1. (a) We consider sequences consisting of opening and closing parentheses. A sequence is *well-formed*, if it corresponds to a sequence of parentheses in some well-formed arithmetic expression: for example, the sequences '(((())))', '()(())()((()))', '' (empty sequence), are well-formed, while the sequences ')(', '(()))', '()(()', are not. Design an efficient BSP algorithm that, given an array of characters containing a sequence of parentheses, determines whether it is well-formed. Describe the main idea for any BSP algorithm used as a subroutine; you do not need to describe any such algorithms in detail. [12]

Solution: Denote the sequence of parentheses by $x[i]$, where $0 \leq i < n$. Define an integer array y of size n as

$$y[i] \leftarrow \begin{cases} 1 & x[i] = \text{'('}, \\ -1 & x[i] = \text{')'}. \end{cases}$$

Let z be an array of prefix sums of y:

$$z[i] \leftarrow \sum_{0 \leq j \leq i} y[j]$$

Intuitively, $z[i]$ gives the nesting depth at position immediately following parenthesis $x[i]$. Sequence x is well-formed, if and only if $z[i] \geq 0$ for all i, $0 \leq i < n$, and $z[n-1] = 0$.

The algorithm proceeds as follows. Each processor reads a contiguous block of n/p elements of sequence x, and initialises a corresponding block of array y. The processors then perform the BSP prefix sums algorithm on array y, using the addition operator; as a result, each processor obtains a contiguous block of the prefix sums array z. Finally, each processor checks the conditions $z[i] \geq 0$ for all i, $0 \leq i < n$, and $z[n-1] = 0$ on its local block of array z, and notifies a designated processor if at least one of the conditions is violated. The designated processor returns "false (not well-formed)" if it has received such a notification from any of the processors, otherwise is returns "true (well-formed)".

(b) For the algorithm developed in part (b), give its asymptotic BSP cost, justifying your answers, and stating all necessary assumptions. [5]

Solution: (Application) The input and output run in $O(n/p)$ local computation and communication, and one superstep. The prefix sums subroutine runs in $O(n/p)$ local computation and communication, and $O(1)$ supersteps. The final checks run in $O(n/p)$ local computation and $O(p)$ communication. Therefore, the whole algorithm has local computation cost $O(n/p)$, communication cost $O(n/p)$ and synchronisation cost $O(1)$. For the prefix sums subroutine, we need $n/p \geq p$, hence $n \geq p^2$.