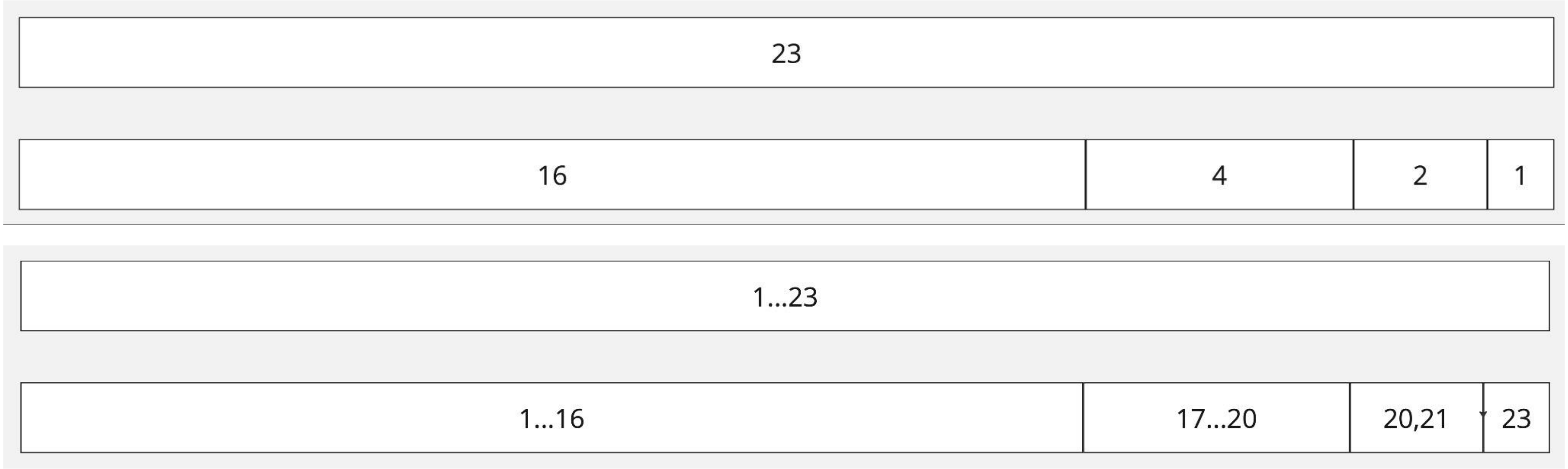
Binary Lifting

The Idea

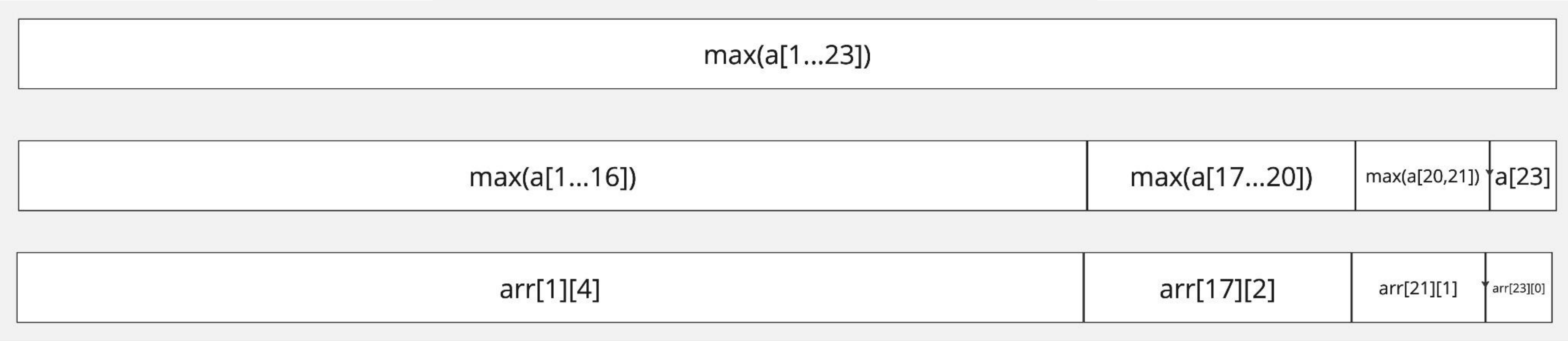
$$3^{23} = 3^{16} \times 3^{4} \times 3^{2} \times 3^{1}$$



