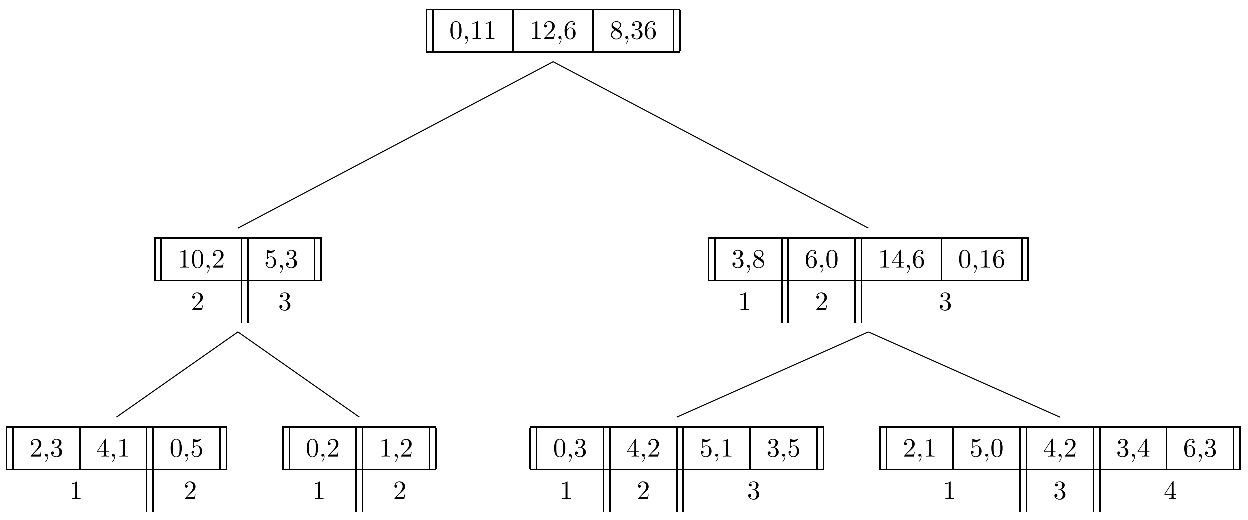
Design


0,11
12,6
8,36
10,2
5,3
2
3
3,8
6,0
14,6
0,16
1
2
3
2,3
4,1
0,5
1
2
0,2
1,2
1
2
0,3
4,2
5,1
3,5
1
2
3
2,1
5,0
4,2
3,4
6,3
1
3
4

Algorithm 1 Main Algorithm

```
Shared Objects
    n: node
        n.parent: parent of n
        n.left: left child of n
        n.right: right child of n
        n.last: index of the last block in the array
        n.blocks: array for block objects of n
            block.enq: integer tuple(left, right)
            block.deq: integer tuple(left, right)

function Do(operation op)
    l= p's assigned leaf in tree
    l.append(op)
    Propagate(l.parent)
    return Compute(op)
end function

function Propagate(node n)                              ▷ propagates $n$ up to the root
    block ← Read & Merge(n)
    if not CAS(n.last, n.last, n.last+1) then
        CAS(n.last, n.last, n.last+1)
    end if
    blocks[last] ← Read & Merge(n)
    if n==root then return
    else Propagate(n.parent)
    end if
end function

function Search(node n, type t, index i , optional:{left, right})
                        ▷ returns #block containing $op_i$ of type $t$ in node $n$
end function

function Prefix-Sum(node n, type t, index i , optional:{left, right})
                        ▷ returns #ops before $op_i$ of type $t$ in node $n$
end function
```

Algorithm 2 Main Algorithm Continued

```
 1: function READ & MERGE(node n)
 2:     new-block ← {}
 3:     for each type ∈ {enq, deq} do
 4:         new-block.type ← (n.left.new(type), n.right.new(type))
 5:     end for
 6:     overall-left ← (n.left.done(enq) - n.left.done(deq) + n.left.done(nill-deq)) + n.left.new(enq) - n.left.new(deq)
        + n.left.new(nill-deq)
 7:     overall-right ← (n.left.done(enq) - n.left.done(deq) + n.left.done(nill-deq)) + n.left.new(enq) -
        n.left.new(deq) + n.left.new(nill-deq)
 8:     return tuple(−overall-left, −overall-right)
 9: end function

10: function GET(node n, index i, type∈{enq, deq})          ▷ returns $op_i$ in the subtree of node $n$
11:     position ← SEARCH(n, type, i)
12:     #before-position ← $\sum_{j=0}^{position-1} n.blocks[j].type.left + n.blocks[j].type.right$
13:     direction ← (#before-position + n.blocks[position].type.left ≥ $i$) ? left : right
                                ▷ calculate block position of $i$ and direction of the child
14:     if direction=left then
15:         $\#older_{right} \leftarrow \sum_{j=0}^{position-1} n.blocks[j].type.right$
16:         GET(n.left, $i - \#older_{right}$)
17:     else
18:         $\#older_{left} \leftarrow \sum_{j=0}^{position} n.blocks[j].type.left$
19:         GET(n.right, $i - \#older_{left}$)
20:     end if
21: end function

22: function ORDER(node n, index i, given-type ∈ {enq, deq, nill-deq})
                                                          ▷ let $b$ be the $i$th block in $n$,
        ▷ returns how many operations of the given type are before $b$'s last operation in the whole ordring
23:     if n==root then return PREFIX-SUM(n, given-type, i)
24:     else if type ∈ {enq, deq} then
25:         direction ← (n.parent.left==n) ? left : right
26:         #self-ops ← PREFIX-SUM(n, given-type, i)
27:         parent-position ← SEARCH(n.parent, given-type, i, direction)
28:         ORDER(n.parent, parent-position, given-type)
29:     else                                              ▷ TODO: nill-deq case
30:     end if
31: end function
```

Algorithm 3 Main Algorithm Continued

```
function COMPUTE(operation op)                          ▷ returns result of operation op
    l= op's assigned leaf in tree
    offset= op's block index in the l      ▷ TODO:handle other cases(not complete block)
    if op.type==ENQ then return
    else
        enqs ← ORDER(l, offset, {ENQ})
        deqs ← ORDER(l, offset, {DEQ})
        nill-deqs ← ORDER(l, offset, {Nill-DEQ})
        return (enqs-deqs+nill-deqs) > 0 ? GET(root, enqs-deqs+nill-deqs) : null
    end if
end function
```