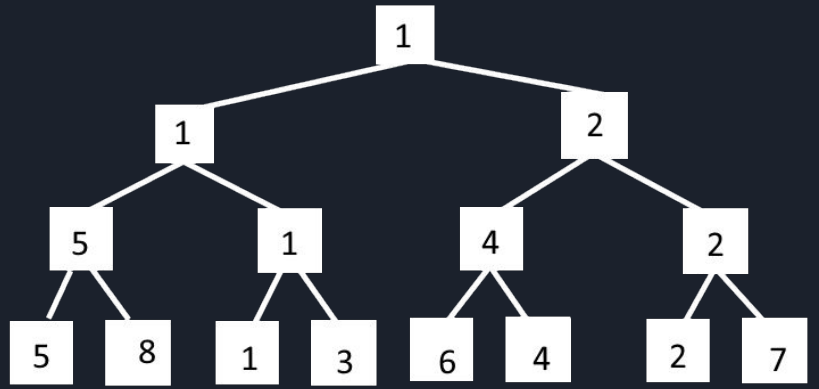
# Progteam Spring Week 7

Segment Trees

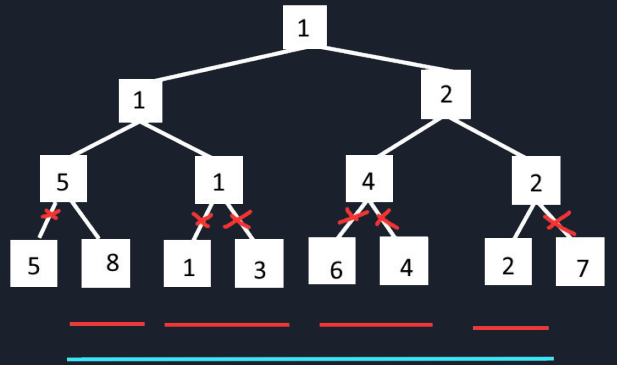
# What is a segment tree?

- Data structure with “left” and “right” nodes
- Stores information about all of its children
- Example: Minimum-Value Segment Tree



# Range Queries: Why is this helpful?

- With some array, we may want to know information about a subarray (a range from L to R)
  - Examples include sum of elements, maximum element, etc.
- Segment Trees let us answer these queries in O(N log N)
  - Approach: stop at node when all children are in range
  - Don't process node if none of its children are in range



# Range Queries

In Pseudocode:
(Assuming function is in a class with variables: node.min_val, node.left, node.right)

```
query(L, R):
    if all elements of node in range [L,R]:
        return min_val;
    if left not in range:
        return right.query(L,R)
    if right not in range:
        return left.query(L,R)
    return min(left.query(L,R), right.query(L,R))
```

Same idea but shorter:
Let the range of a node be [node.a, node.b]

```
query(node, L, R):
    if a <= L and b >= R:
        return.min_val;
    if a > R or b < L:
        # Out of bounds!
        return -INFINITY;
    return min(left.query(L,R), right.query(L,R))
```

# Updating a value

- It’s easiest to update one value at a time

```
update(position, value):
    if a == b:
        # Terminal node here!
        min_val = value; # Depending on context, you may not want to overwrite old
        return;
    mid = (a + b) / 2;
    if position <= mid: left.update(position, value);
    else: right.update(position, value);
    min_val = min(left.min_val, right.min_val);
```

# Example: Count Inversions

Problem: Count the number of inversions in an array, A

Assume n <= $10^5$, for all i, 1 <= A[i] <= n

Definition: An *inversion* is an instance of two indices [i,j] such that:

i < j

A[i] > A[j]

Example:

A = [1,5,4,2,7,3] (0-indexed)

A[1], A[5] is an inversion

(5, 3)

A[2], A[3] is an inversion

(4, 2)

# Example: Count Inversions

Approach: We can do this directly with a segment tree that counts the number of elements in a range:

```
# ... In t.update:
  if a == b:
      sum += 1; # This tree keeps track of a sum for every range
      return;
#....

ans = 0
Tree t(1, n);
for i = [1...n]:
    ans += t.query(a[i] + 1, n); # Add one inversion for all elements > a[i]
    t.update(a[i]); # Add one to the index of a[i]
```

# Example: K-th Largest

- Most things that you can do with a BBST, you can also do with a Segment Tree
- Classic example: K-th largest element
  - If you know all elements are in a range [1…n], keep track of the size of each subtree
- Codes of these two problems can be found on the Github under “examples”