

Задание 1

Отсортировать n элементов не больше $\log n$ за $O(n)$ work и $O(\text{polylog } n)$ span (без конкурентных операций).

```
1 func Sort(n int, A []int) []int {
2     histograms := make([][]int, n/log(n))
3
4     blockLen := log(n)
5     blocks := ceil(n / blockLen)
6
7     // делаем линейный подсчет на блоках длины len(n)
8     ParallelFor(blocks, func(idx int) {
9         blockStart := idx * blockLen
10        blockEnd := min((idx+1)*blockLen, n)
11
12        histograms[idx] = calcHistogram(blockLen, A[blockStart:blockEnd])
13    })
14
15    // собираем все посчеты в единую гистограмму
16    Reduce(blocks, func(block1 int, block2 int) {
17        for i := range blockLen {
18            histograms[block1][i] += histograms[block2][i]
19        }
20    })
21
22    histogram := histograms[0]
23
24    // индексы блоков с одинаковыми элементами
25    postitions := Scan(histogram)
26
27    result := make([]int, n)
28
29    ParallelFor(blocks, func(element int) {
30        start := postitions[element]
31        end := postitions[element+1]
32
33        ParallelFor(end-start, func(idx int) {
34            result[idx] = element
35        })
36    })
37
38    return result
39 }
40
41 func calcHistogram(length int, A []int) []int {
42     histogram := make([]int, length)
43     for _, el := range A {
44         histogram[el] += 1
45     }
46     return histogram
47 }
```

Задание 2

Дано подвешенное дерево за вершину r с уже построенным Эйлеровым обходом. Нужно для каждой вершины найти её глубину в дереве за $O(n)$ work и $O(\text{polylog } n)$ span.

```
1 type Edge struct {
2     From, To int
3 }
4
5 func Depth(n int, r int, traverse []Edge) []int {
6     // получаем родителей для каждой вершины разрывом цикла после последнего
    // вхождения r и подсчетом List ranking
7     parents := GetParents(n, r, traverse)
8
9     weights := make([]int, len(traverse))
10    ParallelFor(len(traverse), func(idx int) {
11        edge := traverse[idx]
12        weight := -1 // для ребер сын->родитель
13        if edge.From == parents[edge.To] {
14            weight = 1 // для ребер родитель->сын
15        }
16        weights[idx] = weight
17    })
18
19    // получаем глубину вершин в которые приходим на каждом шаге обхода
20    depths := Scan(weights)
21
22    result := make([]int, n)
23    ParallelFor(len(traverse), func(idx int) {
24        edge := traverse[idx]
25        if edge.From == parents[edge.To] {
26            result[edge.To] = depths[idx]
27        }
28    })
29
30    return result
31 }
```