# 2018 Battle of Brains Editorial

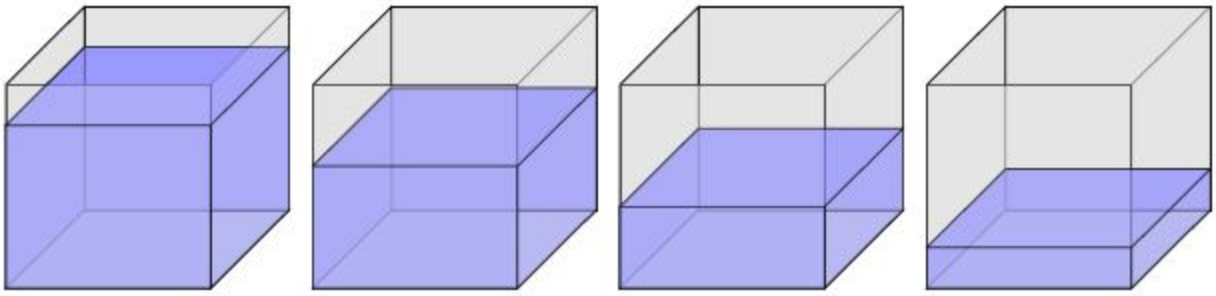
## Problem A:

Straightforward. Check whether 2*(a+b+c+d) >= (n+1) or not.

## Problem B:

**Problem Statement:**
You are given mass and density of a liquid called vibranium gift. You need to calculate the minimum surface area needed to wrap the liquid.

**Solution:**
In order to wrap some liquid with a wrapping paper, you will need to make a container to hold the liquid within. The shape of the container will determine the surface area needed to wrap the liquid. Of course, we need to choose a container so that after filling it up with all the liquid, there won't be any free space remaining in the container. We can't choose a container like following:



But what we can do is, we can reduce the size of the container ( from the above picture ) thus reducing the surface area needed to wrap the whole container ( this actually leads to wrap the liquid as well. )

**Can we do better?**
What if we used a spherical container to hold the liquid?

Let's consider $A_1$ as the surface area for the cubic container.
Let's consider $A_2$ as the surface area for the spherical container.

We know from standard physics that:

$$volume = \frac{Mass}{Density}$$

We know that if the side length of a cube is $a$ , then it has a volume of $V = a^3$ . and surface area of $A_1 = 6a^2$
Having a radius of $r$ , a spherical container can have a volume of $V = \frac{4}{3}\pi r^3$ and surface area of $A_2 = 4\pi r^2$

As $V$ is same for the above two cases, we can rewrite the equations for surface area as:

$$A_1 = 6V^{\frac{2}{3}}$$
$$A_2 = 4(\frac{3}{4})^{\frac{2}{3}}\pi^{\frac{1}{3}}V^{\frac{2}{3}} = 4^{\frac{1}{3}}3^{\frac{2}{3}}\pi^{\frac{1}{3}}V^{\frac{2}{3}}$$

Comparing the above two equations, it is clear that $A_2 < A_1$ will be true. In fact, it will be true for any value of $V$ . Because $V^{\frac{2}{3}}$ gets cancel out and $4^{\frac{1}{3}}3^{\frac{2}{3}}\pi^{\frac{1}{3}}$ =4.835976... which is clearly less than 6.

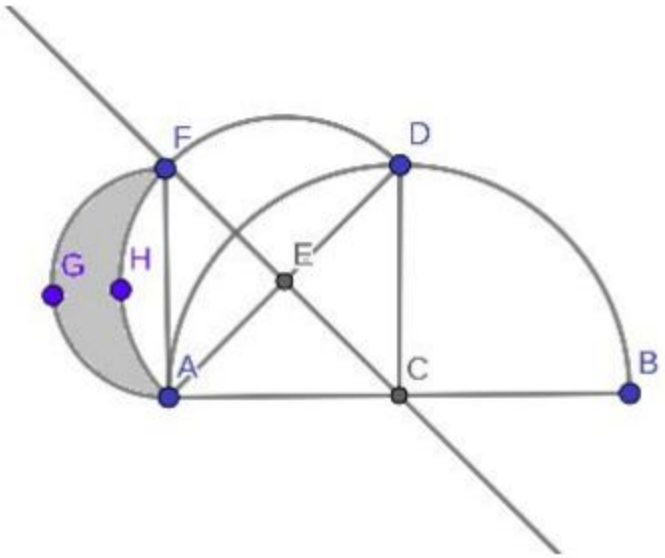
Anyone can try with any kind of container but the most optimized solution will always come out from the spherical one. So, we need to choose a spherical container and print it's surface area.

