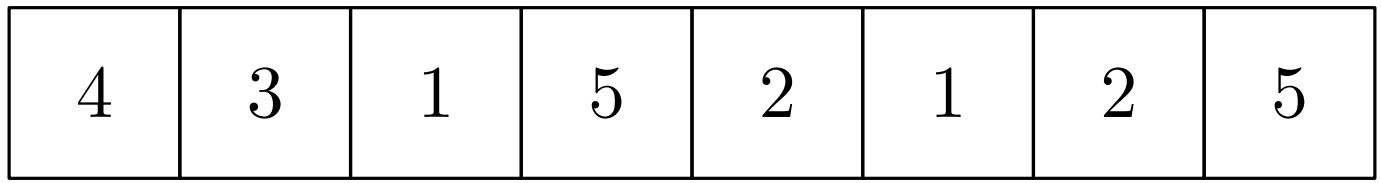
# Segmenttipuut

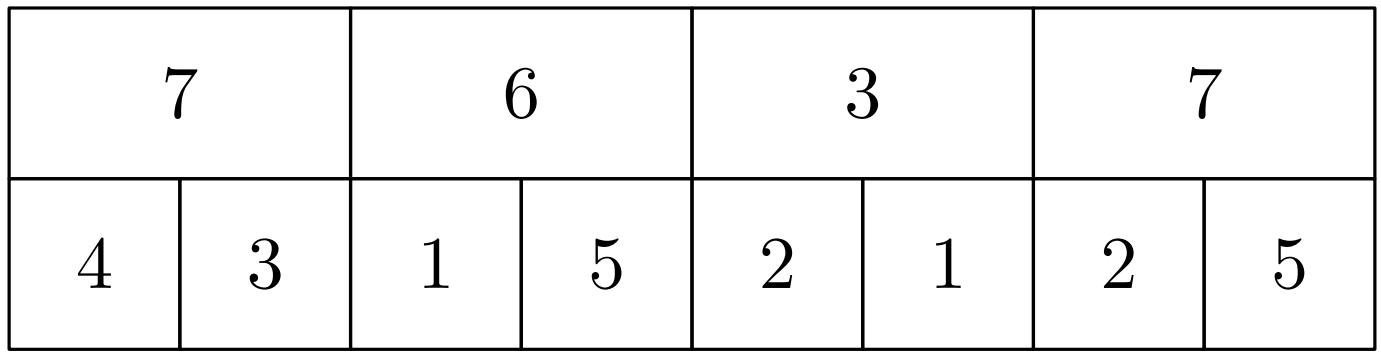
BOI-leiri 2024

Alkuperäiset diat: Topi Talvitie (2017)

