1 Домашняя работа(Мироев Д.Ш.)

2. Используем структуру данных очередь с поддержкой максимума.

```
# digits - исходное число длины n, порядок цифр от старших к младшим.
# k - количество цифр, которые осталось вычеркнуть
def maxSubnumber(digits, k):
    if k == 0:
        return digits
    answer = list()     # тут будет лежать ответ
    n = len(digits)     # получаем n
    assert(k < n)
    q = QueueWithMax()  # создаём очередь с поддержкой максимума
    p = 0               # количество обработанных разрядов исходного числа
    # положим в очередь k + 1 наиболее значимых цифр
    # логика такова: для выбора максимального элемента самое главное выбрать
    # максимальную старшую цифру. Она может находиться среди первых k + 1
    # цифр, потому что в крайнем случае мы вычеркнем k самых левых цифр.
    while p < k + 1:
        q.enqueue(digits[p])
        p += 1
    while k > 0:
        # Удалим из очереди цифры до максимальной, то есть вычеркнем их
        m = q.max()
        while q.first() != m:
            q.dequeue()
            q.enqueue(digits[p])
            p += 1
            k -= 1
        # Добавим максимум в ответ
        answer.append(m)
        q.dequeue()
    return answer
```

Алгоритм работает за линейное время, потому что для каждой цифры исходного числа будет выполнено ровно две операции добавление в очередь и изъятие.

3.
```
# data      - массив из n элементов
# request   - тройки x, l, r из команд "add(x, l, r)"
def operate(data, requests):
    n = len(data)                   # получим n из массива элементов
    delta = [0 for _ in range(n)]   # создадим массив разностей размера n заполненый нулями
    for x, l, r in requests:        # (1) для всех команд
        delta[l] += x           # увеличим l-ую разность на x
        if (r + 1 < n):
            delta[r + 1] -= x   # уменьшим r+1-ую разность на x
    s = 0
    for i in range(n):  # (2) восстановим результат из массива разностей
        s += delta[i]
        data[i] += s
```

Алгоритм работает за время O(n + m), потому что цикл (1) содержит m итераций, а цикл (2) содержит n итераций, каждая из которых выполняется за константное время.

4.
```
# numbers - массив из n элементов
# k - необходимое количество различных элементов
def shortestSubintervalWithAtLeastKDifferentNumbers(numbers, k):
    n = len(numbers)  # получаем n
    repeats = [0 for _ in range(n)] # количество повторений индекса в текущем подотрезке
    best = [0 for _ in range(n)]    # длина кротчайшего подотрезка начинающегося с индекса
    d = 0 # количество различных элементов в текущем подотрезке
    i = 0 # начало текущего подотрезка
    j = 0 # конец текущего подотрезка
    while j < n: # (1)
        if d < k: # если текущий подотрезок не удовлетворяет условию удлинняем его вправо
            repeats[numbers[j]] += 1
            if repeats[numbers[j]] == 1:
                d += 1
            j += 1
        else: # в противном случае переходим к следующему начальному индексу
            best[i] = j - i
            repeats[numbers[i]] -= 1
            if repeats[numbers[i]] == 0:
                d -= 1
            i += 1
    # найдём минимальный подотрезок удовлетворяющий условию и вернём его или n в
    # случае, если такого не существует
    minLength = n
    minL = 0
    minR = 0
    for pos in range(n): # (2)
        if best[pos] != 0 and best[pos] < minLength:
            minLength = best[pos]
            minL = pos
            minR = pos + best[pos] - 1
    return (minL, minR)
```

Алгоритм работает за время O(n), потому что цикл (1) и цикл (2) содержат каждый по n итераций, каждая из которых выполняется за константное время.

