Name:_____**Shuqing Ye**_____**UCINETID:**_____**shuqiny2**_____

Test cases (including the edge cases):

Input: “” // empty string Expected out: [0,0,0,0]
Input: ([[( ))]] Expected out:[1,1,0,0]
Input: (hello [my [(name is (ray)]]] Expected out: [0, 2, 1, 0]
Input: (hello [my)name is] Expected out: [1,1,0,0]
Input: (hello(my[name) Expected out: [0,1,0,1]

```
public int[] checkParens (String s) {
    Int[] res = new int[4];
    If (s.length() == 0 )
        return res;

    // stack only stores ‘(‘ and ‘[‘
    Stack<Character> stack = new Stack();

   // queue stores unmatched ‘)’ and ‘]’
    Queue<Character> queue = new LinkedList<>();

    for (char c : s.toCharArray()) {
        If (c == ‘(’ || c == ‘[’)
            Stack.push(c);
        else if (c == ‘)’ || c == ‘]’) {
            If (isMatching(stack.peek(), c))
                Stack.pop();
            else queue.offer(c);
        }
        else
              continue;    // ignore other characters
    }

    matchStackAndQueue(stack, queue, res);

    return res;
}
```

// T(n) = O(n) because we traverse the string once, which is O(n), then we empty the stack and queue, which is also O(n).

```
// check if parentheses matches
private boolean isMatching (char a, char b) {
    return ( a == ‘(’ && b == ‘)’ ) ||
                ( a == ‘[’ && b == ‘]’ );
}

// match the top element of stack and pop
private void matchStackTop
                    (Stack<Character>stack, int[] res) {
          char c = stack.pop();
          c == ‘(’ ? res[1] ++ : res[3] ++;
}

// match the front element of queue and remove
private void matchQueueFront
                    (Queue<Character>queue, int[] res) {
          char c = queue.poll();
          c == ‘)’ ? res[0] ++ : res[2] ++;
}

// recursively match the top of stack and the front of
queue until both are empty
private void matchStackAndQueue
(Stack<Character>stack ,
Queue<Character>queue, int[] res){
    if (stack.isEmpty() && queue.isEmpty())
          return;
    if (isMatching(stack.peek(), queue.peek())) {
          stack.pop();
          queue.poll();
     }
       else {     // peeks don’t match
            if (stack.size() > queue.size())
               matchStackTop (stack, res);
            else
              matchQueueFront(queue, res);
       }
 return matchStackAndQueue(stack, queue, res);
}
```