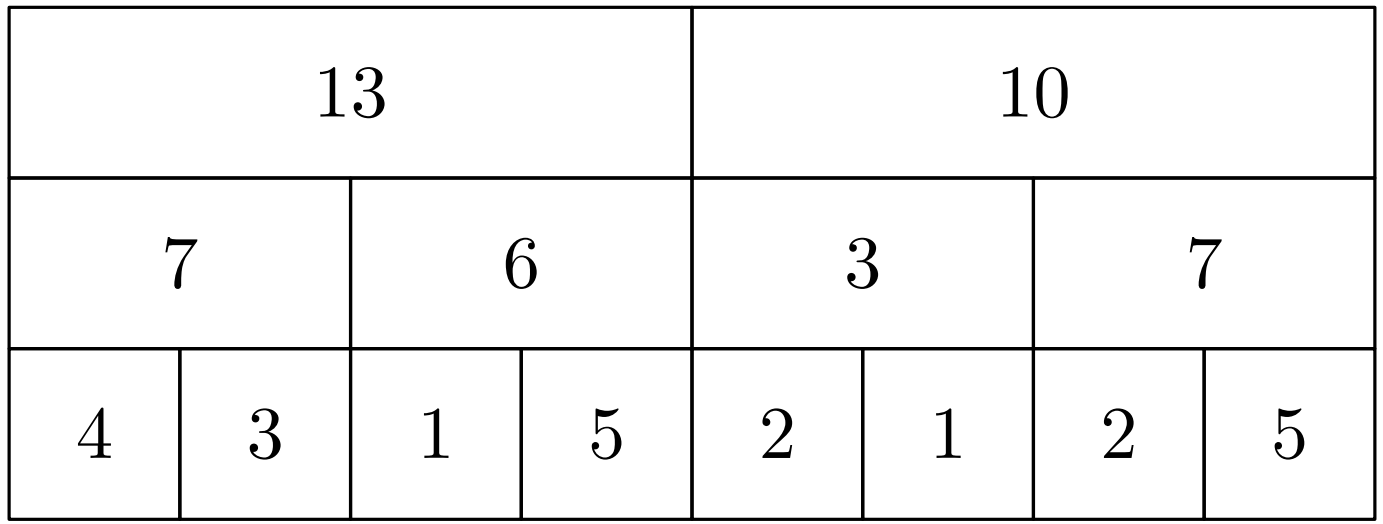
# Välikyselysegmenttipuu: summat



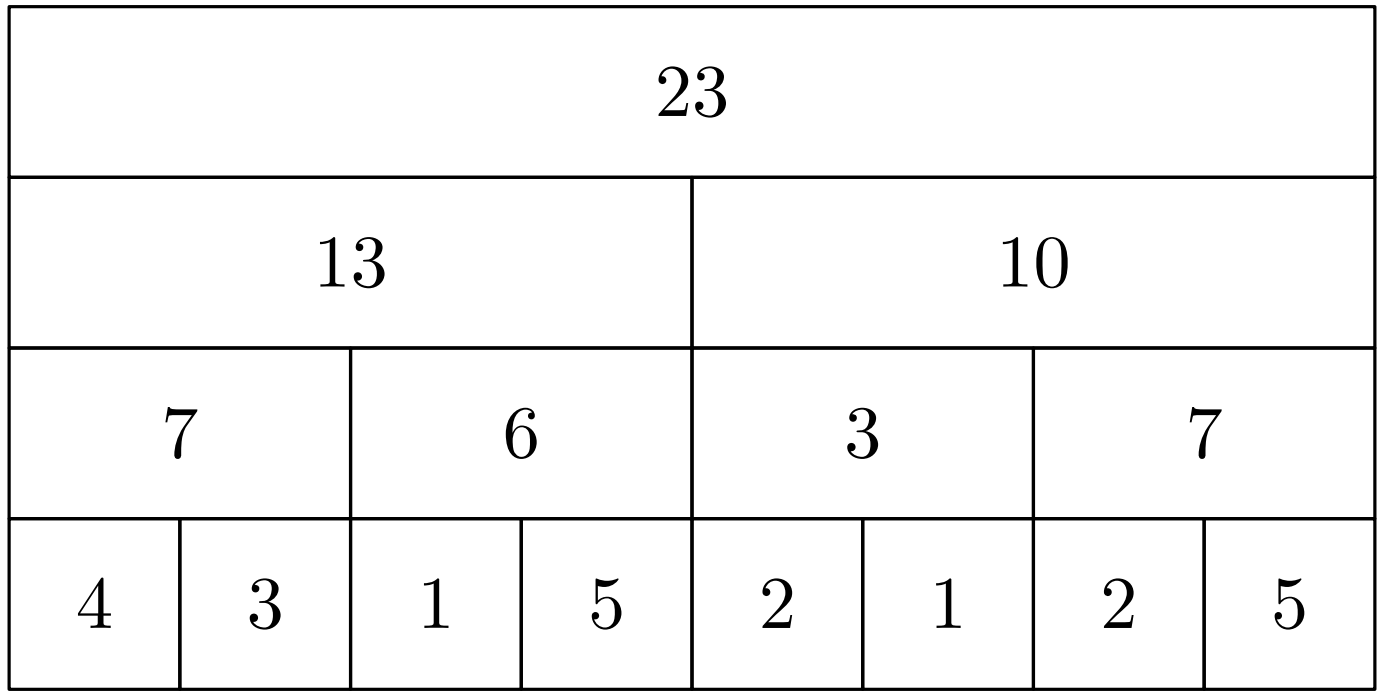
# Välikyselysegmenttipuu: summat



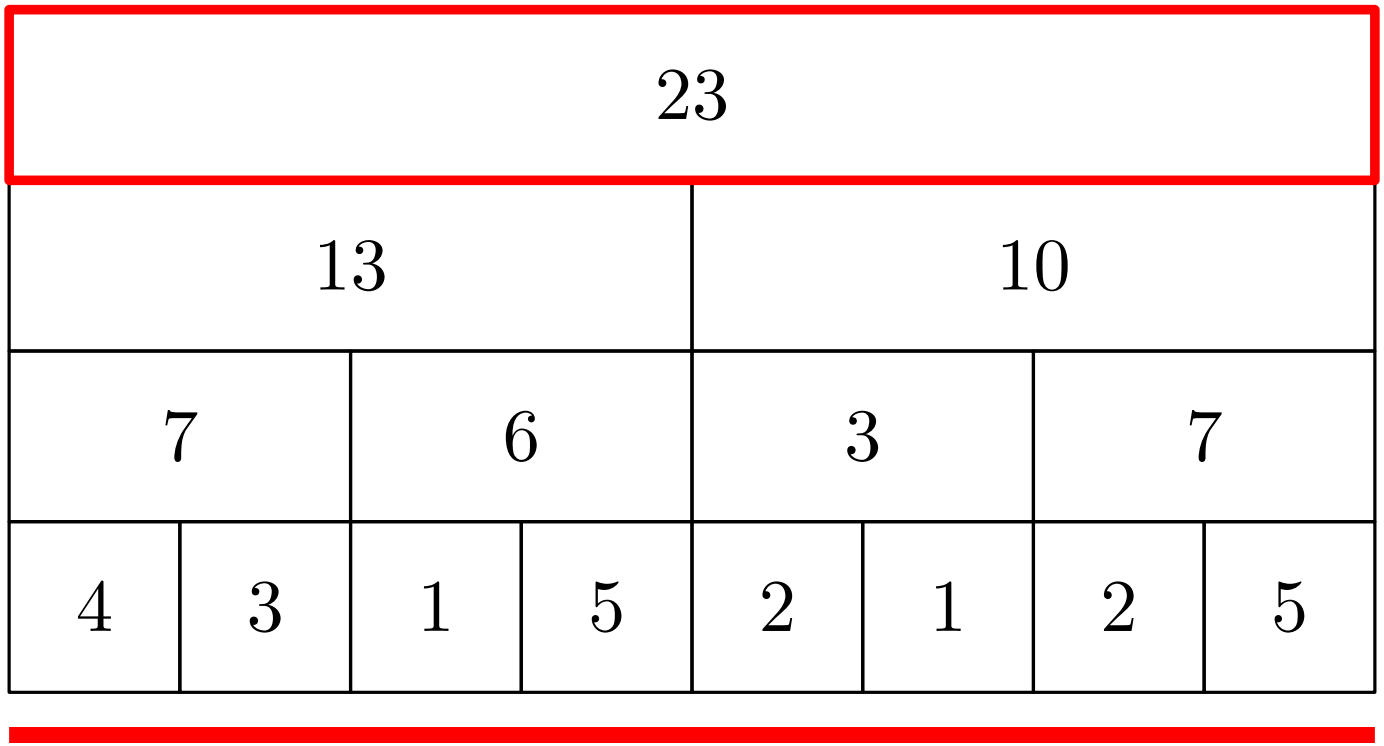
# Välikyselysegmenttipuu: summat



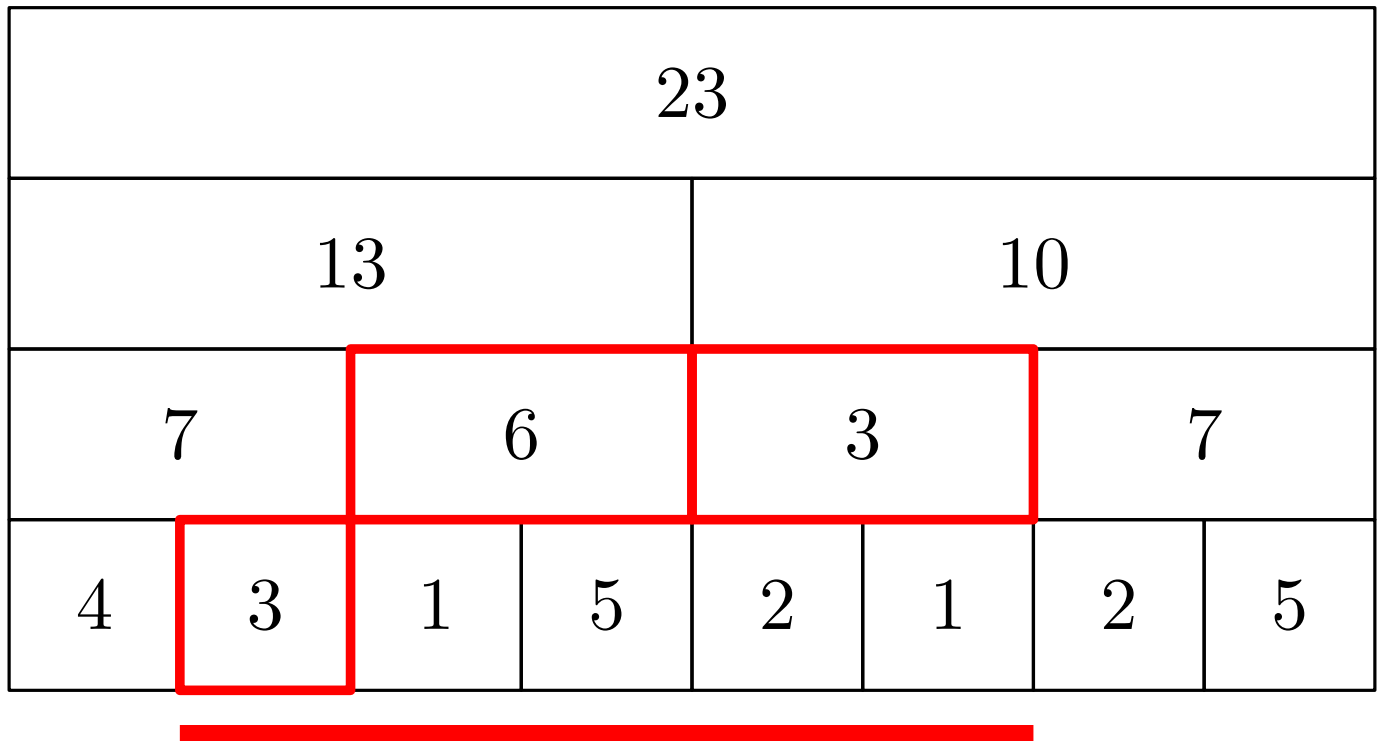
# Välikyselysegmenttipuu: summat



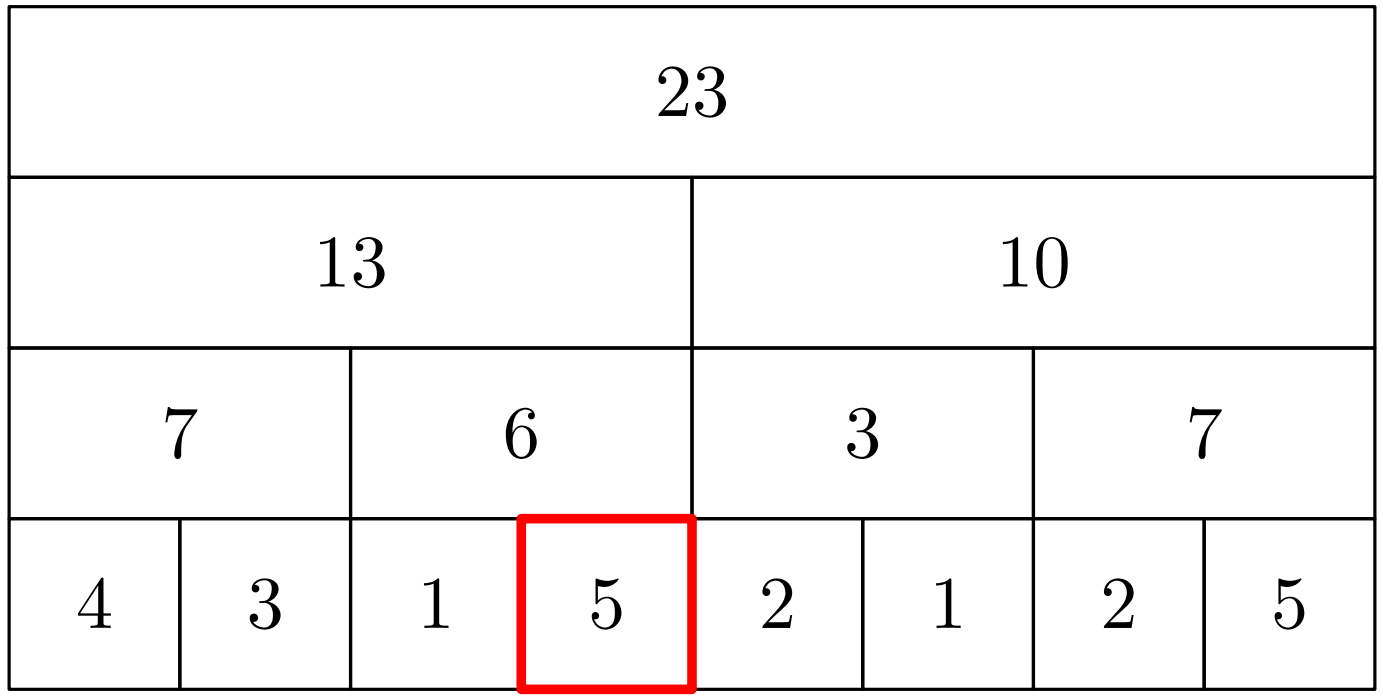
# Välikyselysegmenttipuu: summat



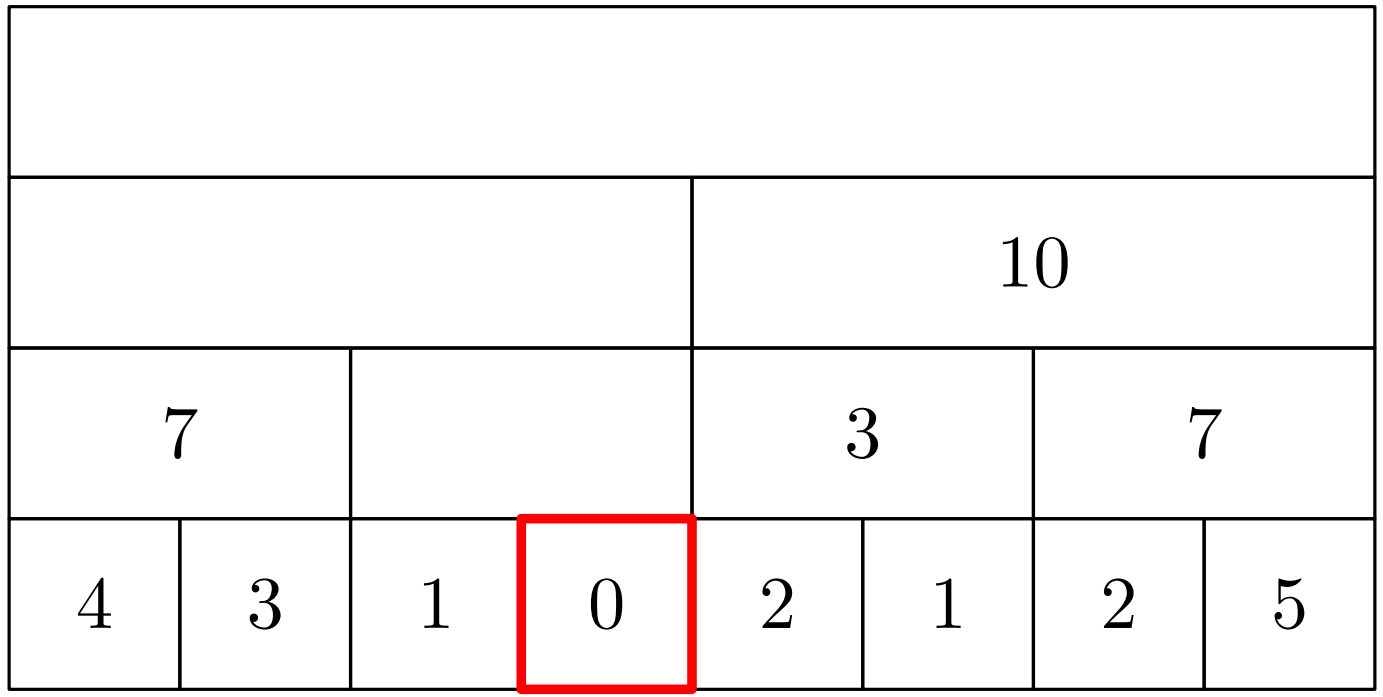
# Välikyselysegmenttipuu: summat



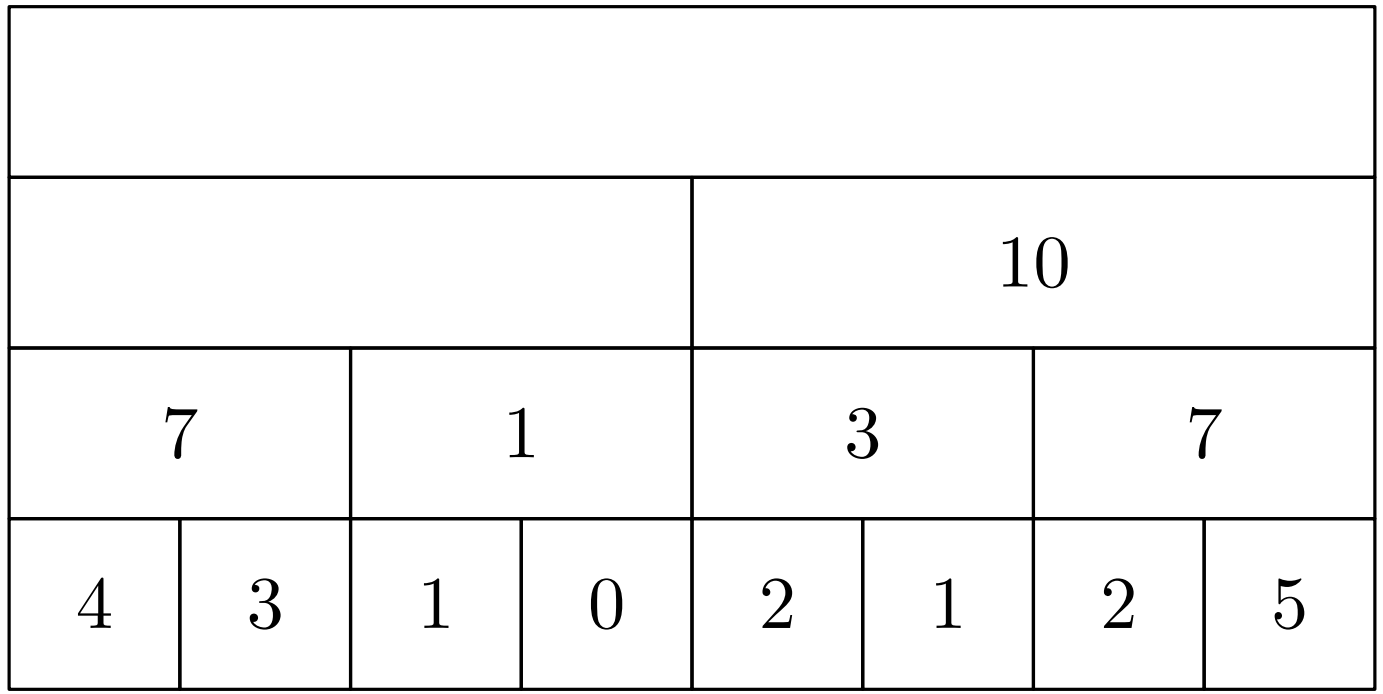
