1 Обязательная часть

1. **Максимальное среднее арифметическое.**
   Заметим, что в условии ничего не сказано о длине искомого подотрезка. Тогда очевидно, что подотрезком с максимальным средним арифметическим яляется просто максимальный элемент массива. Найдем его при помощи двух переменных - $maxi$ (предполагаемый максимум) и tmp (элемент массива, который сейчас рассматриваем). Изначально в $maxi$ кладем первый эл-т, tmp - второй. Если эл-т tmp больше, чем $maxi$ - кладем его в $maxi$. Считываем в tmp следующий эл-т. Когда считаем все элементы данного на вход массива, в $maxi$ будет лежать максимум этого массива.
   Работает за $\mathcal{O}(n)$, так как все, что мы делаем - один пробег по массиву, да и тот при считывании.

2. **Оптимальный отрезок**
   Построим массив частичных сумм (на префиксах). Сумма на подотрезке $[l, r]$ будет равна $pref[r+1] - pref[l]$. Будем (цикл for) перебирать значения r в порядке возрастания. Тогда при фиксированном r максимум суммы достигается при минимальном $pref[l]$.
   При $r = i$ минимальное значение $pref[j]$ $(j = l)$ по условию должно находиться на отрезке $[i - R; i - L]$. Чтобы каждый раз не искать минимум, будем поддерживать очередь с минимумом.
   Но отрезок $[L, R]$ был не единственным ограничением. Найденный подотрезок содержит от A до B различных элементов. То есть интересует такое $pref[j]$, которое находится на отрезке $[s1, s2]$, где s1 - первая позиция такая, что отрезок $[s1 - 1, i]$ содержит не больше B различных чисел, а s2 - последняя позиция такая, что отрезок $[s2 + 1, i]$ содержит хотя бы A различных чисел. При увеличени i количество различных элементов точно не уменьшается, поэтому можно не сбрасывать s1 и s2 каждый раз, а продолжать двигать с прошлой итерации.
   Чтобы проверять, можно ли сдвинуть s2 вперед b нужно ли сдвинуть s1, будем поддерживать два массива-счетчика count1 и count2 каждого числа. При сдвиге i делаем $count1[a[i]] + +$ и $count2[a[i]] + +$, при сдвиге s1 делаем $count1[a[s1]] - -$, при сдвиге s2 - $count2[a[s2]] - -$. Также поддерживаем две переменных с числом ненулевых ячеек в count1 и count2, меняя каждый раз, когда только что измененная ячейка начала или перестала быть нулем.
   Теперь есть два ограничения отрезков, на которых можно работать при каждом i: $[i - R; i - L]$ и $[s1, s2]$. Возьмем их пересечение и будем поддерживать очередь с минимумом для него. На каждом шаге в нее будет либо приходить один и уходить один элемент, либо только приходить, либо только уходить.
   Работает алгоритм за $\mathcal{O}(n)$. За $\mathcal{O}(n)$ строим массив частичных сумм, а затем бежим по этому массиву один раз. На каждом шаге делаем четыре операции - сдвигаем s1, если надо, сдвигаем s2, если надо, считаем пересечение $[i - R; i - L]$ и $[s1, s2]$ и за $\mathcal{O}(1)$ достаем минимум.

3. **Максимизация числа**
Будем поддерживать очередь с максимумом, где храним уже считанные цифры числа. Считываем по одной слева направо. Заметим, что первая цифра нашего итогового числа будет находиться в диапазоне от первой цифры изначального числа до $k+1$-ой. Когда считали $k+1$ цифру вытаскиваем максимум на очереди (пусть это оказалась цифра m). Удаляем из начала все элементы вплоть до первого вхождения m. Саму m выводим на экран и тоже удаляем.
Если мы удалили i цифр, то осталось удалить $k' = k - i$. Тсли $k'+1$ элемент у нас в очереди все еще есть - повторяем. Если нет - добавляем сколько надо из несчитанных цифр. И так далее. Таким образом мы оставим в начале нашего нового числа только наибольшие доступные за k удалений цифры из старого. А значит новое окажется максимальным среди всех возможных.
Каждую цифру мы считаем по одному разу и удалим не больше k цифр. То есть алгоритм работает за линию.

4. **ПСП-подстрока**
Будем считать баланс строки: открывающая скобка — +1, закрывающая — -1. Пусть у нас есть массив pref балансов на префиксах (то же, что и частичные суммы). Заметим, что подстрока $[l, r]$ является ПСП, если $pref[r] - pref[l] == 0$, и для любого i, принадлежащего промежутку $[l, r)$, верно, что $pref[l] \leq pref[i]$. Используя это, найдем максимальную ПСП.
Воспользуемся стеком. Будем бежать по нашей последовательности (1, 1, -1, 1, и тд) и поддерживать стек позиций, после которых не встречался баланс меньший, чем на рассматриваемой позиции. То есть изначально в стеке лежит 0, когда мы встречаем открывающую скобку, то добавляем ее позицию в стек, а при закрывающей будем удалять последний элемент из стека (последнюю открывающую скобку), при этом пытаясь улучшить ответ.
Как улучшается ответ. При найденной новой ПСП $[i, j]$ смотреть на баланс строки с индексом $st.top() + 1$. Если баланс i равен балансу $st.top() + 1$, то длины их ПСП суммируем. Но нужно дополнительно поддерживать массив длин, где будут храниться длины найденных ПСП. Такой способ работает, так как если между найденными ПСП нет других скобок (позиций в стеке), значит их объединение тоже ПСП. Каждый раз при удалении скобки из стека проверяем не получили ли мы сумму больше уже имеющейся. Если да, то обновляем ответ, неправда - не обращаем внимания и просто дополняем наши длины найденной ПСП.
Алгоритм работает за $\mathcal{O}(n)$. Мы один раз считываем нашу скобочную последовательность, заменяя ее на 1 и -1. И один раз бежим по получившемуся массиву, работая со своим стеком.