PROBLEM OF THE MONTH

POM

2015-2016

October, 2015

MATHEMATICS

5 points:

Seven friends are deciding who is going to run to the store to get bagels. They have a traditional die with six faces numbered 1 to 6. How could they choose one of them fairly using that die?

Solution:

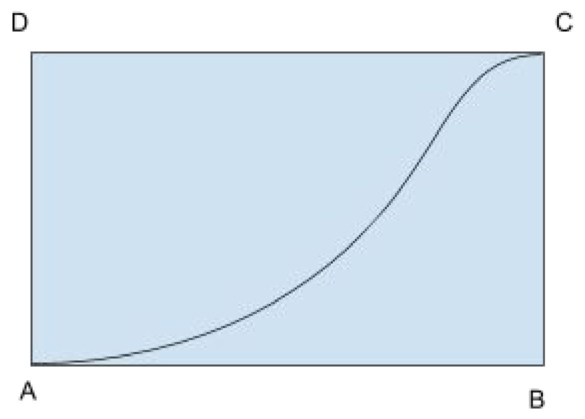
If the fiends roll the dice twice, there are 36 possible outcomes that can be represented as a table 6x6. Each of the 7 friends has to have 5 cells in the table assigned to him, for instance in this way:

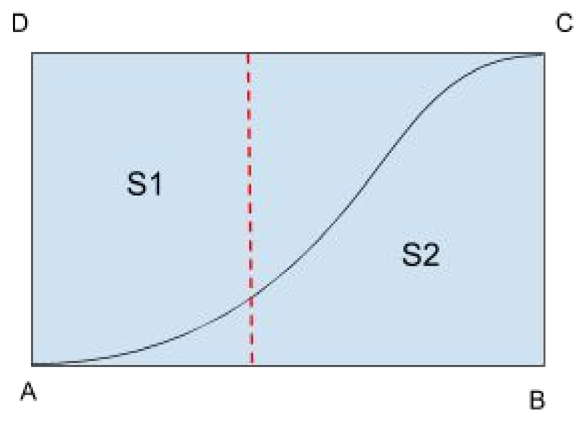
1	1	1	1	1	7
2	2	2	2	2	7
3	3	3	3	3	7
4	4	4	4	4	7
5	5	5	5	5	7
6	6	6	6	6	roll again

In the case that both times they've got '6', the procedure has to be repeated again (and so on). The chance for this to happen is relatively small: 1 out of 36.

10 points:

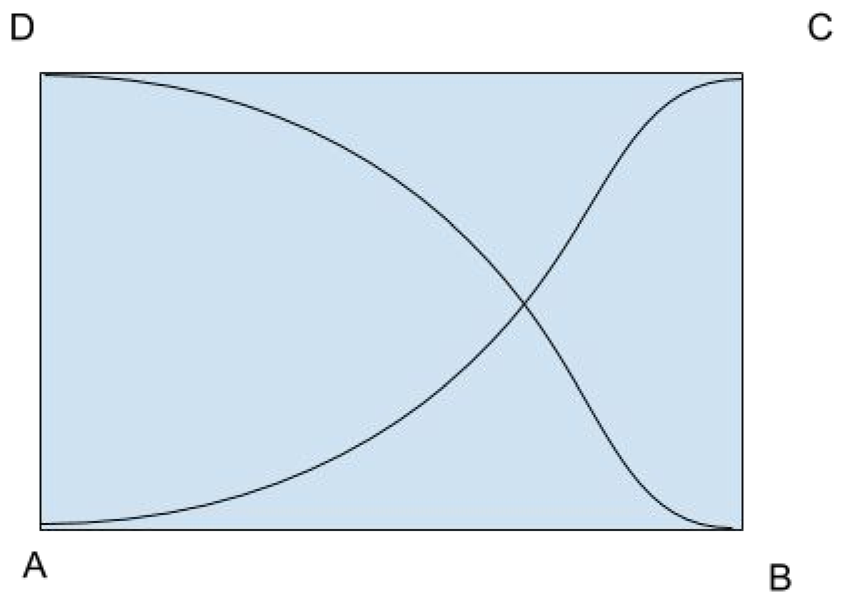
A cake that has the shape of a rectangle ABCD is cut along a curved line that goes from point A to point C, as shown in the figure, always going up. You are allowed to cut the cake along any vertical line and take the two pieces that contain points B and D, respectively. Assuming that you are not on a diet and want to get as much of a cake as possible, describe how should you cut it to obtain the maximum combined size of the two pieces, S1+S2, and substantiate your answer.