# Välikyselysegmenttipuu: summat



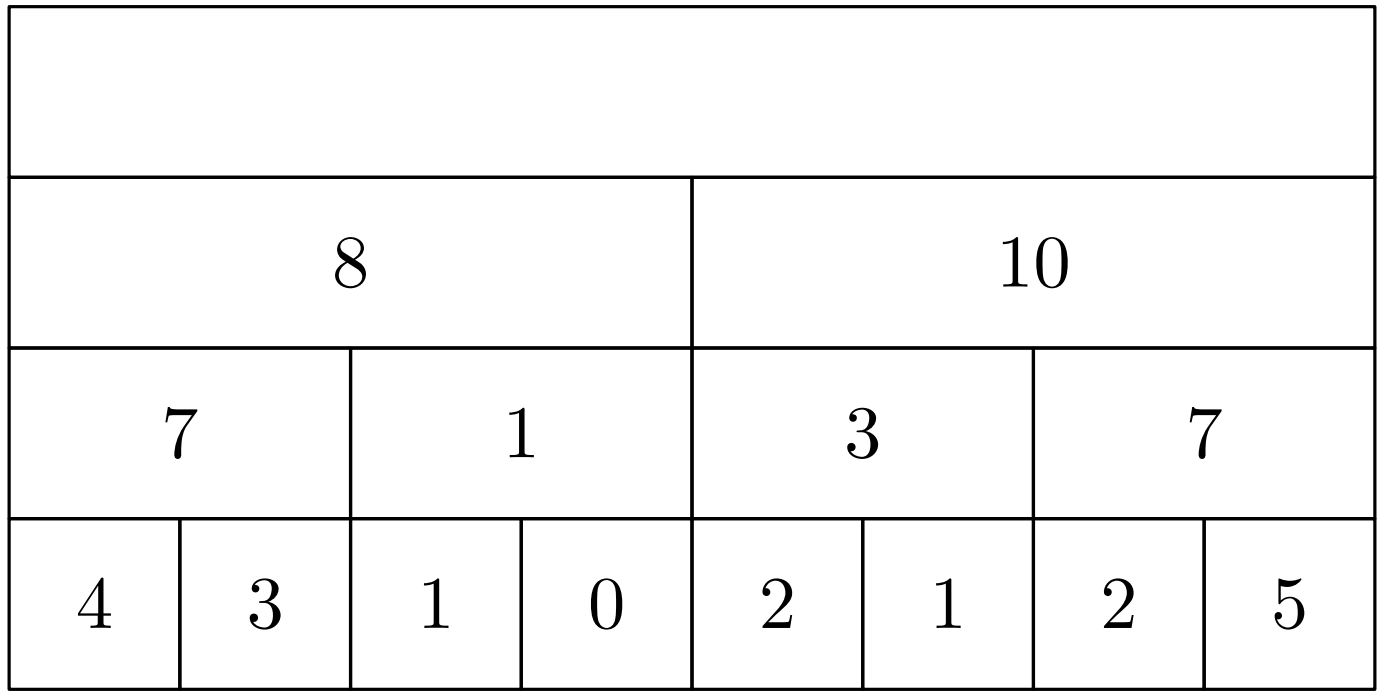
# Välikyselysegmenttipuu: summat



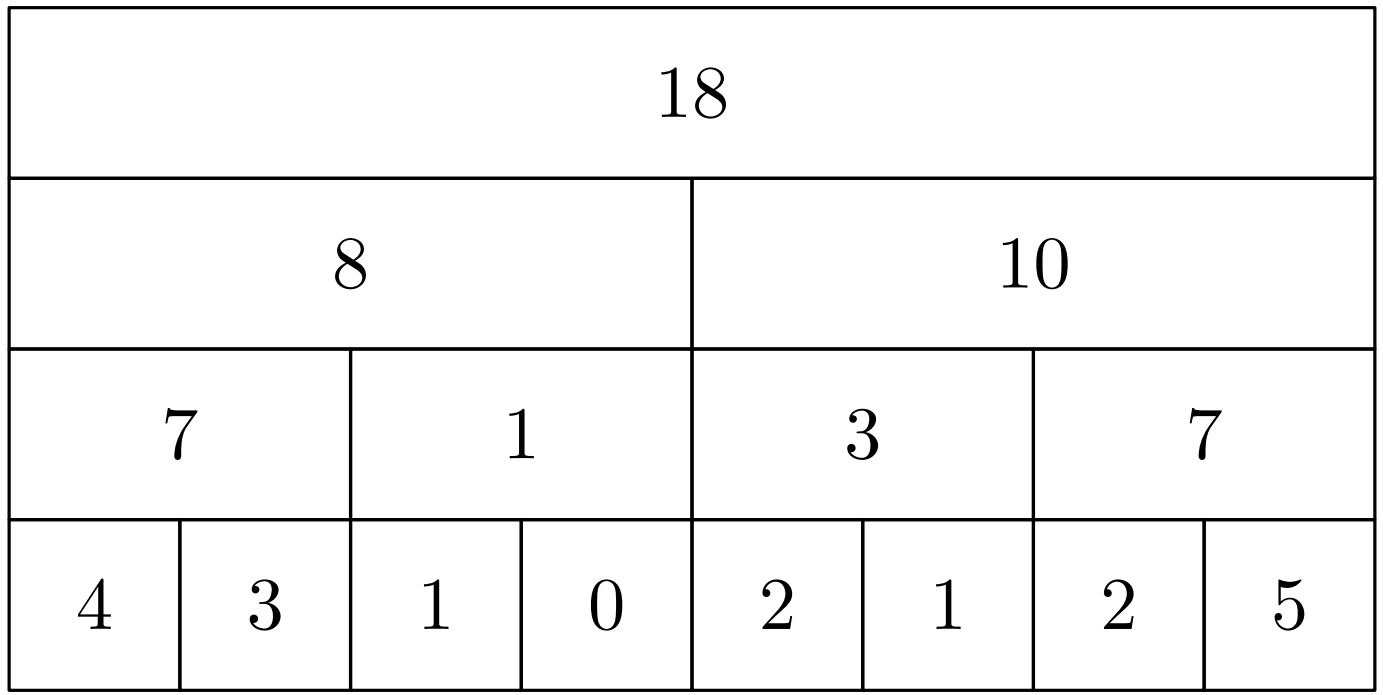
# Välikyselysegmenttipuu: summat



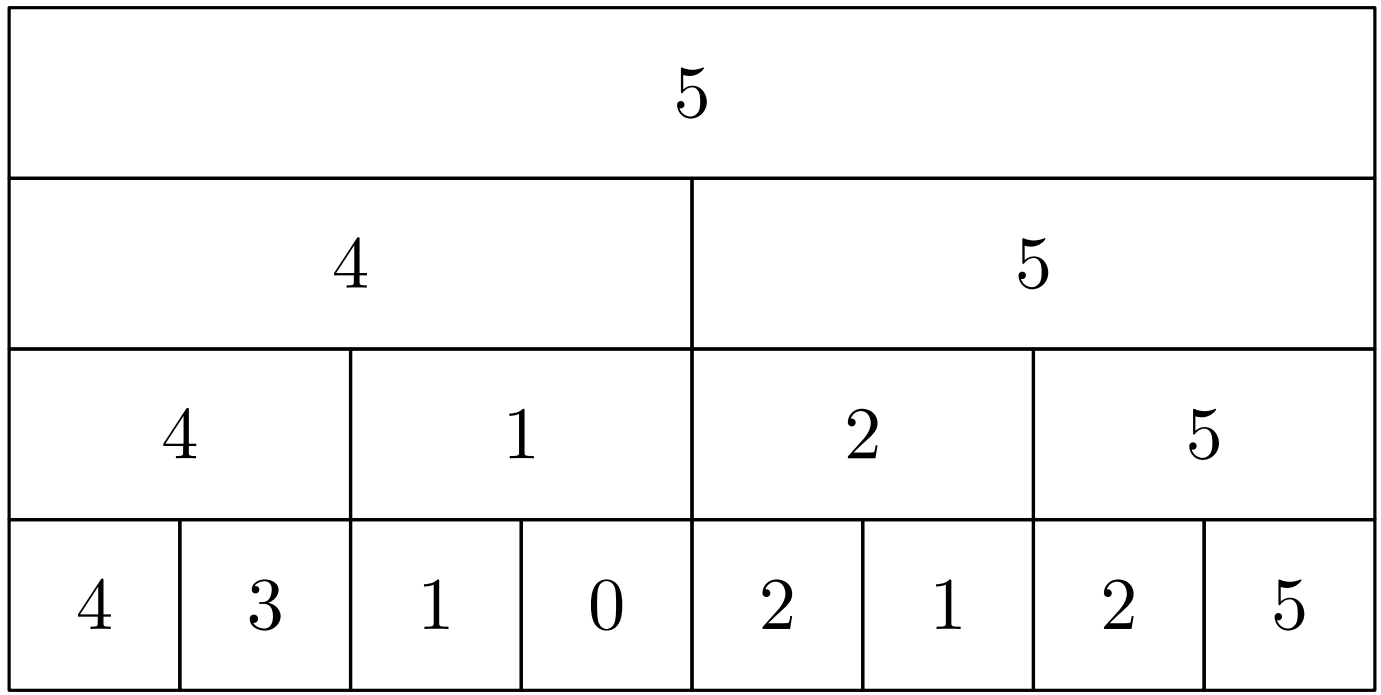
# Välikyselysegmenttipuu: summat



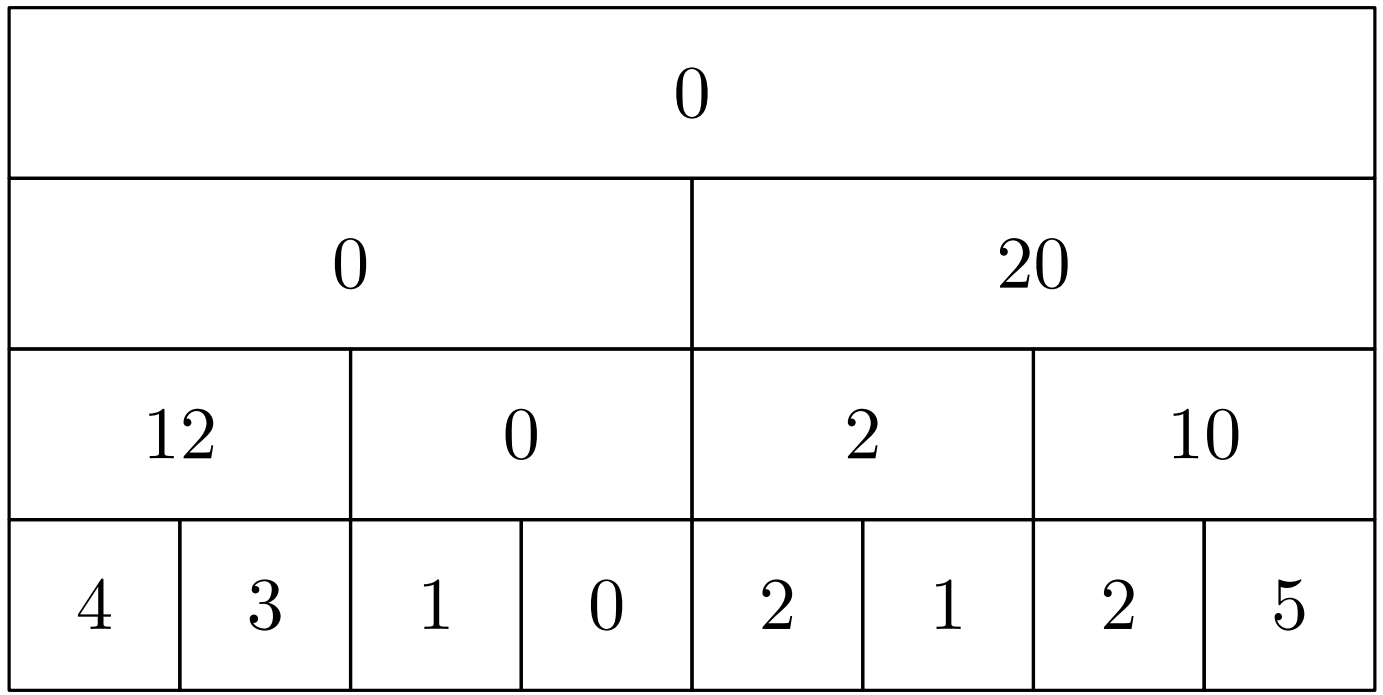
# Välikyselysegmenttipuu: summat



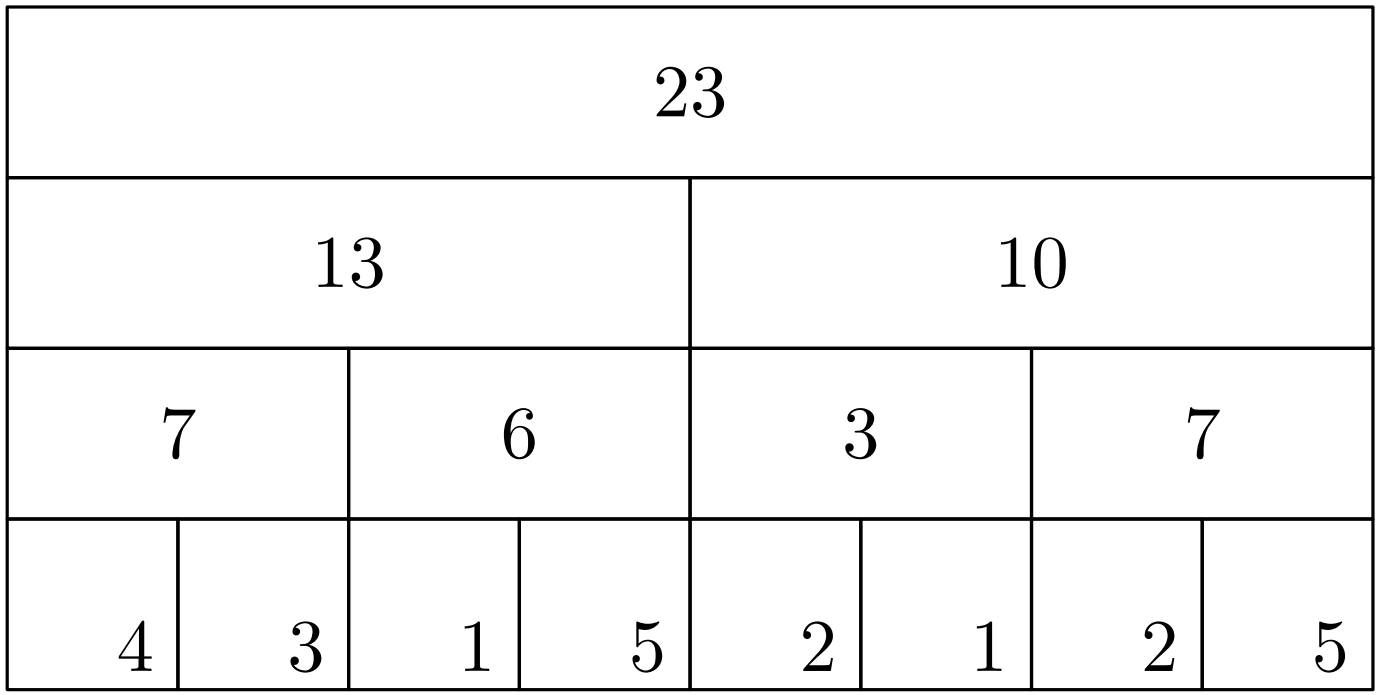
# Välikyselysegmenttipuu: maksimit



# Välikyselysegmenttipuu: kertolaskut

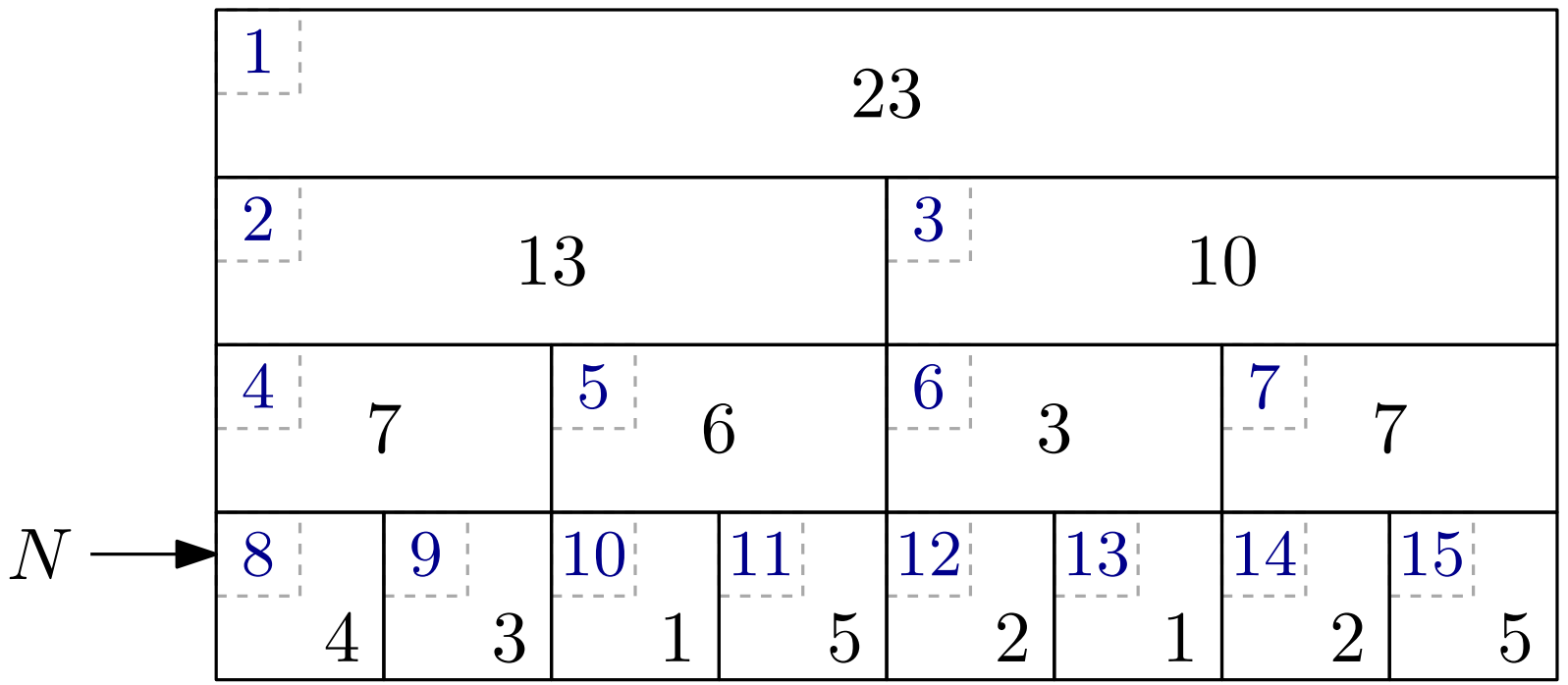


# Alhaalta ylös -toteutus



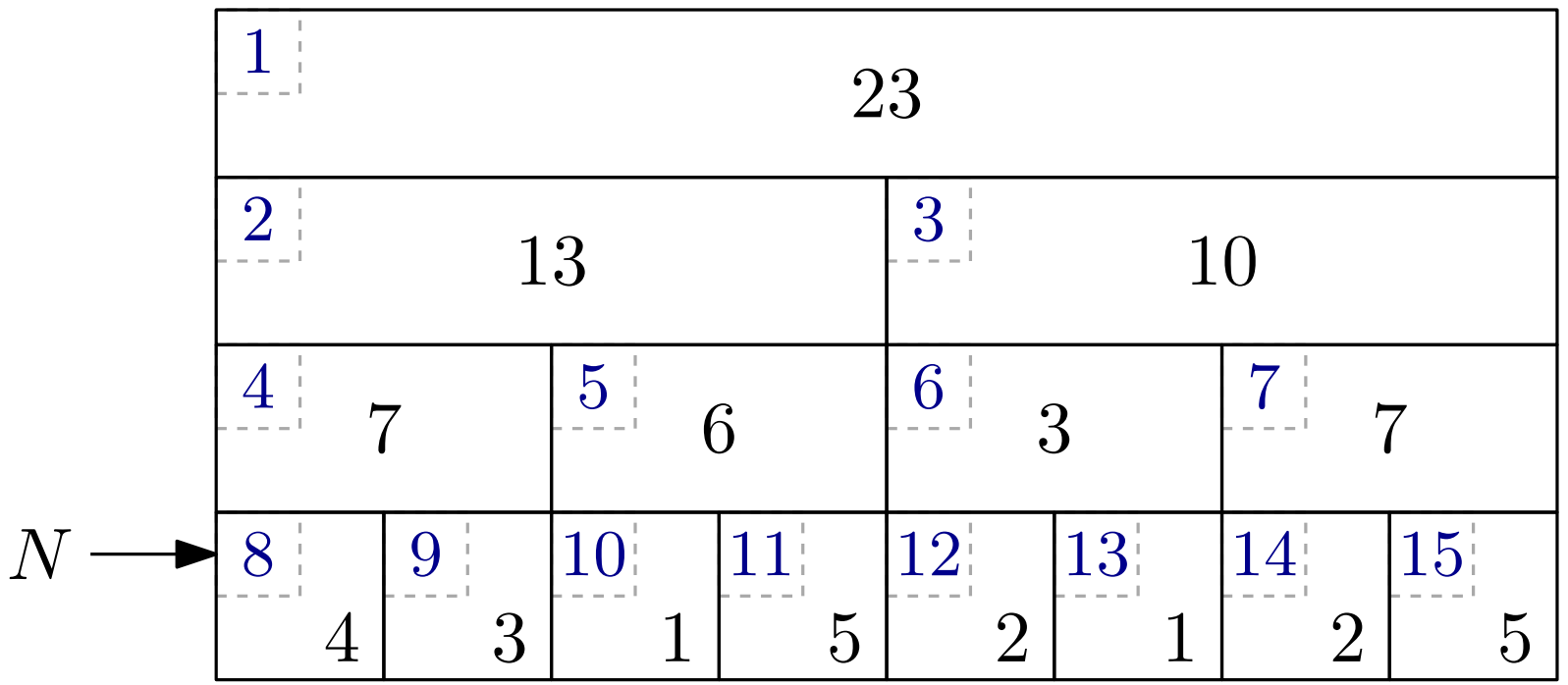
