# Query/Update Operations

Ashar Fuadi

# Factors

- Update: point/range?
- Query: point/range?
- Ordering: offline/online?

We will consider **increase** update and **sum** query.

# Combination #1

- Update: **point**
- Query: **point**
- Ordering: **offline** & **online**

Solution: **trivial**

# Combination #2

- Update: **point**
- Query: **range**
- Ordering: **offline**

Solution: **partial sum**

# Partial Sum

**1D**

sum[i] = **sigma** data[1..i] = data[i] + sum[i-1]

query(a..b) = sum[b] – sum[a-1]

**2D**

sum[i][j] = **sigma** data[1..i][1..j] = data[i][j] + sum[i-1][j] + sum[i][j-1] – sum[i-1][j-1]

query(a..b, c..d) = sum[b][d] – sum[a-1][d] – sum[b][c-1] + sum[a-1][c-1]

**3D**

?

# Combination #3

- Update: **range**
- Query: **point**
- Ordering: **offline**

Solution: **partial difference**

# Partial Difference

```
diff[i] = data[i] – data[i-1]
update(a..b) → { diff[a]++; diff[b+1]--; }
```

After all updates:

```
for (int i = 1; i <= N; i++)
    data[i] = diff[i] + data[i-1]
```

# Combination #4

- Update: **range**
- Query: **range**
- Ordering: **offline**

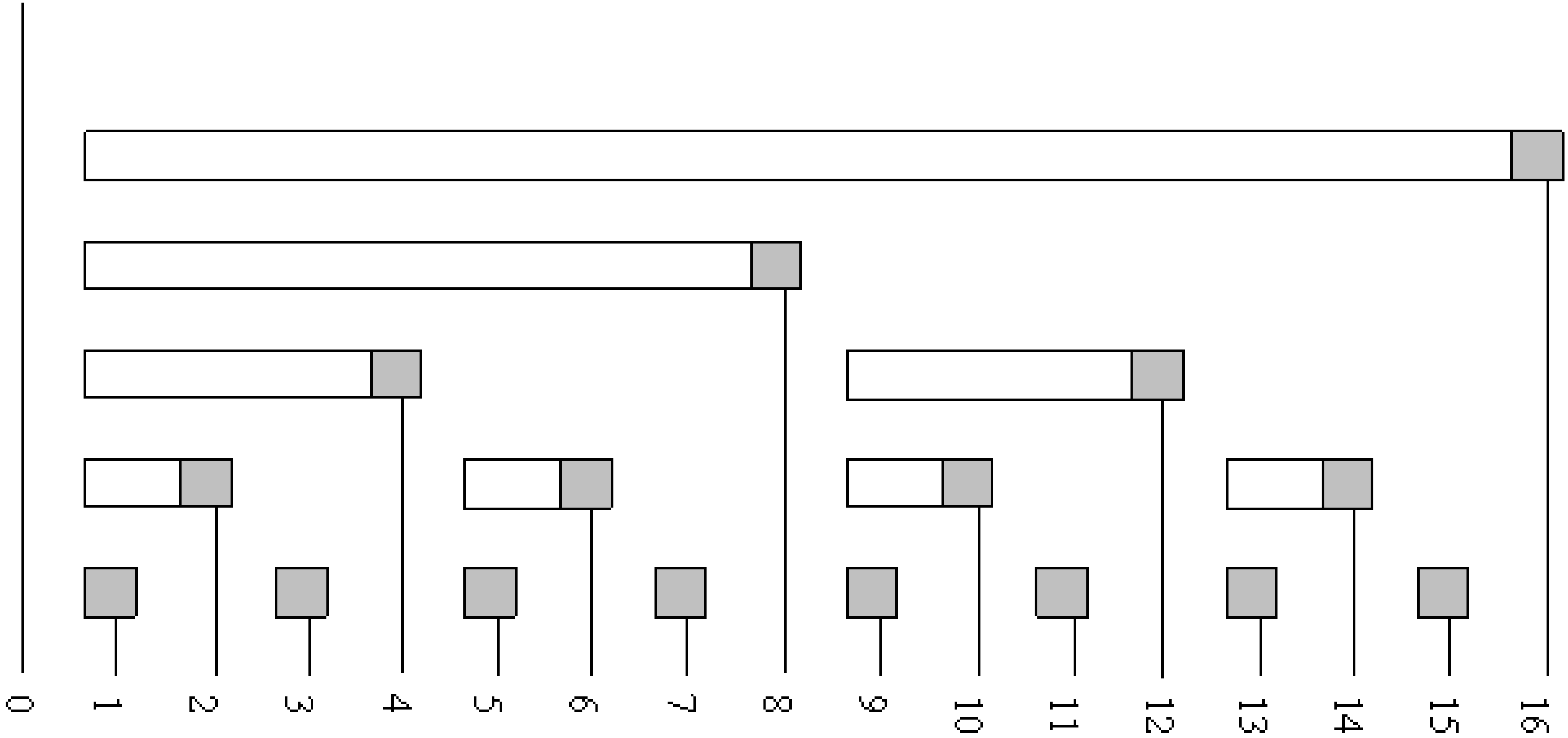
Solution: **partial difference** then **partial sum**