The Idea

If we have a precomputed array which stores info about all ranges with length 2^j from each starting index i:

$$arr[i][j] = \max_{i \leq k < i+2^j} a[i]$$



How to Compute arr[i][j]?

Suppose we have already computed arr[x][j-1] for all starting indices x, computing arr[i][j] is trivial:

<table>
<tr><td>max(a[i...i+2^(j-1)-1])</td><td>max(a[i+2^(j-1)...i+2^j-1])</td></tr>
<tr><td>arr[i][j-1]</td><td>arr[i+2^(j-1)][j-1]</td></tr>
<tr><td colspan="2">arr[i][j]=max(arr[i][j-1], arr[i+2^(j-1)][j-1])</td></tr>
<tr><td colspan="2">max(a[i...i+2^j-1])</td></tr>
</table>

Code (Precompute)

```
for (int i = 1; i <= n; i++) st[i][0] = a[i];

for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= log2(n); j++) {
        st[i][j] = max(st[i][j - 1], st[i + (1 << j - 1)][j - 1]);
    }
}
```

Code (Range Query)

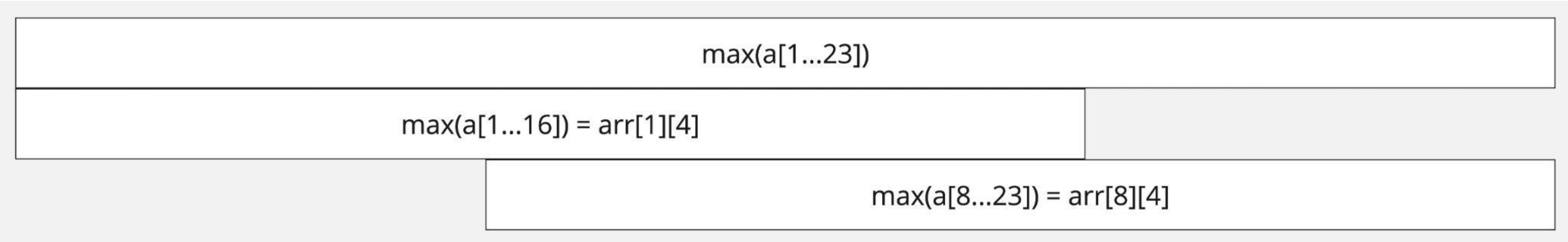
```
int k = r - l + 1, ans = 0;

for (int i = log2(n); i >= 0; i--) {
    if (k & (1 << i)) {
        ans = max(ans, st[l][i]);
        l += 1 << i;
    }
}
```

Special Case: Max/Min Query

You can do it in O(1)!

Trick: take the largest range from the beginning and largest range from the end



You can do this since taking max/min over duplicate entries doesn’t affect answer

But you can’t do this when the problem ask you to calculate the sum!

Binary Lifting on Trees

Each path from a leaf to the root is an array, where leaf is at index 1 and root is at index n.

But how do we know the index of i+2^j? We only know the next index (i.e. father of current node).

Similar idea to how we computed arr[i][j]!