# Alhaalta ylös -toteutus

```
void change(int i, int k) {
        i += N;
        T[i] = k;
        for (i /= 2; i >= 1; i /= 2) {
                T[i] = T[2 * i] + T[2 * i + 1];
        }
}
```



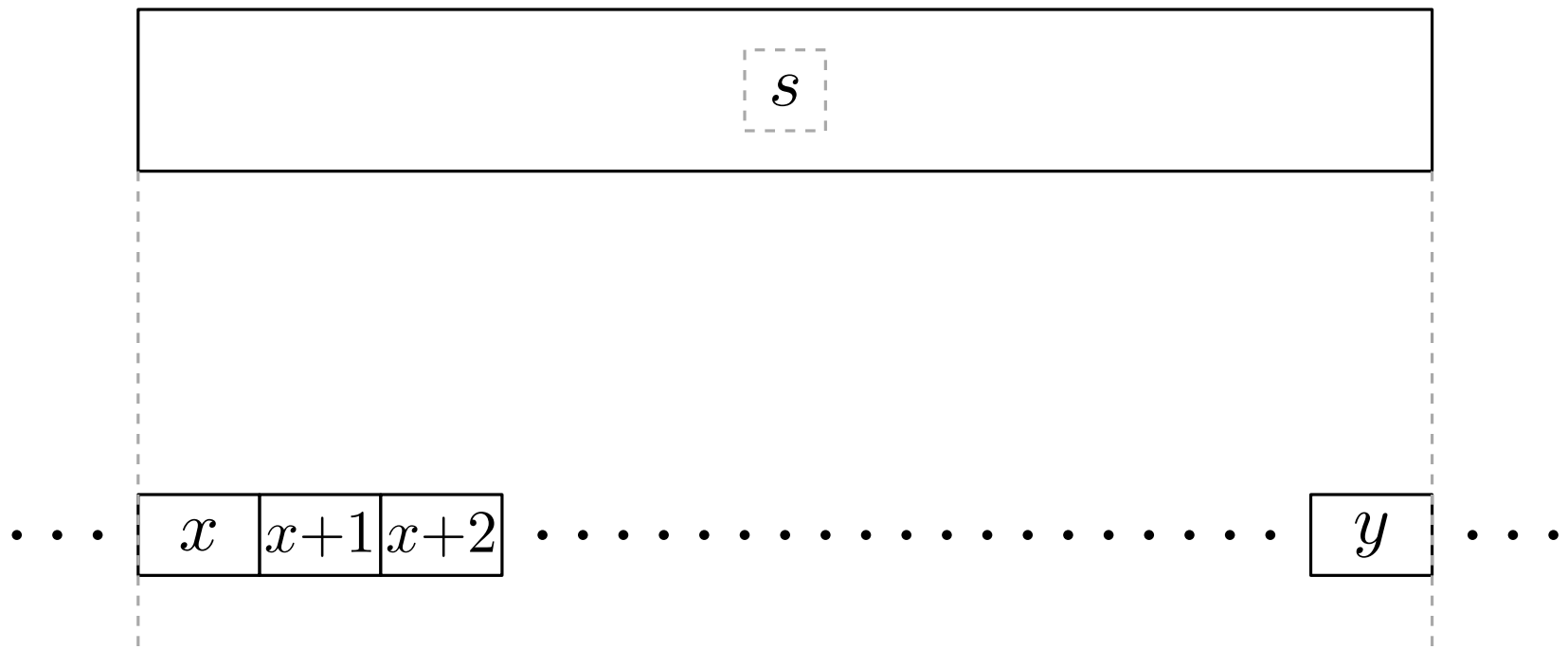
# Alhaalta ylös -toteutus

```
int query(int l, int r) {
        l += N; r += N;
        int s = 0;
        while (l <= r) {
                if (l%2 == 1) s += T[l++];
                if (r%2 == 0) s += T[r--];
                l /= 2; b /= 2;
        }
}
```



