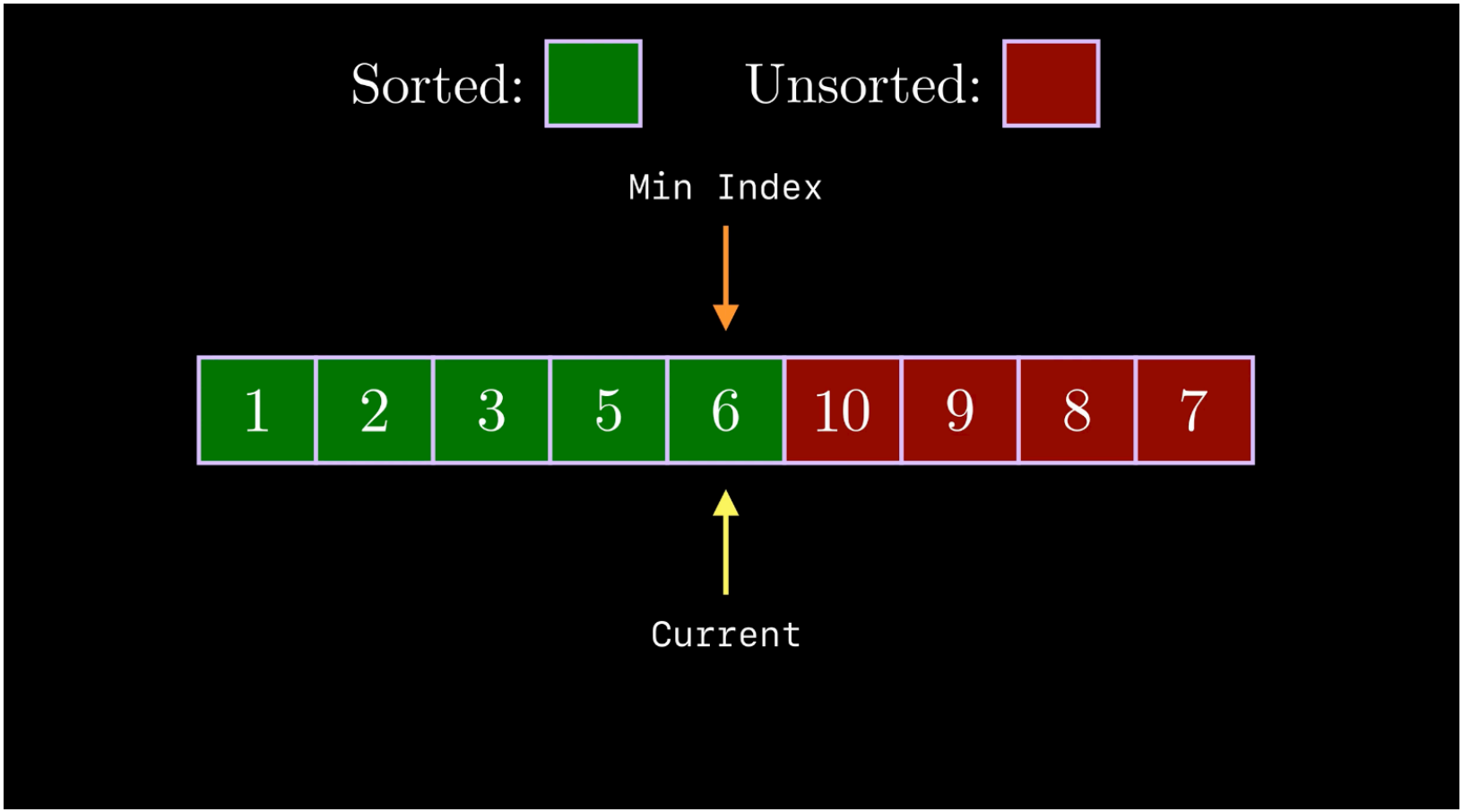
Selection Sort



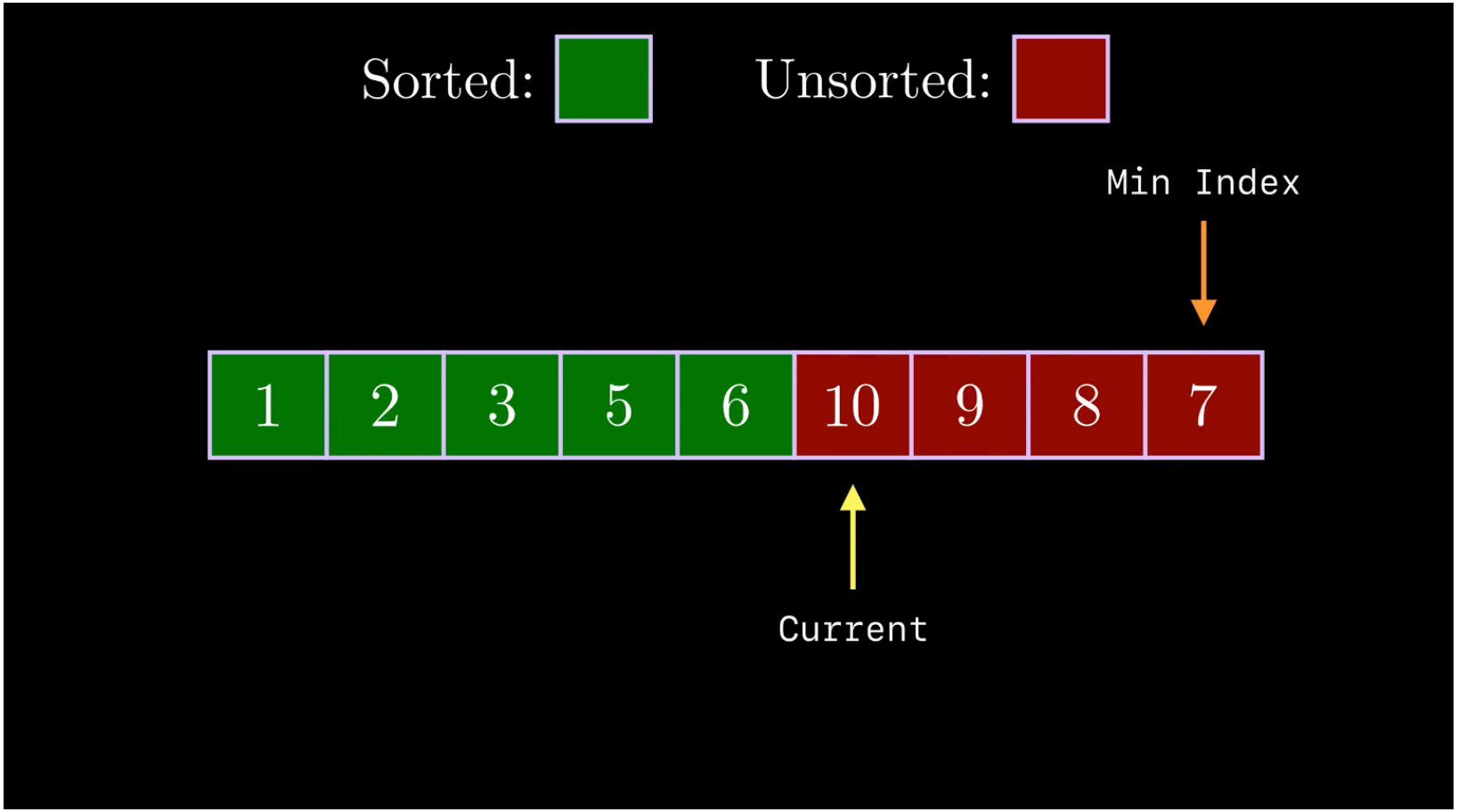
Selection Sort



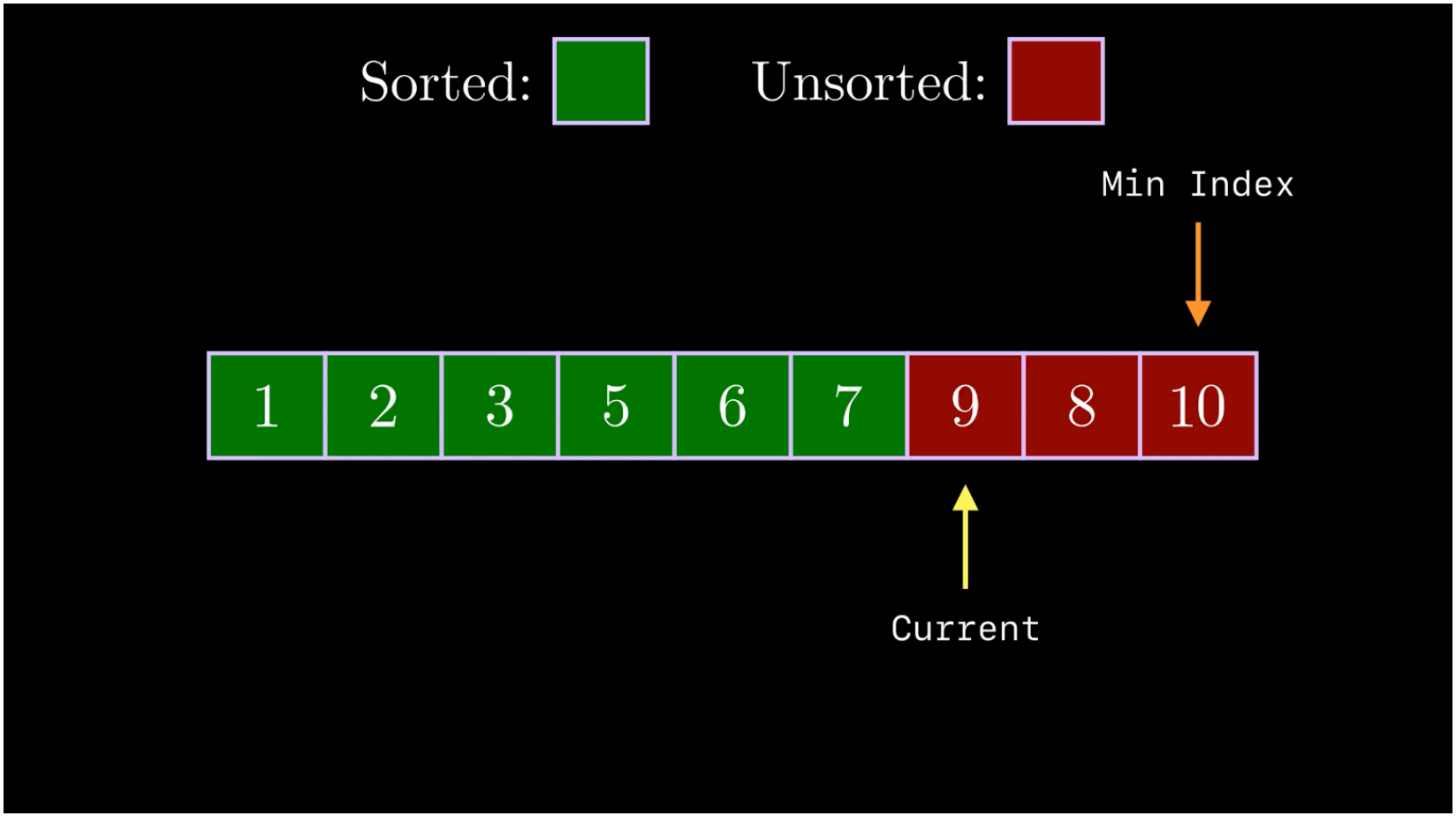
Selection Sort



