1. The City Council of Sham-Poobanana needs to partition Purple Street into voting districts. A total of n people live on Purple Street, at consecutive addresses $1, 2, \dots, n$. Each voting district must be a contiguous interval of addresses $i, i+1, \dots, j$ for some $1 \le i < j \le n$. By law, each Purple Street address must lie in exactly one district, and the number of addresses in each district must be between k and $2k$, where k is some positive integer parameter.

   Every election in Sham-Poobanana is between two rival factions: Oceania and Eurasia. A majority of the City Council are from Oceania, so they consider a district to be *good* if more than half the residents of that district voted for Oceania in the previous election. Naturally, the City Council has complete voting records for all n residents.

   For example, the figure below shows a legal partition of 22 addresses into 4 good districts and 3 bad districts, where $k = 2$. Each `O` indicates a vote for Oceania, and each `X` indicates a vote for Eurasia.

   

   Describe an algorithm to find the largest possible number of *good* districts in a legal partition. Your input consists of the integer k and a boolean array GoodVote$[1 .. n]$ indicating which residents previously voted for Oceania (True) or Eurasia (False). You can assume that a legal partition exists. Analyze the running time of your algorithm in terms of the parameters n and k.

2. Suppose we want to split an array $A[1 .. n]$ of integers into k contiguous intervals that partition the sum of the values as evenly as possible. Specifically, define the *cost* of such a partition as the maximum, over all k intervals, of the sum of the values in that interval; our goal is to minimize this cost. Describe and analyze an algorithm to compute the minimum cost of a partition of A into k intervals, given the array A and the integer k as input.

   For example, given the array $A = [1, 6, -1, 8, 0, 3, 3, 9, 8, 8, 7, 4, 9, 8, 9, 4, 8, 4, 8, 2]$ and the integer $k = 3$ as input, your algorithm should return the integer 37, which is the cost of the following partition:

   $$\Big[\ \overbrace{1, 6, -1, 8, 0, 3, 3, 9, 8}^{37}\ \Big|\ \overbrace{8, 7, 4, 9, 8}^{36}\ \Big|\ \overbrace{9, 4, 8, 4, 8, 2}^{35}\ \Big]$$

   The numbers above each interval show the sum of the values in that interval.

3. A sequence of integers is *mostly odd* if strictly more than half of its elements are odd. Describe an algorithm that computes the length of the longest *mostly odd* increasing subsequence of a given array $A[1 .. n]$ of integers. (You can assume that A has at least one mostly-odd increasing subsequence.)

   For example, given the input array

   $$[2, 4, 6, 8, 10, \mathbf{1}, \mathbf{7}, 12, \mathbf{13}, 14, \mathbf{19}, \mathbf{25}, 16, 18, 20, 22, 24],$$

your algorithm should output the integer 7, which is the length of the mostly-odd increasing subsequence [4, 6, **7**, **13**, 14, **19**, **25**]. (This is not the only mostly-odd subsequence of length 7.) The increasing subsequence [2, 4, 6, **7**, 12, **13**, 14, **19**, 20, 22, 24] is longer, but it is not mostly odd.

4. The StupidScript language includes a binary operator `@` that computes the *average* of its two arguments. For example, the StupidScript code `print(3 @ 6)` would print `4.5`, because $(3+6)/2 = 4.5$.

   Expressions like 3 @ 7 @ 4 that use the @ operator more than once yield different results when they are evaluated in different orders:

$$(3 \mathbin{@} 7) \mathbin{@} 4 = 5 \mathbin{@} 4 \; = \; 4.5 \qquad \text{but} \qquad 3 \mathbin{@} (7 \mathbin{@} 4) = 3 \mathbin{@} 5.5 \; = \; 4.25$$

   Here is a larger example:

$$\begin{aligned}
((((8 \mathbin{@} 6) \mathbin{@} 7) \mathbin{@} 5) \mathbin{@} 3) \mathbin{@} (0 \mathbin{@} 9) \; &= \; 4.5 \\
((8 \mathbin{@} 6) \mathbin{@} (7 \mathbin{@} 5)) \mathbin{@} ((3 \mathbin{@} 0) \mathbin{@} 9) \; &= \; 5.875 \\
(8 \mathbin{@} (6 \mathbin{@} (7 \mathbin{@} (5 \mathbin{@} (3 \mathbin{@} 0))))) \mathbin{@} 9 \; &= \; 7.890625
\end{aligned}$$

   Your goal for this problem is to describe and analyze an algorithm to compute, given a sequence of integers separated by @ signs, the ***largest possible*** value the expression can take by adding parentheses. Your input is an array $A[1..n]$ listing the sequence of integers.

   For example, if your input sequence is [3, 7, 4], your algorithm should return 4.5, and if your input sequence is [8, 6, 7, 5, 3, 0, 9], your algorithm should return 7.890625. Assume all arithmetic operations (including @) can be performed exactly in $O(1)$ time.

   (a) Tommy Tutone suggests the following natural greedy algorithm: Merge the adjacent pair of numbers with the smallest average (breaking ties arbitrarily), replace them with their average, and recurse. For example:

$$\begin{gathered}
8 \mathbin{@} 6 \mathbin{@} 7 \mathbin{@} 5 \mathbin{@} \underbrace{3 \mathbin{@} 0} \mathbin{@} 9 \\
8 \mathbin{@} 6 \mathbin{@} 7 \mathbin{@} \underbrace{5 \mathbin{@} 1.5} \mathbin{@} 9 \\
8 \mathbin{@} 6 \mathbin{@} \underbrace{7 \mathbin{@} 3.25} \mathbin{@} 9 \\
8 \mathbin{@} \underbrace{6 \mathbin{@} 5.125} \mathbin{@} 9 \\
\underbrace{8 \mathbin{@} 5.5625} \mathbin{@} 9 \\
\underbrace{6.78125 \mathbin{@} 9} \\
7.890625
\end{gathered}$$

   Tommy reasons that with an efficient priority queue, this algorithm will run in $O(n \log n)$ time, which is *way* faster than any dynamic programming algorithm.

   Prove that Tommy's algorithm is incorrect, by describing a specific input array and proving that his algorithm does not yield the largest possible value for that array.

   (b) Describe and analyze a correct algorithm for this problem. Poor, poor Tommy.