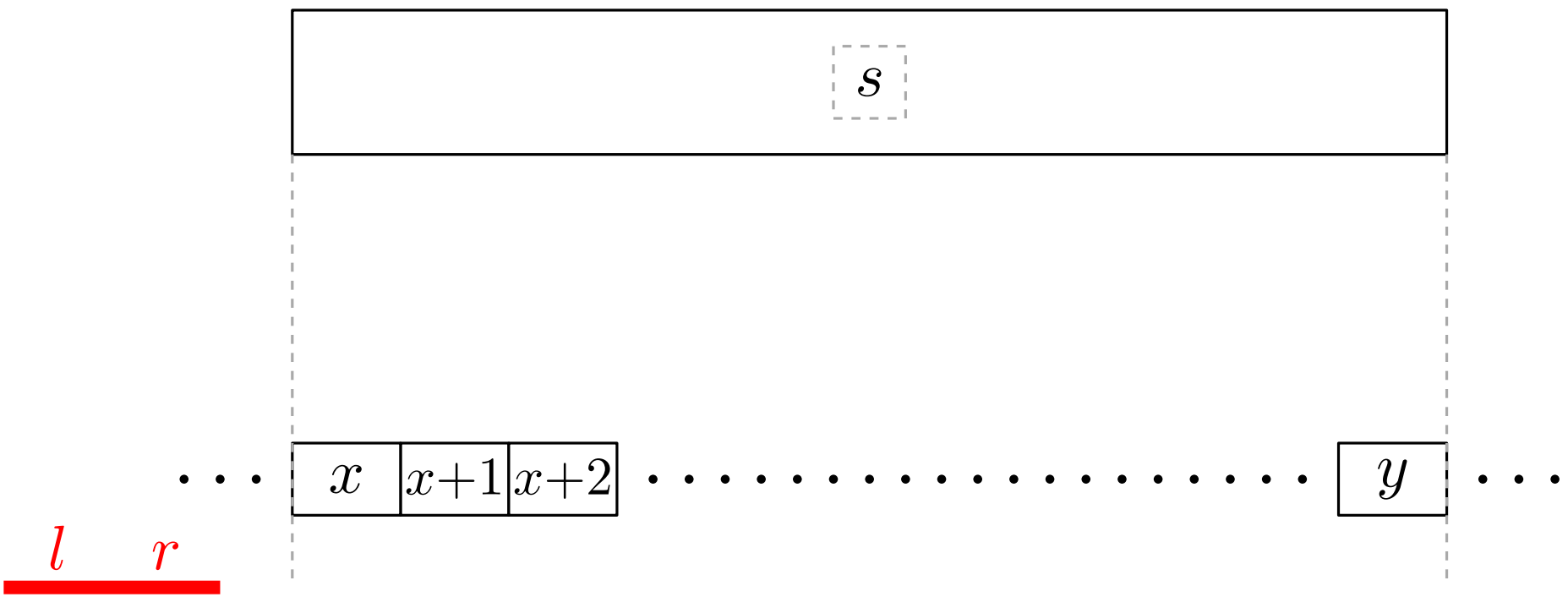
# Ylhäältä alas -toteutus

```
int query(int l, int r, int s=1, int x=0, int y=N-1) {
        if (y < l || r < x) return 0;
        if (l <= x && y <= r) return T[s];
        int m = (x + y) / 2;
        return query(l, r, 2 * s, x, m) +
               query(l, r, 2 * s + 1, m + 1, y);
}
```



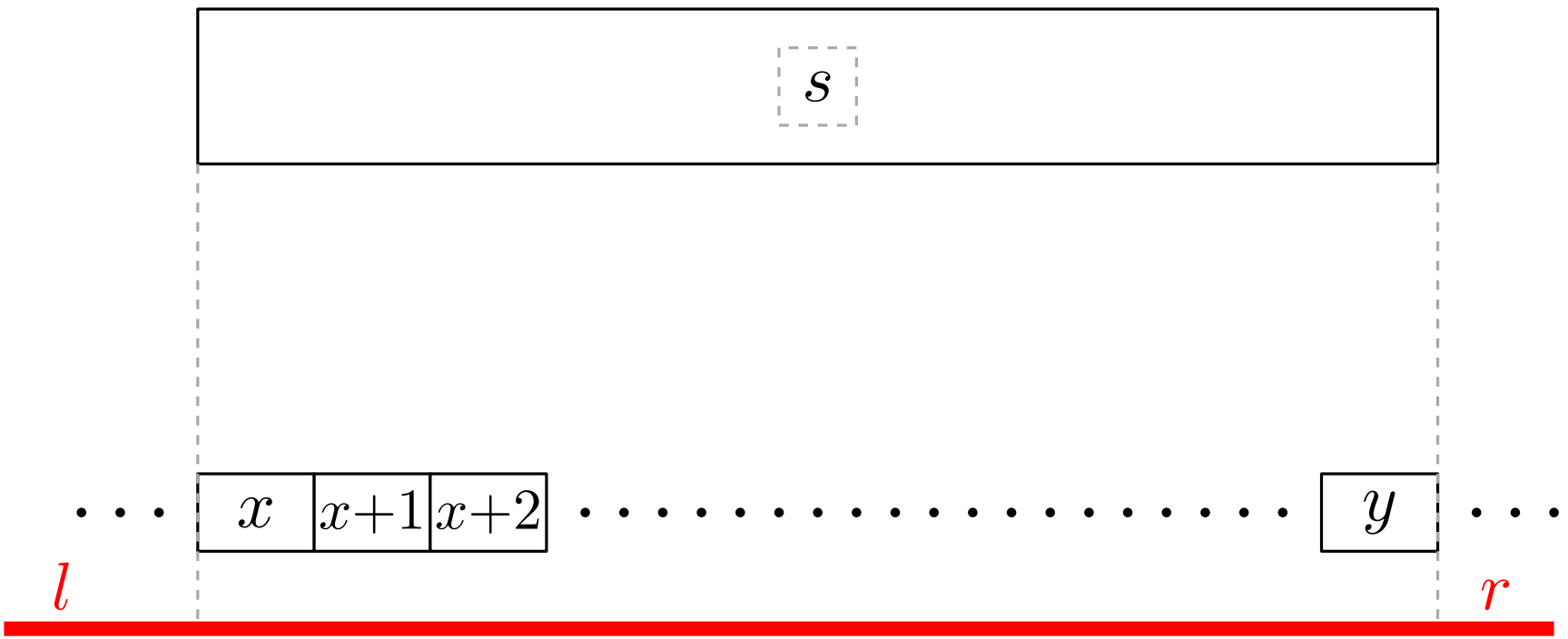
# Ylhäältä alas -toteutus

```
int query(int l, int r, int s=1, int x=0, int y=N-1) {
        if (y < l || r < x) return 0;
        if (l <= x && y <= r) return T[s];
        int m = (x + y) / 2;
        return query(l, r, 2 * s, x, m) +
               query(l, r, 2 * s + 1, m + 1, y);
}
```



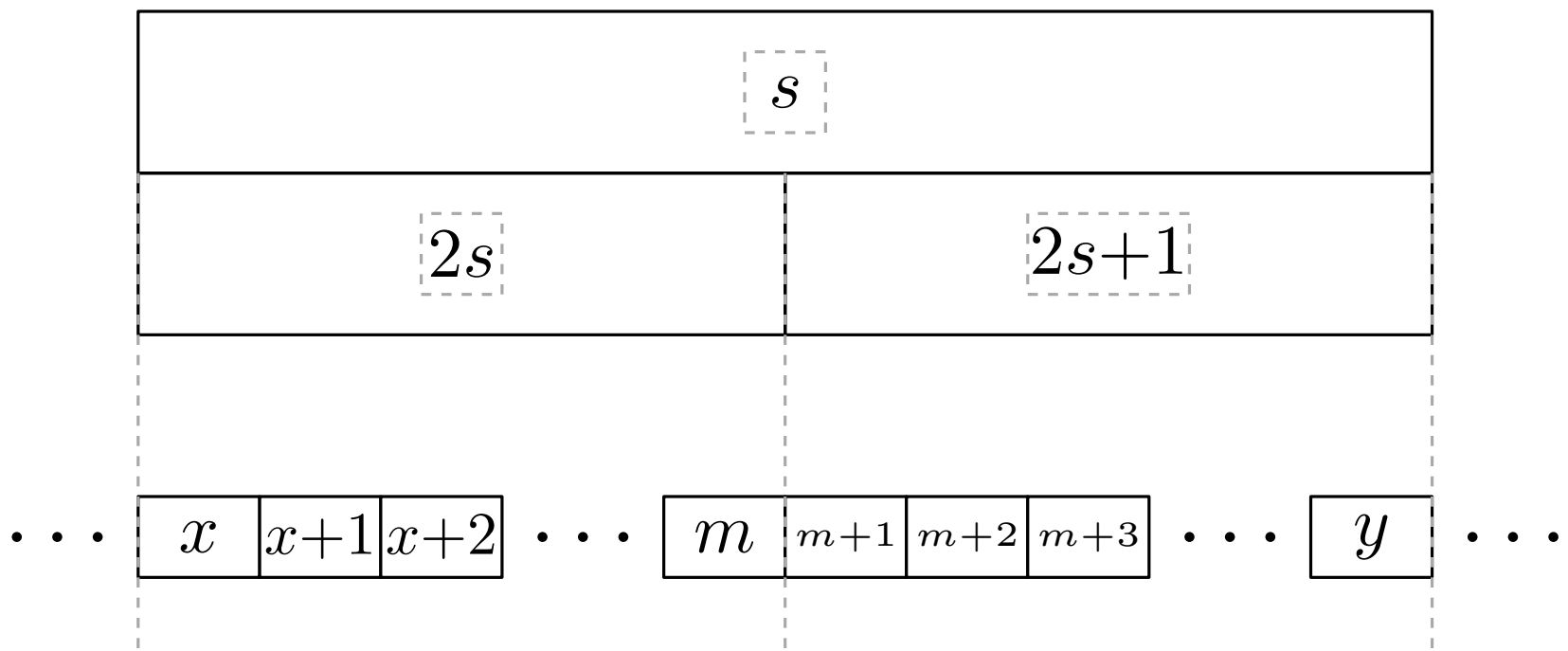
# Ylhäältä alas -toteutus

```
int query(int l, int r, int s=1, int x=0, int y=N-1) {
        if (y < l || r < x) return 0;
        if (l <= x && y <= r) return T[s];
        int m = (x + y) / 2;
        return query(l, r, 2 * s, x, m) +
               query(l, r, 2 * s + 1, m + 1, y);
}
```



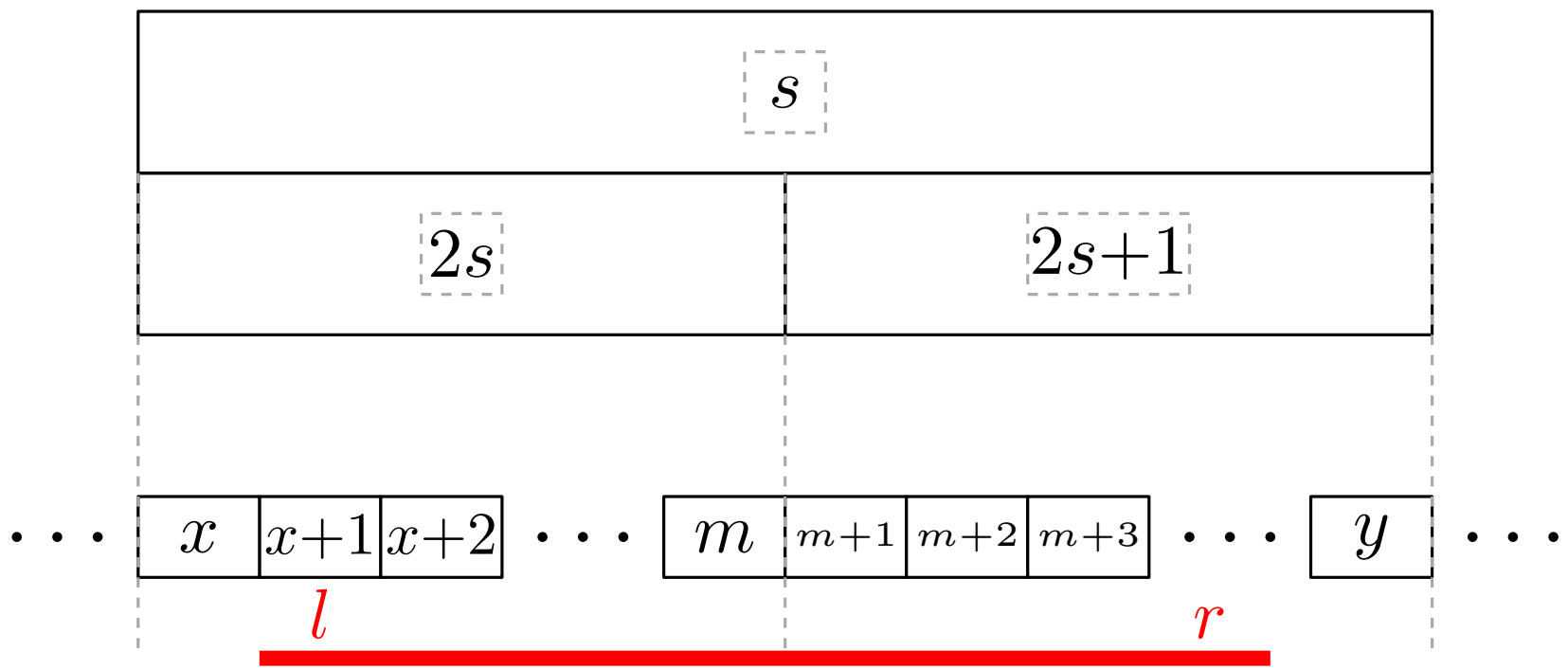
# Ylhäältä alas -toteutus

```
int query(int l, int r, int s=1, int x=0, int y=N-1) {
        if (y < l || r < x) return 0;
        if (l <= x && y <= r) return T[s];
        int m = (x + y) / 2;
        return query(l, r, 2 * s, x, m) +
               query(l, r, 2 * s + 1, m + 1, y);
}
```