## Problem C:

Problem Statement:

In this problem, a radius of a semicircle is given. You need to determine the area which is specified in the problem statement.

Solution:




We know the radius of the semicircle ADBCA. So the radius is AC/CD. Now we have to determine the area of AGFHA.

So the solution is:

Area of **AGFHA** = Area of semicircle **AGFA**+Area of triangle **AEF**
- Area of quarter-circle **AHFEA**

- Area of a circle is $\pi$*(radius)$^2$
- Area of a triangle is ½ * height * base

**Hints:**

1. We can get the value of AE from ½ AD. AD will be derived from AC and CD.
2. As AE = EF (both are radius of AHFDEA semicircle) we can easily get the value of AF here.

# Problem D:

**Palindrome and Chocolate**

The task is to find the nth odd length palindromic number. The idea should come easily by listing the odd length palindromic numbers:

n=1 → answer=1
n=2 → answer=2
…
n=9 → answer=9
n=10 → answer=101 (not 11, because of odd length)
n=11 → answer=111
…
n=19 → answer=191
n=20 → answer=202
n=21 → answer=212
…
n=98 → answer=989
n=99 → answer=999
n=100 → answer=10001 (not 1001, because of odd length)
n=101 → answer =10101
…
n=1234 → answer=1234321
…

Observation: The nth odd length palindromic number can simply be formed by first writing n and then appending the digits of n in the order of increasing significant digits (excluding the least significant one or, the last digit of n).

Coding: For each case, first print the case number followed by the value of n. Then immediately remove the last digit of n by dividing by 10 (integer division). Next to that, a loop is enough to extract all the remaining digits of n. Until n decreases into zero, in each iteration print (n%10) (the last digit of current value of n) and divide n by 10 (integer division, to remove the last digit of n). Finally, print a newline character to put an end to the case.

## Problem E:

Greedy!
If 2^d + 2^(d-1) + 2^(d-2) +... 2^0 < X, the robot cannot reach to X.
Otherwise, Move facing X (left/right which is necessary). You will eventually reach to X.
Why would it work?

Because we can make all the numbers in [0, 2^(d+1)-1] by just adding and subtracting the power of two's (maybe skipping the last few of them).
Complexity: O(d)

# Problem F:

**Solution**

The probability should be p/q where p is the number of numbers which make tree and q is the total number which is given in input.
This problem basically needs some case analysis.
Suppose you have a number which has three prime divisors p1, p2 and p3 then what will be the divisors? Obviously p1, p2, p3, p1*p2, p1*p3, p2*p3 (Removing 1 and the number). And here any pi*pj will provide pi and pj as prime divisor which we already had in our graph, so that will make a cycle. So in this way we can easily prove that the numbers with more than 2 divisors(extracting 1 and N) or more specifically the numbers which sum of powers of primes more than 2 will always provide a cycle.

**Coding**

Finding the sum of power of primes is not a big deal but look twice carefully, there is 10^6 test cases and we may need to do the job for the number 10^6 which may cost 10^9 operations with square root operation per test case and also we need to find the gcd which will cost some more . So, we need to do it wisely. As there is no update or something, we can easily preprocess and keep cumulative sum. In this way we can answer the test cases in order of finding gcd which is approximately logn.

# Problem G:

For each character $S[i]$ of string $S$, let $d[i]$ be the sum of the length of all possible substrings that contain $S[i]$. Now, the given problem is transformed into calculating the following sum-

$$\sum_{i=1}^{n} S[i] * d[i]$$

In order to find $d[i]$ for each $S[i]$, we'll first solve a smaller dp problem - "For a given string $T$ with length $len$, find the sum of the lengths of all possible substrings of the string".
Let $dp[len]$ be the answer to the problem, then we'll find,

$$dp[len] = dp[len - 1] + \frac{len * (len + 1)}{2}$$

The reason behind is that, if we fix the right end of the substring to be $T[len]$, we can have these lengths- $1, 2, \ldots, len$ taking $T[len], T[len - 1], \ldots, T[1]$ as left end respectively. These lengths sum up to $\frac{len * (len + 1)}{2}$. Taking right end between $1$ and $len - 1$, we can find all possible solutions in $dp[len - 1]$, which is a smaller subproblem.

