1. **Как найти n-е число Фибоначчи, используя O(1) доп. памяти?**

* нужно ввести дополнительный массив, который будет служить как бы «кэшем» для наших вычислений: перед вычислением нового значения мы будем проверять, не вычисляли ли его раньше. Если вычисляли, то будем брать из массива готовое значение, а если не вычисляли — придётся считать его на основе предыдущих и запоминать на будущее:
* рассчитываем по формуле: f[i] = f[i-1] + f[i-2]

function fib\_n(n){  
 if (n <= 2) return 1;  
 var x = 1;  
 var y = 1;  
 var ans = 0;  
 for (var i = 2; i < n; i++){  
 ans = x + y;  
 x = y;  
 y = ans;  
 }  
 return ans;  
}

1. **Найдите наибольшую возрастающую подпоследовательность, в которой чередуется четность элементов.**

а) O(n^2)

int d[MAXN]; // константа MAXN равна наибольшему возможному значению n

if (arr[i] > arr[j] and (arr[i]+arr[j]) % 2 != 0 and dp[i] < dp[j] + 1){ dp[i] = dp[j] + 1

мы проходим все элементы arr [], и для каждого элемента мы проходим все элементы одного и того же массива до этой позиции и проверяем наши условия.

Если он удовлетворяет всем условиям, обновляем dp [j]  
Когда мы дважды обходим все элементы arr во вложенном цикле

б) O(n log n)

Мы берем два новых массива, один нечетный, а другой четный, и инициализируем оба с 1. Мы начинаем наш алгоритм со второго столбца. Мы проверяем элементы, которые предшествуют текущему элементу, с текущим.

Если текущий элемент нечетный, а элемент сравнения четный, то odd (индекс текущего элемента) = even (индекс компаратора) + 1;

Если текущий элемент четный, а элемент сравнения нечетный, то четный (индекс текущего элемента) = нечетный (индекс компаратора) + 1;

1. **Дана матрица, состоящая из нулей и единиц. Найдите квадрат с наибольшей стороной, полностью состоящий из единиц.**

Матрицу A будем просматривать по строкам справа налево, снизу вверх

Предположим, что текущий просматриваемый элемент a[i][j] (все элементы, лежащие в строках с номерами больше i, равно как и все элементы строки i правее a[i][j] считаются уже).

Если элемент a[i][j]=0, то его значение остается нулевым. Если же a[i][j]=1, то для того, чтобы найти максимальный размер квадрата из единиц, у которого (i,j) - верхний левый угол, проанализируем уже просмотренные элементы a[i][j+1], a[i+1][j+1], a[i+1][j] - соответственно длины сторон максимальных квадратов с левым углом справа, по диагонали вниз и просто вниз от данного элемента. Квадрат с верхним левым углом (i,j) может протянуться вправо набольше чем на a[i+1][j]+1, вниз - не больше чем на a[i][j+1]+1 и по диагонали не больше чем на a[i+1][j+1]+1. Следовательно, максимальная сторона квадрата есть

a[i][j]=min{a[i][j+1],a[i+1][j],a[i+1][j+1]}+1.

Max=0;{текущая максимальная длина стороны}

for i=n-1 to 1

for j=m-1 to 1

if a[i][j]!=0

a[i][j]=min(a[i][j+1],a[i+1][j+1],a[i+1][j])+1;

if a[i][j]>Max

Max =a[i][j]

1. **В дощечке в один ряд вбиты гвоздики. Любые два гвоздика можно соединить ниточкой. Требуется соединить некоторые пары гвоздиков ниточками так, чтобы к каждому гвоздику была привязана хотя бы одна ниточка, а суммарная длина всех ниточек была минимальна.**

1. В dp[i][0] будем хранить минимальную суммарную длину ниточек для того, чтобы связать (сделать так, чтобы на каждом гвоздике была хотя бы одна ниточка) первые i гвоздиков, если i-й гвоздик мы соединяем с предыдущим ниточкой. В dp[i][1] мы будем хранить лучший результат, если мы не соединяем этот гвоздик с предыдущим ниточкой. То есть предполагается, что мы соединим его ниточкой со следующим.

dp[i][0] — минимум среди dp[i-1][0] и dp[i-1][0] плюс длина ниточки до предыдущего гвоздика. Также i-й гвоздик мы не можем не связать (dp[i][1]) с предыдущим гвоздиком, если этот предыдущий гвоздик не связан со своим предыдущим, потому что тогда предыдущий гвоздик не будет связан ниточкой вообще ни с каким гвоздиком. Поэтому dp[i][1] = dp[i-1][0], т.к. dp[i-1][0] — лучший результат для первых i-1 гвоздиков, когда последний гвоздик связан с предыдущим ниточкой.

2. Первый гвоздик мы не можем связать с предыдущим, значит полагаем dp[1][0]=inf, где inf — такое число, что min(inf, a)=a для любой достижимый a.

dp[1][1]=0.

**dp[i][0] = min (dp[i - 1][0], dp[i - 1][1]) + (pos[i] - pos[i - 1]);**

**dp[i][1] = dp[i - 1][0];**

3. Ответ будет лежать в dp[n][0], т.к. в dp[n][1] лежит лучший результат, если последний гвоздик не связывать с предыдущим (последний гвоздик будет ни с чем не связан).