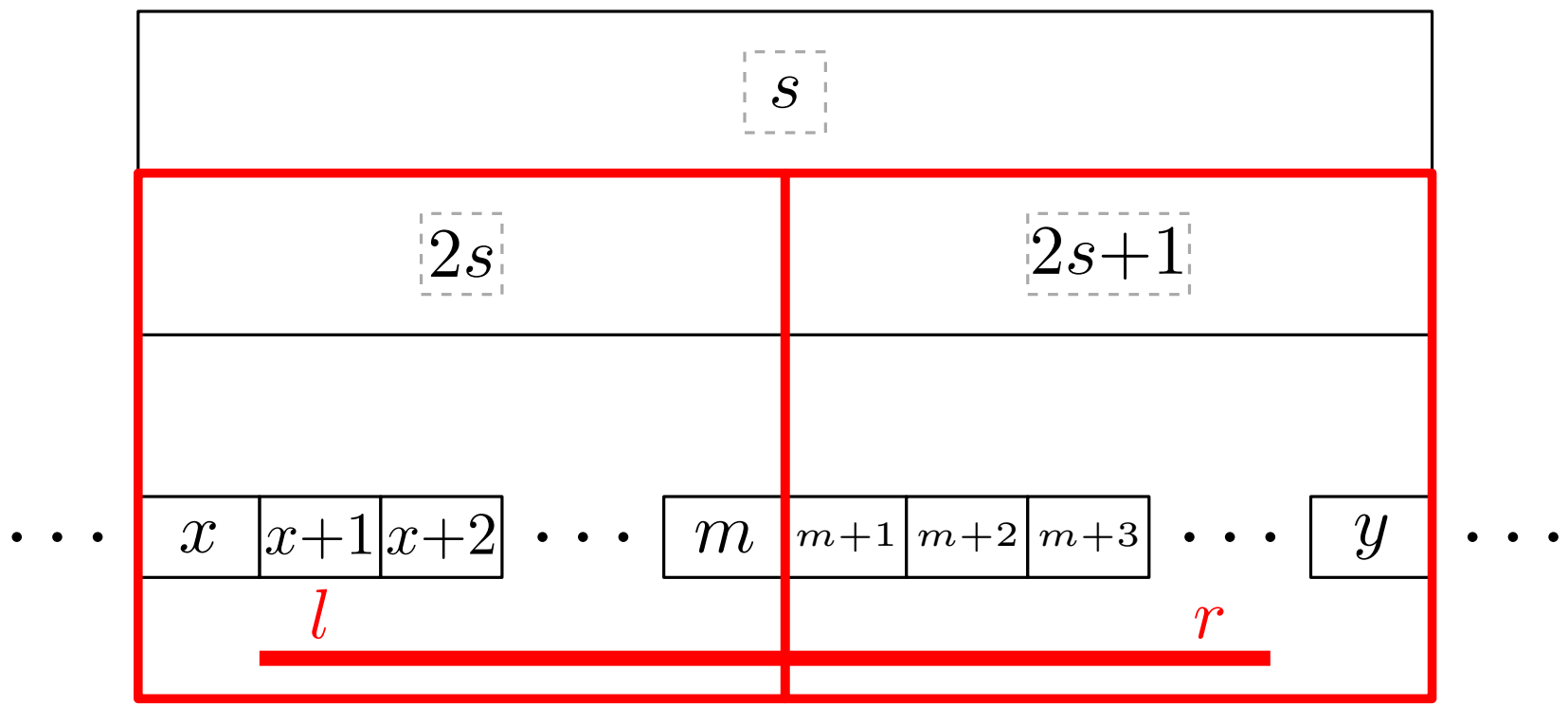
# Ylhäältä alas -toteutus

```
int query(int l, int r, int s=1, int x=0, int y=N-1) {
        if (y < l || r < x) return 0;
        if (l <= x && y <= r) return T[s];
        int m = (x + y) / 2;
        return query(l, r, 2 * s, x, m) +
               query(l, r, 2 * s + 1, m + 1, y);
}
```



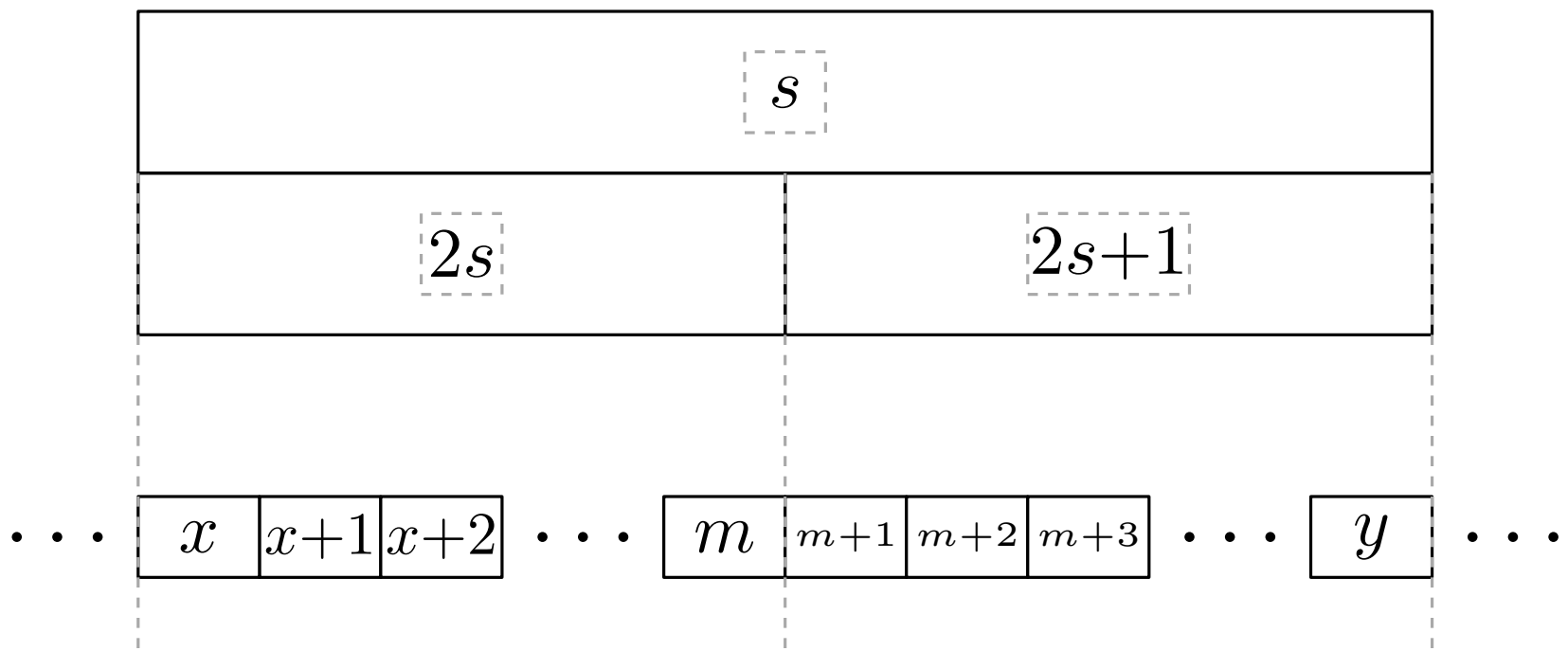
# Ylhäältä alas -toteutus

```
int query(int l, int r, int s=1, int x=0, int y=N-1) {
        if (y < l || r < x) return 0;
        if (l <= x && y <= r) return T[s];
        int m = (x + y) / 2;
        return query(l, r, 2 * s, x, m) +
               query(l, r, 2 * s + 1, m + 1, y);
}
```



# Ylhäältä alas -toteutus

```
int query(int l, int r, int s=1, int x=0, int y=N-1) {
        if (y < l || r < x) return 0;
        if (l <= x && y <= r) return T[s];
        int m = (x + y) / 2;
        return query(l, r, 2 * s, x, m) +
               query(l, r, 2 * s + 1, m + 1, y);
}
```



# Laiska segmenttipuu

- $T[s]$: välin summa, kuten yleensä
- $D[s]$: kaikkiin solmua $s$ vastaaviin alkioihin on lisättävä $D[s]$

```
void push(int s, int len) {
        apply(2 * s, len / 2, D[s]);
        apply(2 * s + 1, len / 2, D[s]);
        D[s] = 0;
}

void apply(int s, int len, int k) {
        T[s] += k * len;
        D[s] += k;
}

void pull(int s) {
        T[s] = T[2 * s] + T[2 * s + 1];
}
```

# Laiska segmenttipuu

- $T[s]$: välin summa, kuten yleensä
- $D[s]$: kaikkiin solmua $s$ vastaaviin alkioihin on lisättävä $D[s]$

```
int query(int l, int r, int s=1, int x=0, int y=N-1) {
        if (y < l || r < x) return 0;
        if (l <= x && y <= r) return T[s];
        push(s, y - x + 1);
        int m = (x + y) / 2;
        return query(l, r, 2 * s, x, m) +
               query(l, r, 2 * s + 1, m + 1, y);
}

void change(int l, int r, int k, int s=1, int x=0, int y=N-1) {
        if (y < l || r < x) return;
        if (l <= x && y <= r) return apply(s, y - x + 1, k);
        push(s, y - x + 1);
        int m = (x + y) / 2;
        change(l, r, k, 2 * s, x, m);
        change(l, r, k, 2 * s + 1, m + 1, y);
        pull(s);
}
```

# Välikyselysegmenttipuu: yleinen operaatio

<table>
<tr><td colspan="8">$a \oplus b \oplus c \oplus d \oplus e \oplus f \oplus g \oplus h$</td></tr>
<tr><td colspan="4">$a \oplus b \oplus c \oplus d$</td><td colspan="4">$e \oplus f \oplus g \oplus h$</td></tr>
<tr><td colspan="2">$a \oplus b$</td><td colspan="2">$c \oplus d$</td><td colspan="2">$e \oplus f$</td><td colspan="2">$g \oplus h$</td></tr>
<tr><td>$a$</td><td>$b$</td><td>$c$</td><td>$d$</td><td>$e$</td><td>$f$</td><td>$g$</td><td>$h$</td></tr>
</table>