Hint:

Draw a line from D to B which is symmetric to the one connecting A and C, as shown in the Figure. Now, using this Figure, try to reformulate the problem so that your pieces of cake contain points D and C.

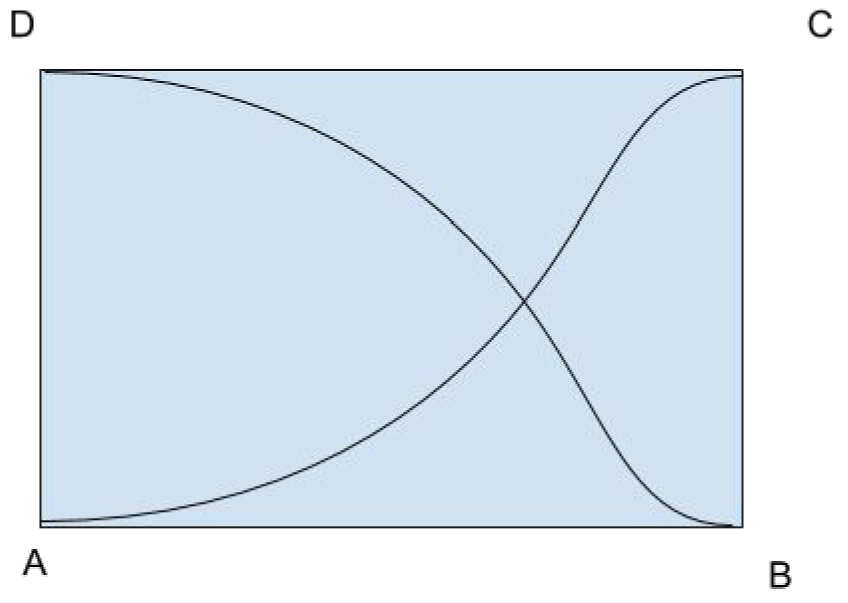


Answer:

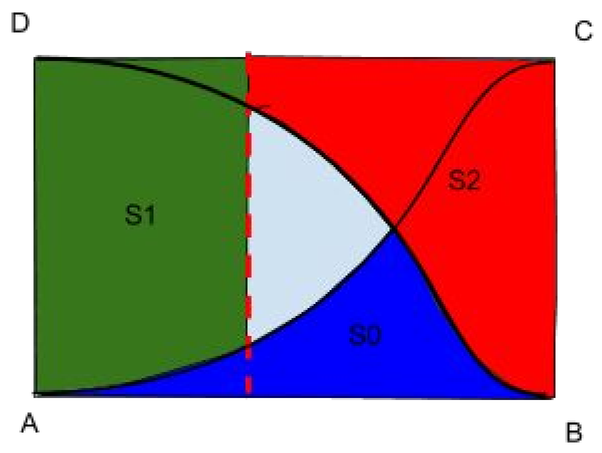
The vertical line must intersect the curve AC at the point that is halfway between the horizontal sides AB and CD.

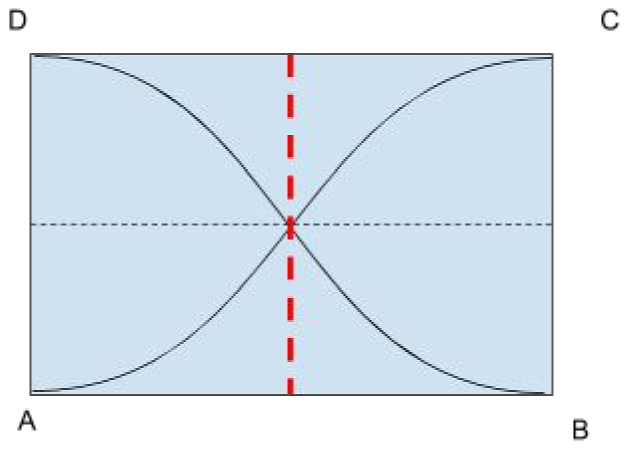
Solution:

Let us draw a line from D to B which is symmetric to the one connecting A and C, as shown in the Figure:



Now, S1 and S2 correspond to areas of green and red and green parts, respectively:



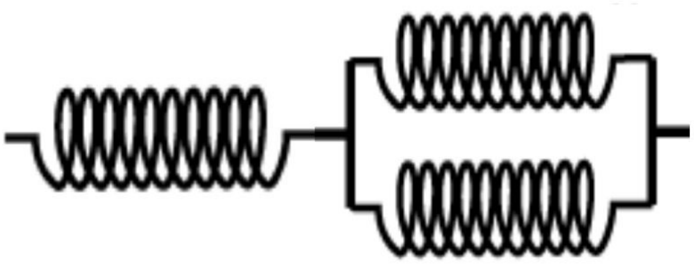


As it is clear from the picture, regardless of where the vertical line is, green and red areas will not contain the region colored blue. This means that the most you can get is the whole cake minus blue region. For that, the cut should be made via the intersection of the who symmetric curves AC and BD. In other words, the vertical line must intersect the curve AC at half the height of the rectangle. The vertical line must intersect the curve AC at the point that is halfway between the horizontal sides AB and CD.

PHYSICS

5 points:

Three identical springs having spring constant k are connected as shown in the figure. Find the effective spring constant of the system.



Hint:

Fix one end of the system and apply the force F to the other end. Find the tension of every spring. What is the distance X by which the system is stretched?

Answer: $K_{eff} = \frac{2}{3}k$

Solution: Let us fix one end of the system and apply the force F to the other end. Considering springs in equilibrium it is easy to see that the same force is applied to the single spring on the left side of the figure. Therefore, the extension of the length of that spring is $X_1 = F/k$. The force applied to each of the springs on the right side of the figure is $F/2$ and they are stretched by $X_2 = F/2k$. The total extension of the system is $X = X_1 + X_2 = F/k + F/2k = 3F/2k$. The effective spring constant is obtained as $K_{eff} = F/X = \frac{2}{3}k$.

.

10 points:

The vessel has a form of a solid of revolution (this means that any horizontal cross-section of the vessel is a circle of some radius, r; this is e. g. the case for a typical glass, or a bowl). A small hole is punched at the very bottom of the vessel. It is known that the level of water in the vessel is decreasing at constant rate due to the water leak from the hole. Find the shape of the vessel.

Hint:

1. See solutions to September Physics POM problems
2. Knowing the exit velocity of the fluid from the hole of a given size try to calculate the rate at which the water level is decreasing.

Answer: $h \sim r^4$ The shape of the vessel is a solid obtained by a revolution of the quartic parabola.

Solution:

If a is an area of the hole and V is a rate at which the level of the water is decreasing we have an equation $a\sqrt{2gh} = \pi r^2 V$ which describes the continuity of the fluid flow. Here h is the height of the water level at a given time and r is the radius of the vessel's cross section at this level. From this equation we obtain

$r = \sqrt{\frac{a\sqrt{2gh}}{\pi V}} \sim h^{1/4}$ or $h \sim r^4$.

