Стек, очередь, квадратичные сортировки

1 A. A + B

Разбор

Это тестовая задача, предложенная для знакомства с тестирующей системой codeforces.

Авторское решение

```
a, b = map(int, input().split())
print(a + b)
```

2 B. Реализация стека

Разбор

В этой задаче было необходимо реализовать стек при помощи структуры vector или её аналога в используемом вами языке (list для Python, List<T> для C#, ArrayList<T> для Java и т.п.).

Особое внимание следовало обратить на формат ответа вашей программы на каждую из команд:

- Команда **push** требовала вывод **ok**
- Команда **clear** требовала вывод **ok**
- Команда **pop** требовала вывод удаляемого элемента или сообщения 'error', если стек пуст
- Команда **back** требовала вывод сообщения 'error', если стек пуст

Авторское решение

```
x = []
c = input()
while c != 'exit':
    cmd = c.split()
    if cmd[0] == 'push':
        x = x + [cmd[-1]]
        print('ok')
    if cmd[0] == 'pop':
        if len(x) > 0:
            print(x[-1])
            x.pop()
        else:
            print('error')
    if cmd[0] == 'back':
        if len(x) > 0:
            print(x[-1])
        else:
            print('error')
    if cmd[0] == 'size':
```

```
        print(len(x))
    if cmd[0] == 'clear':
        x.clear()
        print('ok')
    c = input()
print('bye')
```

3 C. Правильная скобочная последовательность

Разбор

Давайте поймём, что если мы удалим из правильной скобочной последовательности S *сбалансированную* пару скобок, например, $()$, $[]$, $\{\}$, то она останется правильной.

Тогда последовательно удалив **все** пары сбалансированных скобок из S, мы гарантированно получим пустую последовательность.

Воспользовавшись этим, давайте будем удалять все найденные сбалансированные пары скобок в порядке обхода строки. Тогда если после обработки всех символов строки у нас ещё остались хоть какие-то скобки, то S не являлась ПСП.

Авторское решение

```
pairs = {
    ')': '(',
    ']': '[',
    '}': '{'
}

s = input()
stack = []
for c in s:
    stack.append(c)
    if len(stack) > 1 and c in pairs.keys() and stack[-2] == pairs[
    c]:
        stack.pop()
        stack.pop()
if len(stack) == 0:
    print('yes')
else:
    print('no')
```

4 D. Простая очередь

Разбор

В данной задаче предполагалось, что очередь будет реализована на двух стеках. Давайте возьмем два стека b и f и будем класть все добавляемые элементы на вершину стека b. Тогда при вызове команд *front* и *pop* возможны два случая:

1. Стек f пустой, тогда нам нужно переложить все элементы из стека b в него в обратном порядке
2. Стек f непустой и тогда мы просто берём нужный элемент с его вершины

Длина очереди определяется как сумма длин обоих стеков, очистка очереди также подразумевает очистку обоих стеков.

Авторское решение

```
b, f = [], []
c = input()
while c != 'exit':
    cmd = c.split()
    if cmd[0] == 'push':
        b.append(cmd[-1])
        print('ok')
    if cmd[0] == 'pop':
        if len(f) == 0:
            f = list(reversed(b))
            b = []
        print(f[-1])
        f.pop()
    if cmd[0] == 'front':
        if len(f) == 0:
            f = list(reversed(b))
            b = []
        print(f[-1])
    if cmd[0] == 'size':
        print(len(f) + len(b))
    if cmd[0] == 'clear':
        f = []
        b = []
        print('ok')
    c = input()
print('bye')
```

5 E. Сортировка выбором

Разбор

В данной задаче было необходимо при помощи сортировки выбором максимума упорядочить массив по неубыванию.

Ключевые моменты реализации:

1. Введём указатель k, который будет указывать на начало отсортированного участка, изначально $k = n - 1$
2. Запустим поиск индекса i_{max} максимального значения на отрезке от 0 до k, в конце обменяем a_k и $a_{i_{max}}$
3. Будем повторять до тех пор, пока $k > 0$

Авторское решение

```
n = int(input())
a = list(map(int, input().split()))
k = n - 1
while k >= 1:
    i_max = 0
    for i in range(1, k + 1):
        if a[i] > a[i_max]:
            i_max = i
    a[k], a[i_max] = a[i_max], a[k]
    k -= 1
print(' '.join(map(str, a)))
```

6 F. Сортировка вставками

Разбор

В этой задаче было необходимо отсортировать массив по неубыванию методом "вставок".

Зададим указатель i и пройдёмся им по массиву со 2 до n-го элемента, на каждом шаге будем смещать выбранный элемент влево до тех пор, пока его можно двигать и элемент слева больше него.

6.1 Авторское решение

```
n = int(input())
a = list(map(int, input().split()))
for i in range(1, n):
    k = i
    while k > 0 and a[k - 1] > a[k]:
        a[k], a[k - 1] = a[k - 1], a[k]
        k -= 1
print(' '.join(map(str, a)))
```

7 G. Пузырьковая сортировка

Разбор

В этой задаче было необходимо отсортировать массив по неубыванию методом "пузырька".

Пройдёмся по массиву n раз и на каждом шаге будем "поднимать"тяжёлые элементы "наверх"до тех пор, пока это возможно. Более строго, на i-м шаге зададим указатель $j = 0$ и пройдёмся им по массиву $n-1$ раз. Каждый раз, когда выполнится условие $a_j > a_{j+1}$, обменяем два соседних элемента.

Обратим внимание, что после каждого полного прохода указателем j по массиву последние $i+1$ элементов будут отсортированы, поэтому на i-м шаге нам достаточно совершить $n-i-1$ итераций.

Авторское решение

```
n = int(input())
a = list(map(int, input().split()))
for i in range(n):
    for j in range(n - i - 1):
        if a[j] > a[j + 1]:
            a[j], a[j + 1] = a[j + 1], a[j]
print(' '.join(map(str, a)))
```

8 H. Кража в магазине

Разбор

Обратим внимание, что единственный способ минимизировать предполагаемое число украденных клавиатур - считать, что нумерация клавиатур начиналась с минимального имеющегося числа в списке и заканчивалась максимальным.

В таком случае нам достаточно отсортировать массив по возрастанию и найти разность всех соседних элементов:

$$\sum_{i=1}^{N} a_i - a_{i-1} - 1$$

Асимптотика решения: $O(N \times \log N)$
Бонус: существует решение за $O(N)$

Авторское решение

```
n = int(input())
a = list(map(int, input().split()))
a.sort()
cnt = 0
for i in range(1, n):
    cnt += a[i] - a[i - 1] - 1
print(cnt)
```