# Välikyselysegmenttipuu: pisin musta ketju

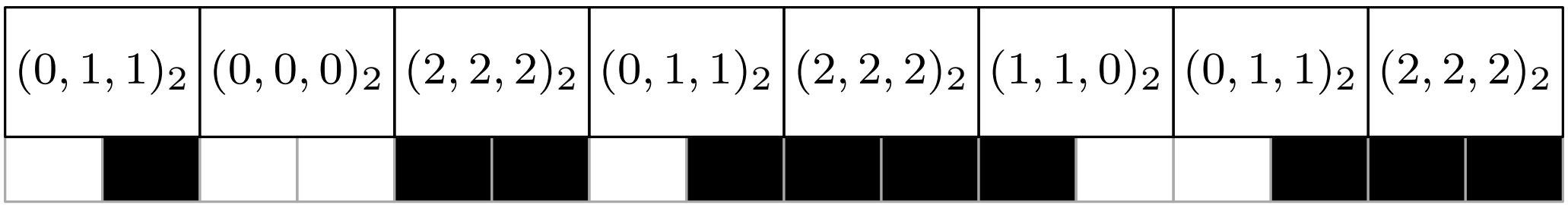
$$(a, b, c)_d$$

$d$ ← välin pituus

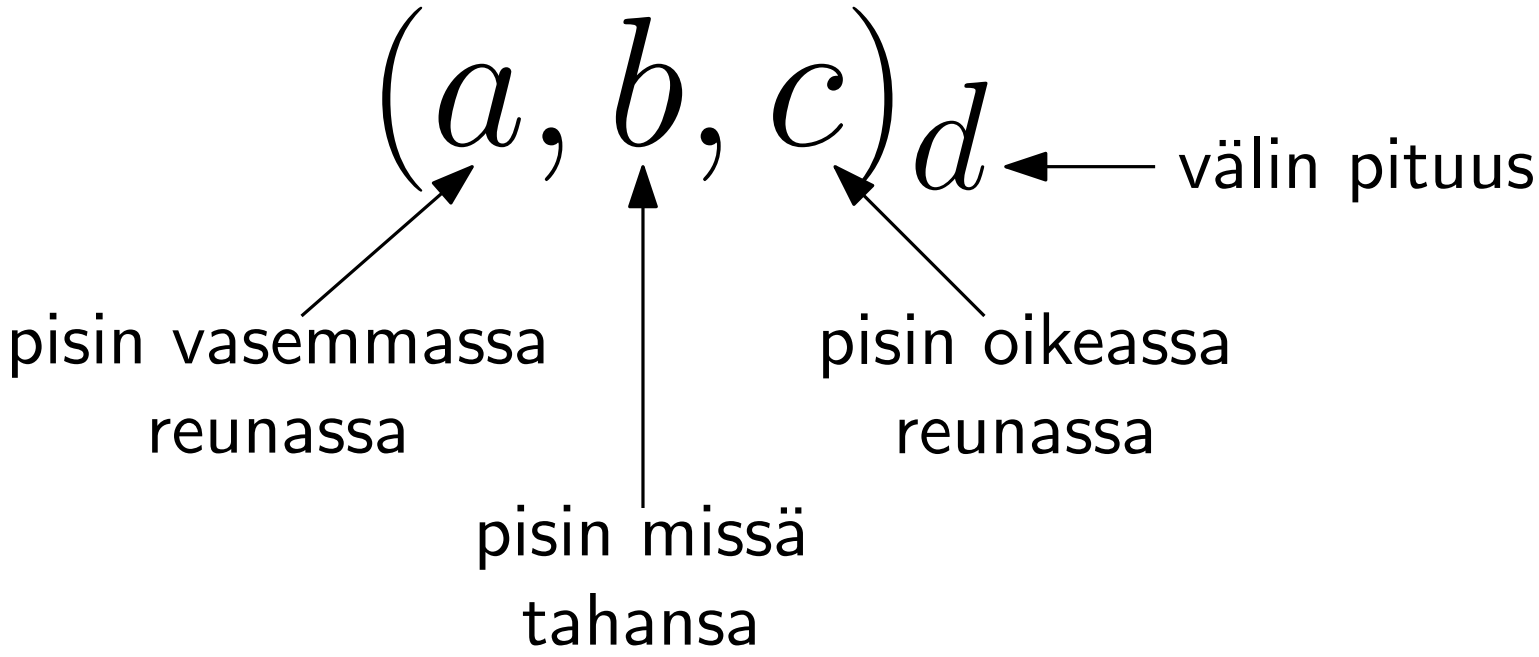
$a$: pisin vasemmassa reunassa

$b$: pisin missä tahansa

$c$: pisin oikeassa reunassa

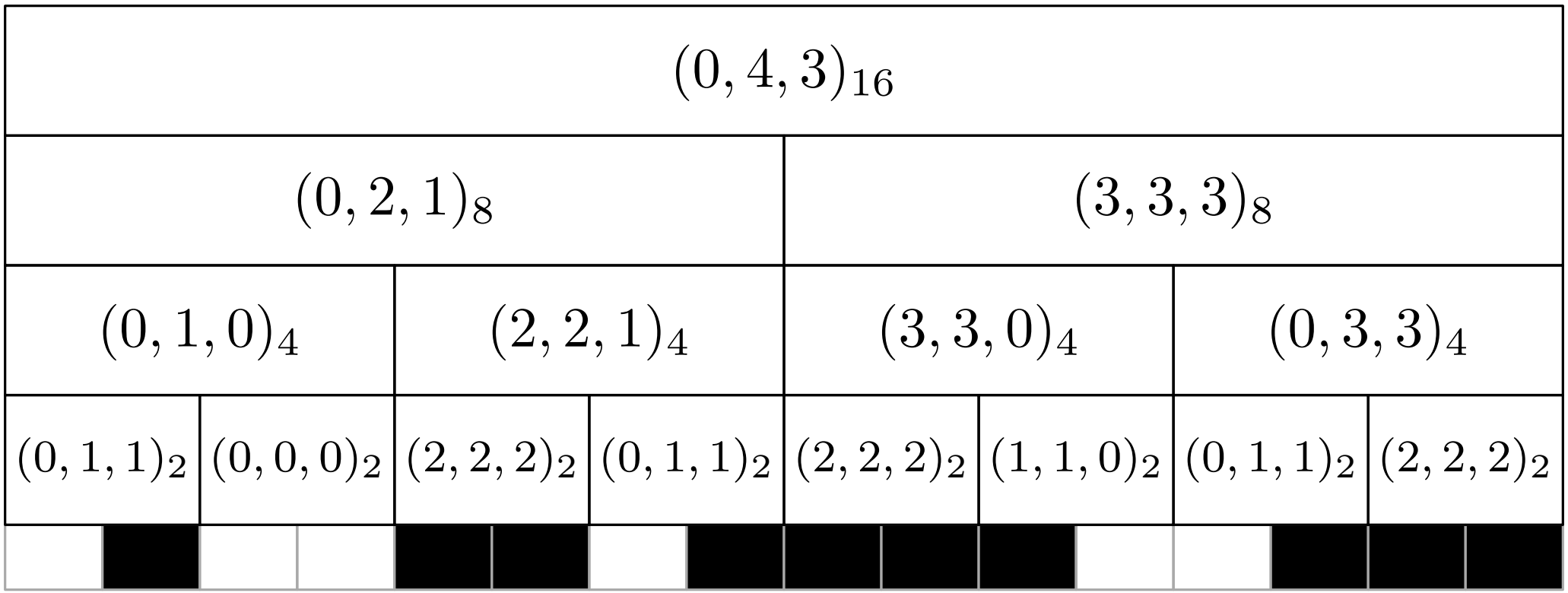


# Välikyselysegmenttipuu: pisin musta ketju




| $(0,1,0)_4$ | | $(2,2,1)_4$ | | $(3,3,0)_4$ | | $(0,3,3)_4$ | |
|---|---|---|---|---|---|---|---|
| $(0,1,1)_2$ | $(0,0,0)_2$ | $(2,2,2)_2$ | $(0,1,1)_2$ | $(2,2,2)_2$ | $(1,1,0)_2$ | $(0,1,1)_2$ | $(2,2,2)_2$ |
| | ■ | | | ■ | ■ | | ■ | ■ | ■ | | ■ | ■ | ■ | ■ |

# Välikyselysegmenttipuu: pisin musta ketju



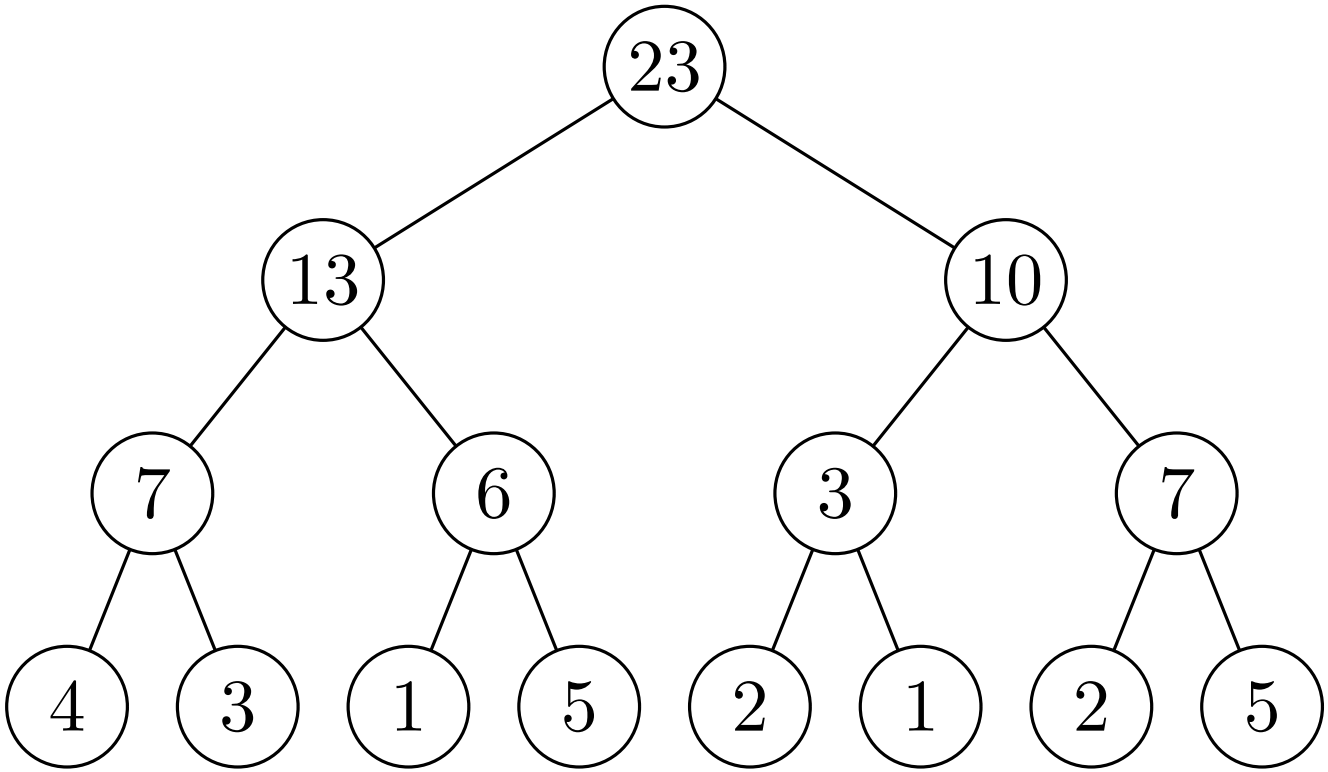
# Harvat segmenttipuut

- Tallennetaan $s$:n vasemman lapsen indeksi `L[s]` ja oikean lapsen indeksi `R[s]`.
- Jos indeksi on 0, solmua ei vielä ole.

