CS 6230 Midterm Exam - Han Tran
(I choose the Spring 2017 exam)

Q1
Each process determines the occurrences of y in n/p elements of **X**. Then the total number of occurrences of y in **X** is obtained by taking a reduction (with sum operation). The complexity is $\mathcal{O}(n/p + \log p)$. Using PRAM model with p = n, we get the complexity of $\mathcal{O}(\log n)$.

Pseudo-code:
```
count = 0;
for i = 0; i < n/p; i++
    if x[i] == y
        count += 1;
    }
}
reduction (count, 1, count_total, sum);
```

Q2
Vertices to be merged are the two being adjacent to each other and having a maximal-weight common edge. This method is called Heavy Edge Matching (HEM) which aims to minimize the cut. We try to get as many pairs as possible. The weight of an edge in the coarser graph will be the sum of weights of the edges (of the initial graph) connecting to the merged vertices.

Pseudo-code:
```
for each vertex v_i
    if v_i is not paired
        search adjacent vertex v_j that is not paired and the shared edge has maximal weight;
        v_i and v_j are paired;
    }
}
```

The complexity of this algorithm is $\mathcal{O}(n/p)$. The weak point of this algorithm is that vertex v_j could be assessed concurrently by different threads. This results in a wrong merging of vertex v_j. To fix this problem, we could have a way to control the timing of pairing the vertices.

Q3
- First, each process determines the occurrences of each label l_i in n/p elements of **L**. This data is saved in counts[i] (the size of counts[] will be k). This search has complexity of $\mathcal{O}(n/p)$. Then the total occurrences of label l_i is obtained by taking a reduction (of k elements) and saved in counts_total[i]. This reduction has complexity of $\mathcal{O}(k \log p)$.
- Next, an exclusive scan is carried out for counts_total[] and the data is saved in offsect[]. This scan has complexity of $\mathcal{O}(\log k)$.

- Finally, the array A is re-ordered based on offsect[]. This reordering has complexity of $\mathcal{O}(n/p)$. Because k is constant, the total complexity of the algorithm is $\mathcal{O}(n/p + k\log p + 2\log k + n/p) = \mathcal{O}(\log n)$ when $p = n$.

<u>Pseudo-code</u>:

```
counts[] = 0; /* size of count[] is k */
for i = 0; i < n/p; i++
    if x[i] == y
        counts [L[i]-1] += 1;
    }
}
reduction all (counts, k, counts_total, sum);
offset = exclusive scan (counts_total);
counts[] = 0;
for i = 0; i < n/p; i++
    Aout [offset [L[i] - 1]] + counts [L[i] - 1] = A [i];
    counts [L[i] - 1] += 1;
}
```

Q4

Evaluation of the polynomial $p_n(x_0)$ could be considered as a sequence of n elements $\{s_1, s_2, \ldots, s_n\}$ where $s_1 = a_0x_0 + a_1$, $s_2 = s_1x_0 + a_2$, …, $s_n = s_{n-1}x_0 + a_n$. Then $p_n(x_0) = s_n$.

The sequence is partitioned into p sub-sequences. Process 0 computes the first n/p elements, process 1 computes the next n/p elements, etc. The last element will be multiplied by $x_0^{f(r)}$ where f(r) is the function of the rank r. The last term s_n (i.e. the value of $p_n(x_0)$) is obtained by having a reduction (with sum). The reduction has complexity of $\mathcal{O}(\log p)$. Thus, the total complexity is $\mathcal{O}(n/p + \log p)$.
I believe it is a typo in the question when writing $\mathcal{O}(n/p + \log n)$. We can obviously see that this is incorrect when we take p = 1 (sequential algorithm).

<u>Pseudo-code:</u>

```
if r is not the root then a [0] = 0;
s = a [0];
for i =0; i < n/p; i++
    s = s*x0 + a [r*(n/p) + i + 1];
}
s = s*(x0) ^(n-(r+1)*(n/p));
reduction (s, 1, pn, sum);
```