原题内容：

20. Valid Parentheses

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

- Open brackets must be closed by the same type of brackets.
- Open brackets must be closed in the correct order.

Note that an empty string is also considered valid.

Example 1:

```
Input: "()"
Output: true
```

Example 2:

```
Input: "()[]{}"
Output: true
```

Example 3:

```
Input: "(]"
Output: false
```

Example 4:

```
Input: "([)]"
Output: false
```

Example 5:

```
Input: "{[]}"
Output: true
```

第一遍解法代码：

```
//时间复杂度O(n) 4ms beats 100%
bool isValid(string s) {

    if (s == "")
        return true;
    stack<char>s1, s2;

    for (auto c : s)
        s1.push(c);

    while (!s1.empty()){
        if (!s2.empty()){
            char ch = s2.top();
            if (ch == '(' || ch == '{' || ch == '[')
                return false;
            else if (ch == ')'){
                if (s1.top() == '('){
                s2.pop();
                }
                else{
                s2.push(s1.top());
                }
            }
```

```
            else if (ch == '}'){
                if (s1.top() == '{'){
                s2.pop();
                }
                else{
                s2.push(s1.top());
                }
            }
            else if (ch == ']'){
                if (s1.top() == '['){
                s2.pop();
                }
                else{
                s2.push(s1.top());
                }
            }
        }
        else
            s2.push(s1.top());
        s1.pop();
    }
    if (s2.empty())
        return true;

    return false;
}
```

网上好的解法：

```
//时间复杂度O(n) 4ms beats 100%
        bool isValid(string s) {
            map<char, char> parenth_dict;
            parenth_dict['('] = ')';
            parenth_dict['{'] = '}';
            parenth_dict['['] = ']';

            stack<char> aux;
            for (int i = 0; i < s.size(); i++) {
                if (s[i] == '(' || s[i] == '{' || s[i] == '[')
                    aux.push(s[i]);
                else if (aux.empty() || parenth_dict[aux.top()] != s[i])
                    return false;
                else
                    aux.pop();
            }

            return aux.empty();

        }
//时间复杂度O(n) 4ms beats 100%
    bool isValid(string s) {
        stack<char> st;
        for(auto c : s){
            if(c == '(' || c == '{' || c == '[') st.push(c);
            else if(st.size() > 0){
                if(c == ')' && st.top() == '(') st.pop();
                else if (c == '}' && st.top() == '{') st.pop();
                else if (c == ']' && st.top() == '[') st.pop();
                else return false;
            } else return false;
        }
        return st.size() == 0;
    }
```

精简优化：

```
//时间复杂度O(n) 4ms beats 100%
bool isValid(string s){
    stack<char> s1;

    if (s.empty())
        return true;
    else if (s.length() % 2 != 0)
```

```
        return false;

    for (auto it : s){
        if (it == '(' || it == '{' || it == '[')
            s1.push(it);
        else if (it == ')' && !s1.empty() && s1.top() == '(')
            s1.pop();
        else if (it == ']' && !s1.empty() && s1.top() == '[')
            s1.pop();
        else if (it == '}' && !s1.empty() && s1.top() == '{')
            s1.pop();
        else
            return false;
    }

    return s1.empty();
}
```

思考：

1. 解法思路方面：

   - 第一遍代码采用了两个stack模拟括号匹配的过程。
   - 关于有效括号表达式的一个有趣的属性是：有效表达式的子表达式也应该是有效表达式。抓住这个特性，我们便可以只是用一个stack即可进行括号的匹配。在遍历s的过程中，遇到右括号则进栈至s1中，如果遇到左括号，则只需要判断s1.top()是否是相应的右括号即可。

2. 代码解决方面：

   - 判断一个整数是否是奇数的时候，用x % 2 != 0，不要用x % 2 == 1，因为x可能是负数。
   - 在判断两个浮点数a和b是否相等时，不要用 a == b，应该判断两者之差的绝对值fabs(a-b)是否小于某个阈值，例如 1e-9
   - 在需要使用s1.top()之前，一定要先判断下s1是否为空，依次类推在要使用容器的元素的时候，一定先思考下该容器现在的状态是否为空。