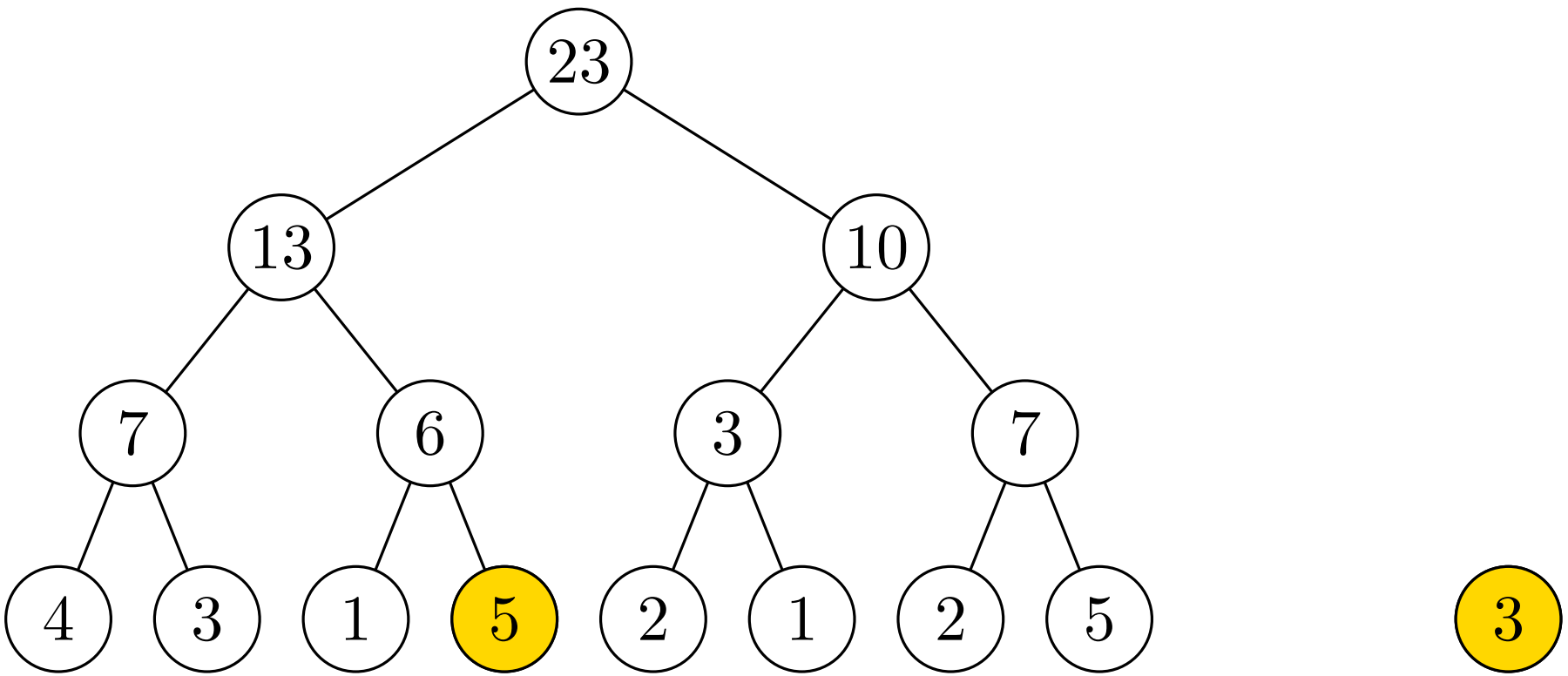
```
int idx = 1;
int left(int s) {
        if(!L[s]) {
                L[s] = idx++;
        }
        return L[s];
}
int right(int s) {
        if(!R[s]) {
                R[s] = idx++;
        }
        return R[s];
}
```

- Korvaa `2*s` $\rightarrow$ `left(s)` ja `2*s+1` $\rightarrow$ `right(s)`

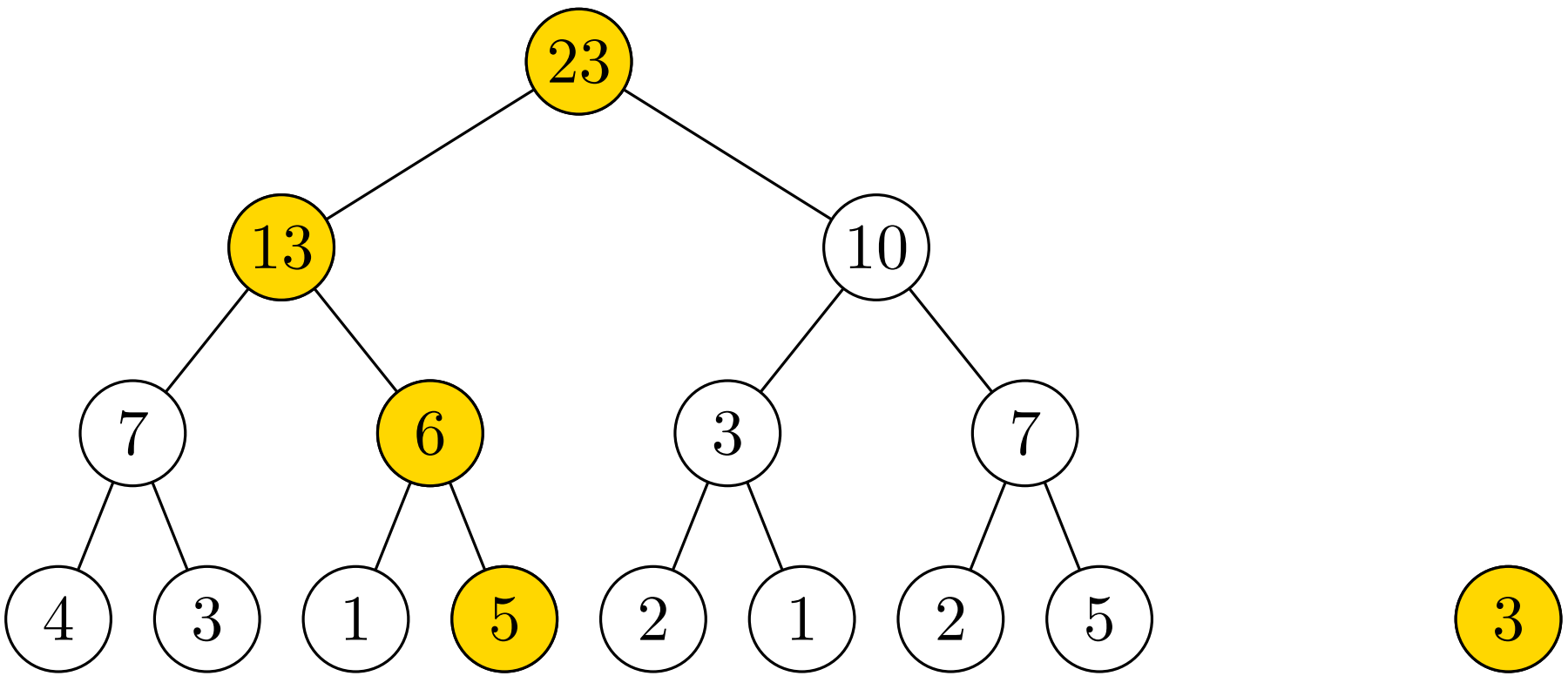
# Persistentit segmenttipuut



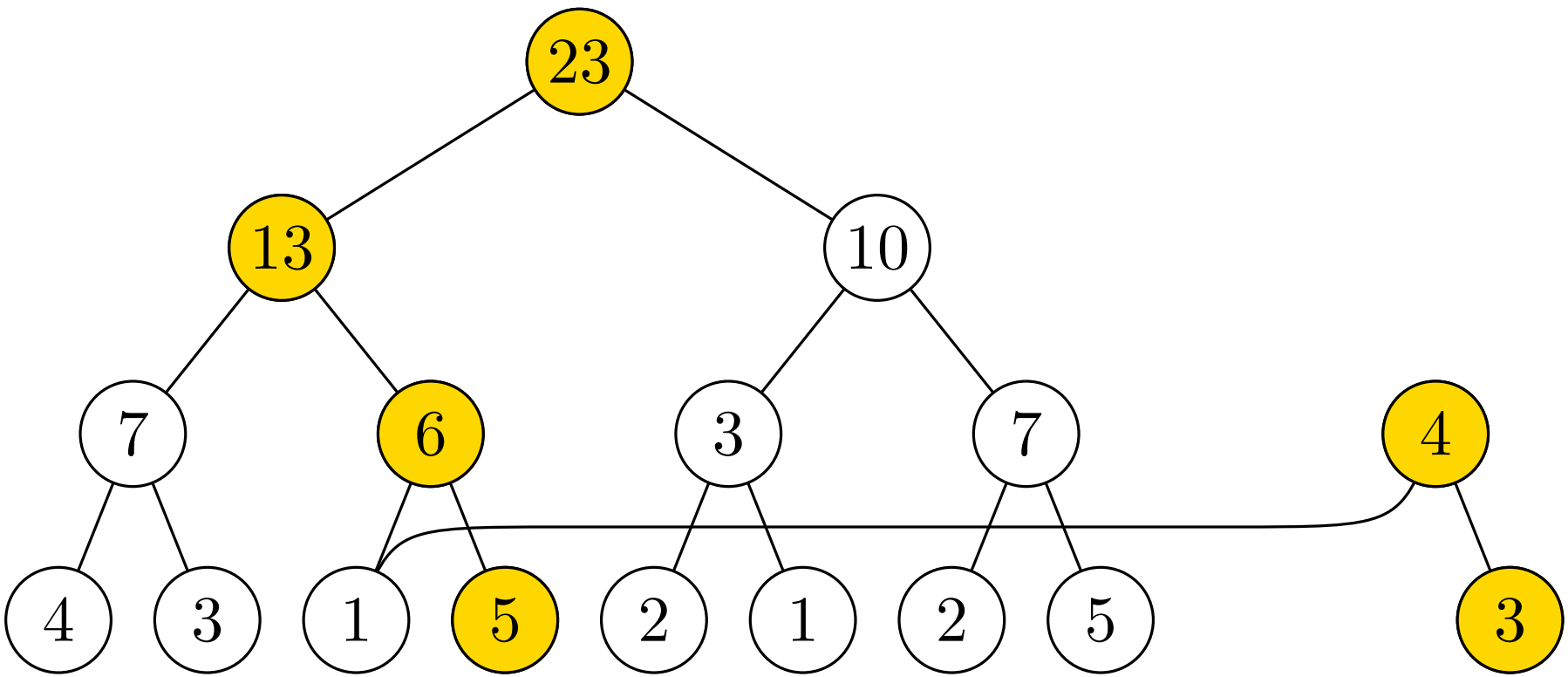
# Persistentit segmenttipuut



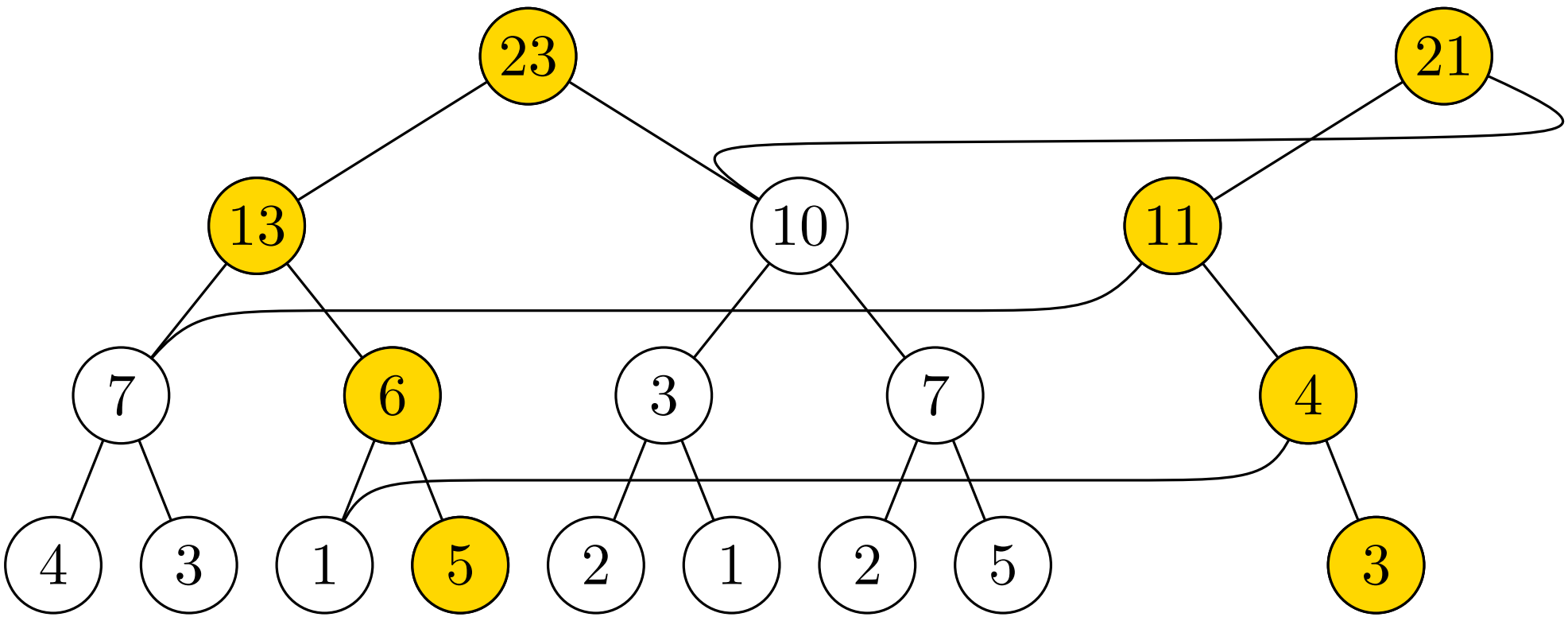
# Persistentit segmenttipuut



# Persistentit segmenttipuut



# Persistentit segmenttipuut



# Persistentit segmenttipuut