Computing i+2^j

$$next[i][j] = \text{node that is } i + 2^j$$

$$next[i][j] = \begin{cases} \text{father of i} & \text{if } j = 0 \\ next[next[i][j-1]][j-1] & \text{otherwise} \end{cases}$$

Interpretation: jump 2^(j-1) indices from i first, and then jump another 2^(j-1) indices

Code (Precompute)

```
void dfs(int u, int fa) {
    nxt[u][0] = fa, arr[u][0] = a[u], dep[u] = dep[fa] + 1;

    for (int i = 1; i <= log2(n); i++) {
        nxt[u][i] = nxt[nxt[u][i - 1]][i - 1];
        arr[u][i] = max(arr[u][i - 1], arr[nxt[u][i - 1]][i - 1]);
    }

    for (int v : e[u]) {
        if (v != fa) {
            dfs(v, u);
        }
    }
}
```

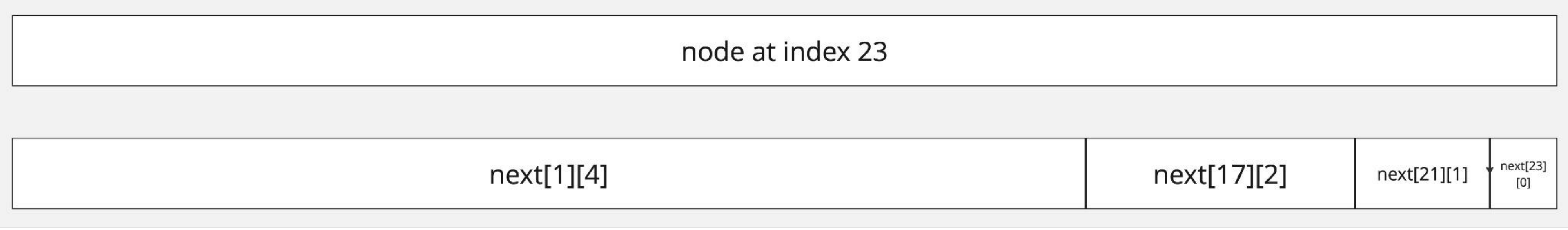
Finding LCA

Given two nodes u and v, how to find their lowest common ancestor if we have computed next[i][j] and arr[i][j] on tree?

First, we need to make sure u and v are on the same level

If we know the depth of u and v and computed next[i][j], this would be fairly easy

Exact the same as what we have done in previous slides!



Code (Step 1)

```
if (dep[u] > dep[v]) swap(u, v);

int diff = dep[v] - dep[u];

for (int i = log2(n); i >= 0; i--) {
    if (diff & (1 << i)) {
        ans = max(ans, arr[v][i]);
        v = nxt[v][i];
    }
}
```

Finding LCA (cont'd)

Now, we know that u and v are on the same level in tree. We know that a node x is an ancestor of LCA if x=next[u][k]=next[v][k] for same integer k

We find the largest value of k such that next[u][k] ≠ next[k] and jump u and v by 2^k

Repeat this process until u=v

Code (Step 2)

```
for (int i = log2(n); i >= 0; i--) {
    if (nxt[u][i] != nxt[v][i]) {
        ans = max(ans, arr[u][i]);
        u = nxt[u][i];
        ans = max(ans, arr[v][i]);
        v = nxt[v][i];
    }
}

if (u != v) {
    ans = max(ans, arr[u][0]);
    u = nxt[u][0];
    ans = max(ans, arr[v][0]);
    v = nxt[v][0];
}
```

Comparison between RMQ Data Structures on Array

Operations	Data Structure(s)	Time Complexity
Point update Point query	Array Vector	No preprocessing O(1) update O(1) query
No update Range query	Binary lifting	O(nlogn) preprocess O(nlogn) update O(1)/O(nlogn) query
Point update Range query	Fenwick tree Segment tree	O(nlogn) preprocess O(logn) update O(logn) query
Range query Range update	Lazy segment tree	O(nlogn) preprocess O(logn) update O(logn) query

Comparison between RMQ Data Structures on Tree

Operations	Data Structure(s)	Time Complexity
Point update Point query	Tree	No preprocessing O(1) update O(1) query
No update Range query	Binary lifting on tree	O(nlogn) preprocess O(nlogn) update O(nlogn) query
Range query Range update	Heavy-light decomposition Splay tree Link-cut tree	O(nlogn) preprocess O(logn) update O(logn) query