For the original problem, let's find the sum of all possible substring of the string $S$ which doesn't contain $S[i]$ as a character. $dp[i - 1]$ is the length-sum of all the substrings whose both ends lie on the left of $S[i]$. And similarly, $dp[n - i]$ is the length-sum of all the substring whose both ends lie on the right side of $S[i]$. Except those above two types of substrings, every possible substring will contain the character $S[i]$. So, we get,

$$d[i] = dp[n] - (dp[i - 1] + dp[n - i])$$

Finally we loop over the string, calculate $d[i]$ and add $s[i] * d[i]$ to our answer. This takes $O(N)$. The dp takes additional $O(N)$. Thus, overall complexity is $O(T * N)$.

---

# Problem H:

Since the given string length is rather small, we can actually precompute the tolerance value $t(i,j)$ for all possible substrings $s[i..j]$ in $O(|s|^2)$ time and memory using the recursive formula: $t(i,j) = |s_i - s_j| + t(i+1, j-1)$. The base cases are easy to figure out. Notice that since $|x|$ is never negative, the values satisfy $t(i,j) \geq t(i+1, j-1)$ for all valid $(i,j)$ pairs. Thus answering a query reduces to a simple binary search that finds the largest $k$ satisfying $t(i+k, j-k) \leq M$ for given query inputs $i, j, M$. This works in $O(|s|^2 + q \lg |s|)$ time and $O(|s|^2)$ memory.

# Problem I:

**Problem Idea:**

**<u>Easy:</u>**

Given a array of N distinct integer calculate the average number(expected value) of local maxima in a random permutation of the array. We say a array index is a local maxima if the number in that index is bigger than all its neighbour. Example: 3,1,2,5,4 here index 0 and index 4 are two local maximas.

**Medium:**

Solve it for 1,2 and 3 dimensional array.

**Hard:**

Solve it for more than 3 dimension. If you can solve it for 4th dimension you probably would be able to solve it for any dimension.

**Note:** For higher dimension we define neighbour as adjacent cells that shares a common side.
For higher dimensional array all dimension has same length. So a 2d N*N size array, 3d N*N*N size array etc. For higher dimension it would be hard to give all the array element as input. So we may define 1 to N^dimension as array element and only give N as input. The idea of the easy version is from a old Putnam Mathematical Competition. But it is possible to come up with a general solution for any dimension. N>=2.

**Solution:**
Easy: O(1)
Medium : O(1)
Hard: O(D logD)

**1D =** $\frac{n+1}{3}$
**2D =** $\frac{3n^2-3n+2}{15}$
**General Solution =** $\sum_{i=0}^{D}(2^i * {}^{D}C_i * (n-2)^{(D-i)} * \frac{1}{2D-i+1})$

**Explanation:**
Let's think about an easier version. Suppose we have an array of length 3. What is the probability that the middle cell will be local maxima in a random permutation?
Well, since we have 3 distinct integer one will be small (S) one will be medium (M) and one will be large (L). Now, SLM, MLS creates local maxima. LMS, LSM, SML, MSL

does not. So there is (2!/3!) = 2/6 = ⅓ chance that it would be local maxima. By the same logic, the corner two cells will have a ½ probability of being local maxima. So the Expected number of local maxima for a random permutation of a 1D array of size 3 is 1*(⅓)+2*(½) = 4/3. And so answer for a 1D array for size N is

$$(n-2)\,(\tfrac{2!}{3!}) \;+\; 2 \;*\; (\tfrac{1!}{2!})$$

But why this simple summing up works? The linearity of expectation. Let's consider a 2D array of size N*N. There are (N-2)*(N-2) cell with 4 neighbors. 4*(N-2) cell with 3 neighbors and 4 cells with 2 neighbors. So the answer is

$$(n-2)^{2} \;*\; (\tfrac{4!}{5!}) \;+\; 4 \;*\; (n-2) \;*\; (\tfrac{3!}{4!}) \;+\; 4 * (\tfrac{2!}{3!})$$

And for 3D it will be:

$$(n-2)^{3} \;*\; (\tfrac{6!}{7!}) \;+\; 6 \;*\; (n-2)^{2} \;*\; (\tfrac{5!}{6!}) \;+\; 12\,(n-2)\,(\tfrac{4!}{5!}) \;+\; 8 \;*\; (\tfrac{3!}{4!})$$

And the general formula is :

$$\sum_{i=0}^{D} \left( (2^{i} \;*\; {}^{D}C_{i}) \;*\; (n-2)^{(D-i)} \;*\; \tfrac{(2D-i)!}{(2D+1-i)!} \right)$$

## Problem J:

This problem basically asks you to write a simulator for a tennis match, a tennis tournament, update rankings and keep track of data of each player. A simple brute force simulator should work.