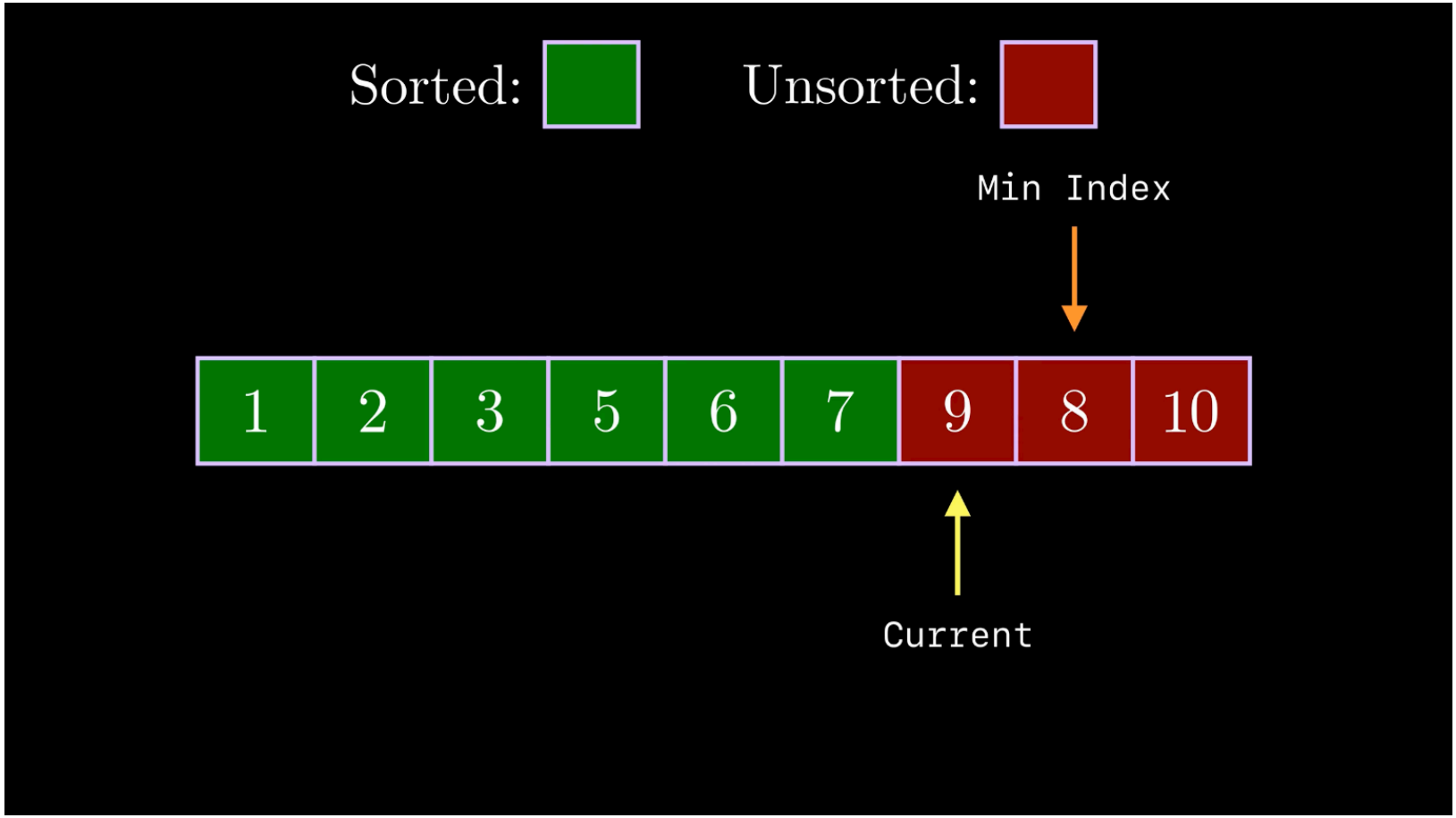
Selection Sort



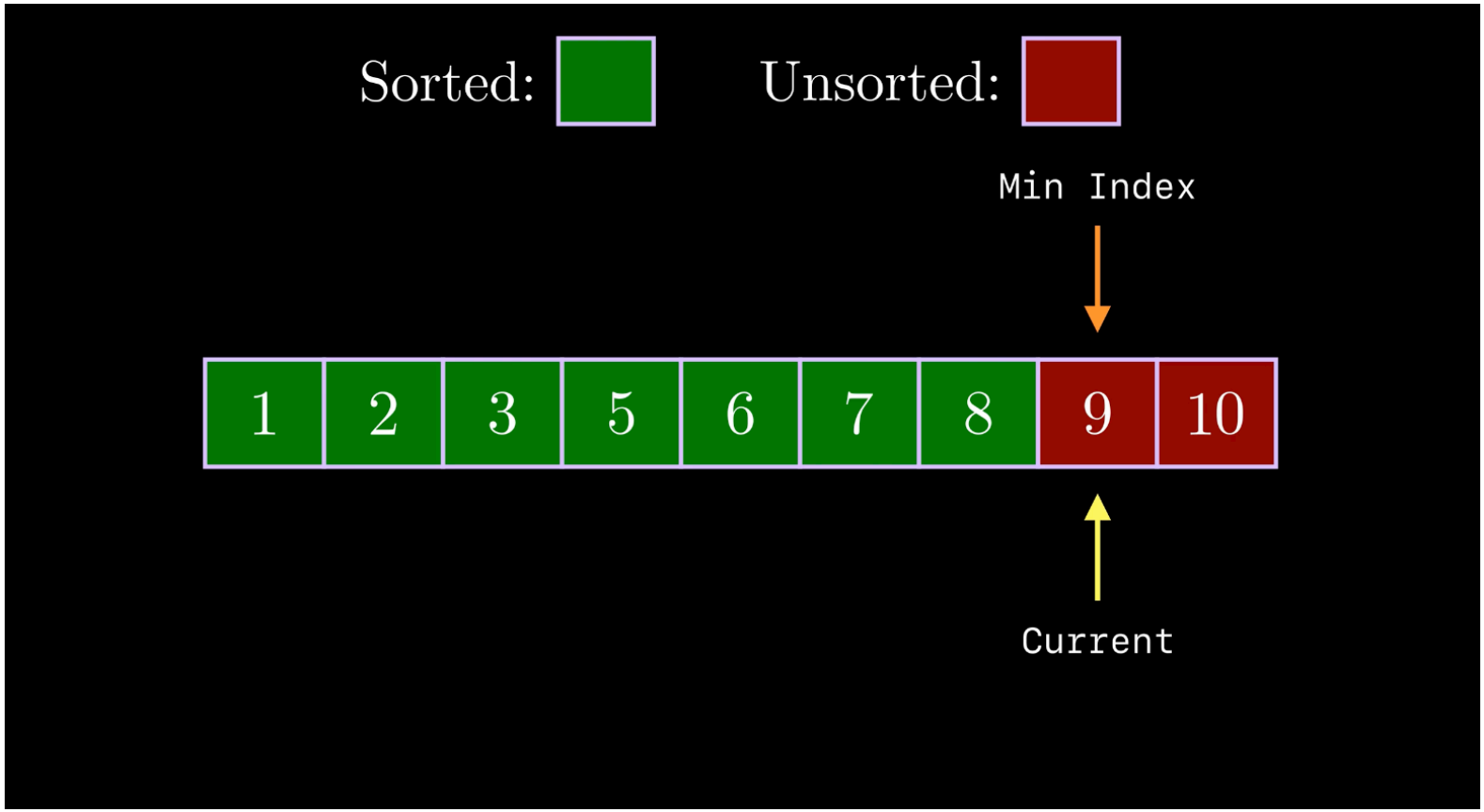
Selection Sort



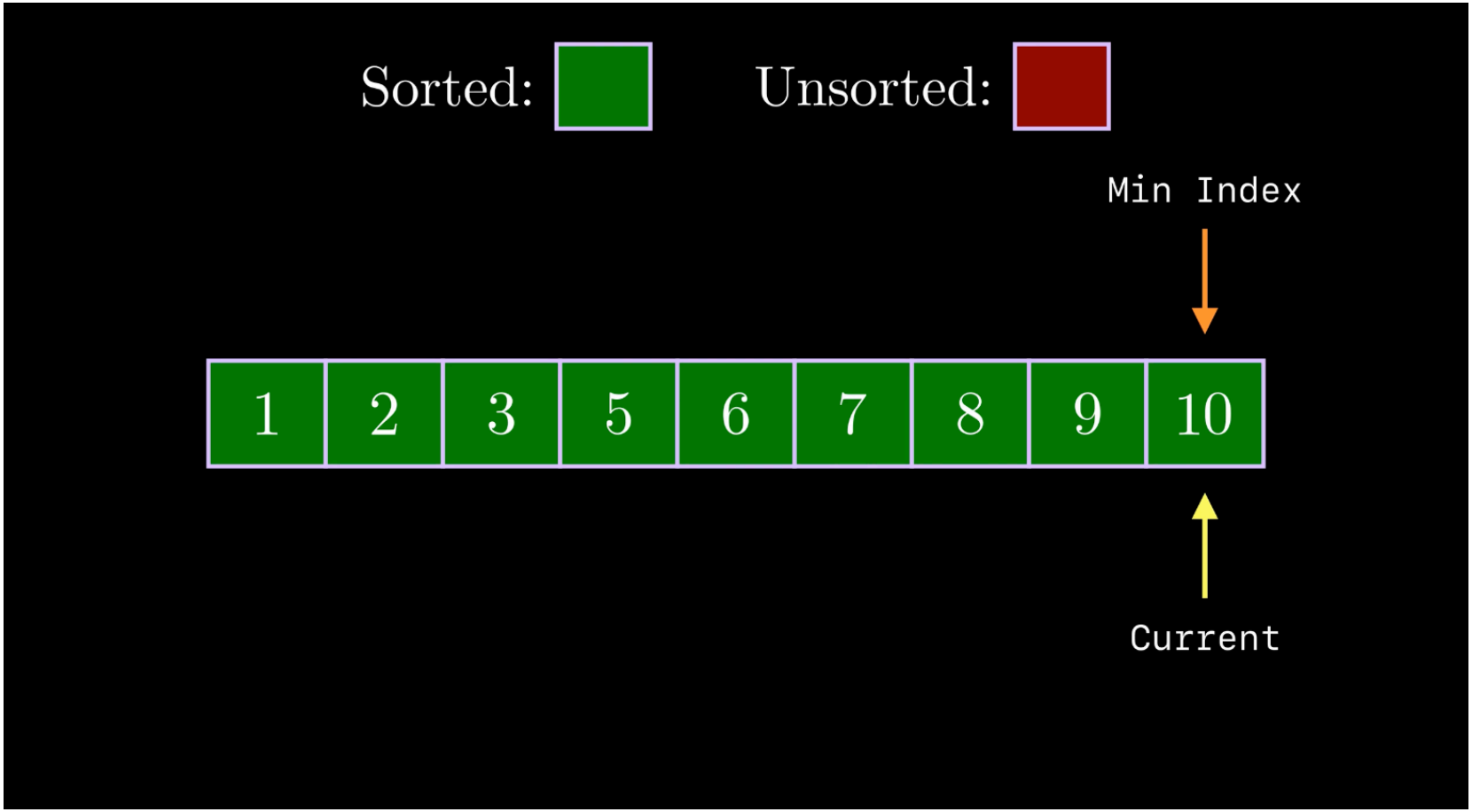
Selection Sort



Selection Sort



Selection Sort



Selection Sort

```
class Solution:
    def selection_sort(self, lst: List[int]) -> None:
        """
        Mutates lst so that it is sorted via selecting the minimum element and
        swapping it with the corresponding index
        """
        for i in range(len(lst)):
            min_index = i
            for j in range(i + 1, len(lst)):
                # Update minimum index
                if lst[j] < lst[min_index]:
                    min_index = j

            # Swap current index with minimum element in rest of list
            lst[min_index], lst[i] = lst[i], lst[min_index]
```

Thank You.

Contact : amisha.gupta@wilp.bits-pilani.ac.in