5.
```
def opens(b):
    if b == ')':
        return '('
    elif b == ']':
        return '['
    elif b == '{':
        return '}'

# brackets - строка из трёх типов скобок
def longestCorrectSubinterval(brackets):
    s = [] # стек для открытых скобок
    n = len(brackets) # получаем n
    # в i-ой ячкйке этого массива будет храниться индекс закрывающей скобки для
    # открывающей в i-ой позиции строки
    corresponding = [-1 for _ in range(n + 1)]
    for i in range(n): # (1) заполняем массив corresponding проходом по строке
        b = brackets[i]
        if b == '(' or b == '{' or b == '[':
            # встретили открывающую, кладём её на стек вместе с индексом
            s.append((b, i))
        elif len(s) != 0 and s[-1][0] == opens(b):
            # встретили подходящую закрывающую скобку, значит подстрока от
            # индекса открывающей до текущего является правильной скобочной
            # последовательностью
            corresponding[s[-1][1]] = i
            s.pop()
        else:
            # встретили закрывающую без подходящей пары. правильных скобочных
            # последовательностей, содержащих такую подстроку не существует,
            # поэтому стек можно опустошить и продолжить поиск правильных
            # последовательностей со следующего символа.
            s = []
    i = 0
    maxLength = 0
    maxStart = 0
    maxEnd = 0
    # вычисляем длиннейшую подстроку являющуюся правильной скобочной
    # последовательностью на остнове массива corresponding
    # Код ниже работает по двум причинам:
    # 1. Подстрока от i до corresponding[i] - правильная, если corresponding[i] != -1
    # 2. Если подстрока от i до corresponding[i] и от corresponding[i] + 1 до
    #    corresponding[corresponding[i] + 1] правильные, то от i
    #    до corresponding[corresponding[i] + 1] тоже находится правильная подстрока
    while i < n: # (2)
        if corresponding[i] == -1:
            i += 1
        else:
            start = i
            while i < n and -1 != corresponding[corresponding[i] + 1]:
                i = corresponding[i] + 1
            i = corresponding[i] + 1
```

```
            end = i - 1
            l = end - start + 1
            if maxLength < l:
                maxLength = l
                maxStart = start
                maxEnd = end
    return (maxStart, maxEnd);
```

Алгоритм работает за линейное время, потому что цикл (1) выполняет ровно n итераций, а (2) не более n, каждая из которых занимает константное количество времени.

6.
```
def maxSum(numbers, l, r, a, b):
    if r == 1 or b == 1:
        return maximum(numbers)

    n = len(numbers)

    maxSum   = 0 # максимальная сумма отрезка удовлетворяющего условию
    maxLeft  = 0 # начало такого отрезка
    maxRight = 0 # конец такого отрезка

    # i-ая ячейка хранит количество вхождений i
    # в обозреваемом подотрезке
    repeats = [0 for _ in range(n)]

    i = 0 # индекс начала обозреваемого подотрезка
    j = 1 # индекс конца обозреваемого подотрезка плюс один
    d = 1 # количество различных чисел в обозреваемом подотрезке
    repeats[numbers[0]] += 1
    s = numbers[0] # сумма чисел в обозреваемом подотрезке
    while j < n:
        c = j - i # длина обозреваемом подотрезка
        x = numbers[j] # следующее за обозреваемым отрезком число
        # если последовательность не станет слишком длинной
        # и не будет содержать слишком много разных чисел
        if c + 1 < r and (repeats[x] != 0 or d + 1 < b):
            if repeats[x] == 0:
                d += 1
            # удлиняем её
            repeats[x] += 1
            j += 1
            s += x
        # в противном случае последовательнось максимальной суммы начинающаяся
        # с индекса i найдена и можно переходить к следующему индексу
        else:
            if maxSum < s:
                maxSum = s
                maxLeft = i
                maxRight = j - 1
            s -= numbers[i]
            repeats[numbers[i]] -= 1
            if 0 == repeats[numbers[i]]:
                d -= 1
            i += 1
    return (maxLeft, maxRight)
```

Подотрезок максимальной суммы, удовлетворяющий условию и начинающийся с первого числа, можно найти, последовательно добавляя числа и проверяя выполнение условия с использованием массива для подсчёта количеств символов различных символов в выбранной последовательности. Если добавление следующего символа нарушает условие, то подотрезок найден. Эта сумма записывается в массив maxSumStartingAt в индекс 0. Рассмотрим этот подотрезок, исключив первый символ. В нём не больше различных символов, чем в исходном, а значит и не больше чем можно по условию. Его длина меньше, чем длина исходного отрезка, а значит мешьне, чем можно по усло-

вию. Будем удлинять его вправо по такому же принципу, пока не найдём максимальную сумму, начинающюуся с этого индекса и удовлетворяющую условию. Повторим рассуждение необходимое число раз.