# Combination #5

- Update: **point**
- Query: **range**
- Ordering: **online**

Solution: **fenwick (binary indexed) tree**

# Binary Indexed Tree



*source: http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=binaryIndexedTrees*

# Binary Indexed Tree

query(1 .. 0b1101) = bit[0b1101] +
bit[0b1100] +
bit[0b1000] +
bit[0b0000]

query(a .. b) = query(1 .. b) – query(1 .. a-1)

# Binary Indexed Tree

query(1 .. x) →

```
int res = 0;
for (int i = x; i; i -= i & -i)
    res += bit[i];
return res;
```

# Binary Indexed Tree

update(x) →

```
for (int i = x; i <= N; i += i & -i)
    bit[i]++;
```

# Combination #6

- Update: **range**
- Query: **point**
- Ordering: **online**

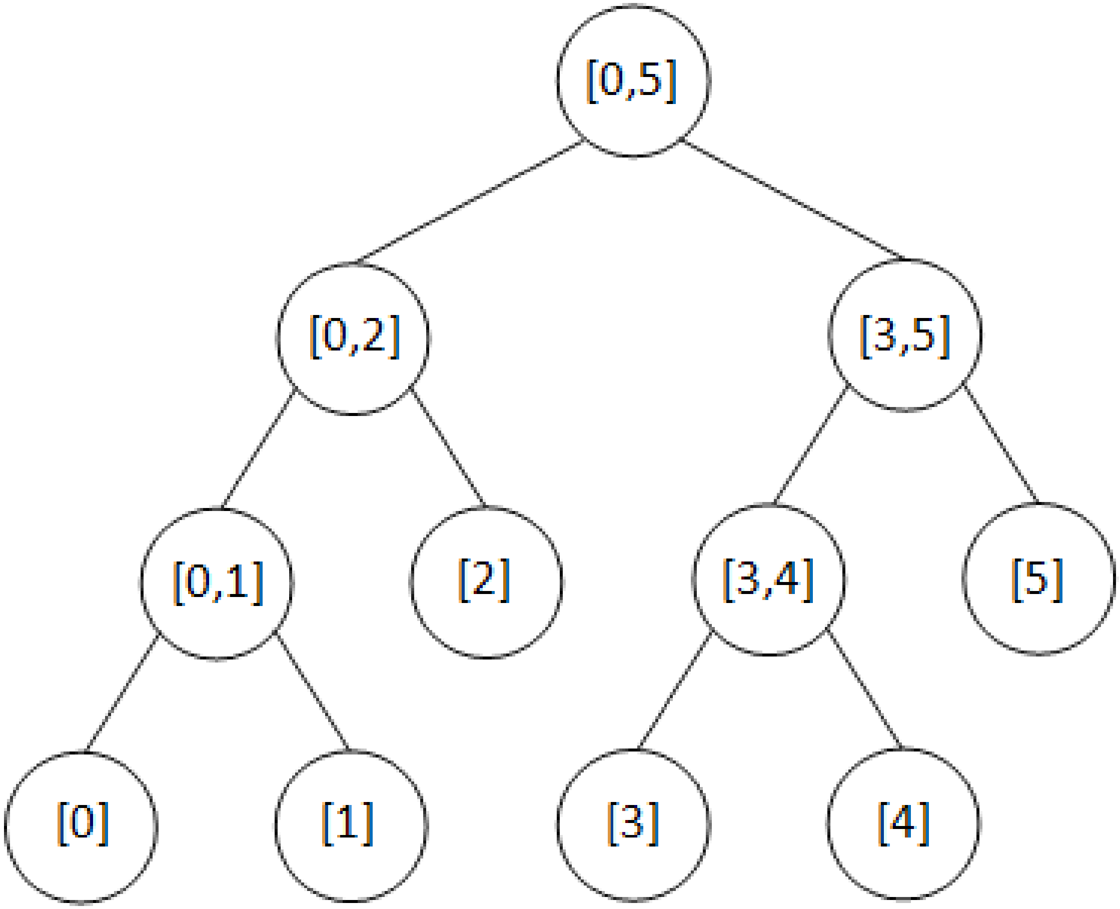
Solution: **partial difference + BIT**

# Combination #5 Revisited

- Update: **point**
- Query: **range**
- Ordering: **online**

Another solution: **basic segment tree**

# Basic Segment Tree



*source: https://thanabhat.wordpress.com/2011/02/12/segment-tree/*

# Basic Segment Tree

```
// call: query(1, 1, N, a, b)
int query(int node, int b, int e, int i, int j)
{
    if (e < i || j < b)
        return 0;
    else if (i <= b && e <= j)
        return tree[node];
    else
    {
        return query(2*node+0, b, (b+e)/2, i, j) +
        query(2*node+1, (b+e)/2+1, e, i, j);
    }
}
```

# Basic Segment Tree

```
// call: update(1, 1, N, x)
void update(int node, int b, int e, int x)
{
    if (e < i || j < b)
        return;
    else if (x == b && b == e)
        tree[node]++;
    else
    {
        update(2*node+0, b, (b+e)/2, x);
        update(2*node+1, (b+e)/2+1, e, x);
        tree[node] = tree[2*node+1] + tree[2*node+2]
    }
}
```

# Combination #7

- Update: **range**
- Query: **range**
- Ordering: **online**

Solution: **lazy segment tree**

# Lazy Segment Tree

```
// call: update(1, 1, N, a, b)
void update(int node, int b, int e, int i, int j)
{
    if (e < i || j < b)
        return;
    else if (i <= b && e <= j)
        todo[node]++;
    else
    {
        propagate(node);
        update(2*node+0, b, (b+e)/2, i, j);
        update(2*node+1, (b+e)/2+1, e, i, j);
        combine(node, b, e);
    }
}
```

# Lazy Segment Tree

```
void propagate(int node)
{
    todo[2*node+0] += todo[node];
    todo[2*node+1] += todo[node];
    todo[node] = 0;
}

int value(int node, int b, int e)
{
    return tree[node] + todo[node] * (e - b + 1);
}

void combine(int node, int b, int e)
{
    tree[node] = value(2*node+0, b, (b+e)/2) +
                 value(2*node+1, (b+e)/2+1, e);
}
```

# Lazy Segment Tree

```
query?
```

# Combination #2 Revisited

- Update: **point**
- Query: **range**
- Ordering: **offline**

But, queries are max/min instead of sum!

Solution: **sparse table**