CHEMISTRY

5 points:

Alice and Bob, her technician, have been preparing for tomorrow class devoted to the study of the properties of bases and acids. Alice asked Bob to prepare 1% solution of sodium hydroxide. Upon having examining lab's shelves, Bob found the bottle with solid sodium hydroxide was empty. “Alice, we have a problem”, he said. “We have no sodium hydroxide, no potassium hydroxide, no hydroxides of other alkaline metals. What will we do? Should we cancel tomorrow's class?”
“Yes, Bob, that is a problem. An interesting problem,” - Alice replied. “That is a good opportunity for us to check if we are good problem-solvers. Let's see what we have on our shelves. Sodium chloride, barium nitrate, calcium iodide, manganese acetate, barium hydroxide, sodium phosphate, sodium carbonate - Bob, we have enough chemicals to prepare sodium hydroxide solution!” She took a sheet of paper and wrote one formula on it. “You take this, dissolve in water…” “Stop, Alice, please!” - Bob exclaimed. “Don't explain it! Now I myself realize which chemicals I should take. You may go home, I'll prepare sodium hydroxide solution.”
Which formula did Alice write, and how will Bob prepare sodium hydroxide?

Hint:

A reaction between a soluble salt of a metal X and a soluble hydroxide of a metal Y yields a hydroxyde of the metal X if the second product of this reaction is ….

Answer:

n/a

Solution:

Briefly, if you mix sodium carbonate with barium hydroxide, you obtain two new compounds, sodium hydroxide and barium carbonate. The later is insoluble, and can be filtered out.
The reaction is as follows:

$$Na_2CO_3 + Ba(OH)_2 \rightarrow 2NaOH + BaCO_3$$

To get a pure solution of NaOH (without traces of barium), Bob has to take $23 \cdot 2 + 12 + 16 \cdot 3 = 106$ grams of sodium carbonate and $137 + (16+1) \cdot 2 = 171$ grams of barium hydroxide (I believe you understand that I took atomic weights and calculated molecular masses using standard elementary school arithmetics rules).
By the way, this method is close to the most common method that was used to produce NaOH

in past. The only difference is $Ca(OH)_2$ was used instead of $Ba(OH)_2$

Below is the explanation of this and wast majority of similar reactions.
The compounds listed by Alice are salts and hydroxides. The compounds of that type are *ionic*, which means in solution they spontaneously decompose to *ions*, molecular fragments having an electric charge. The first fragment that forms in that process (which is called *dissociation*) is a metal ion (which is always positive), and the second fragment is negative. For example, table salt (sodium chloride, of NaCl) dissociates as follows:

$NaCl \rightarrow Na^+ + Cl^-$

This process is reversible, and when you evaporate salty water, the ions associate back into the solid crystals of NaCl. However, in the aqueous solution of NaCl, there is no NaCl molecules, just Na^+ and Cl^- ions.
What is we mix two different salts, for example, sodium chloride (NaCl) and potassium bromide (KBr)? In water solution, both salts dissociate:

$NaCl \rightarrow Na^+ + Cl^-$
$KBr \rightarrow K^+ + Br^-$

which means the solution will contain four different type ions, Na^+, Cl^-, K^+, and Br^-. What is important, the ions in water solution are totally free, so they do not remember where they came from, for example, if we mix not NaCl with KBr, but NaBr with KCl, the solution would have the same composition, i.e. the same four ions would be present there.
If we decide to evaporate this solution, will we get back NaCl and KBr, or NaBr and KCl? Actually, since, for example, potassium ion does not “remember“ whether it came from KCl or KBr, it can from both salts upon evaporation. The same is true for sodium, and for all other ions. That means the mixture of NaCl and KBr solutions yields, upon evaporation, a complex mixture of all four possible combinations, i.e. NaCl, NaBr, KCl, and KBr.
However, you probably know that some salts (or hydroxides, and other ionic compounds) are not soluble in water. For example, chalk ($CaCO_3$) or gypsum ($CaSO_4$) are extremely insoluble. What does it mean? That means Ca^{2+} ions and CO_3^{2-} ions cannot coexist in aqueous solution.
The same is true for barium and CO_3^{2-} ions. When Bob dissolves barium hydroxide in water, barium hydroxide dissociates:

$Ba(OH)_2 \rightarrow Ba^{2+} + 2\ OH^-$

When Bob dissolves sodium carbonate in water (in a *separate* flask), it dissociates too

$Na_2CO_3 \rightarrow 2\ Na^+ + CO_3^{2-}$

Now, when these two solutions are mixed together, the resulting "soup" will contain all four ions: Ba^{2+}, OH^-, Na^+, and CO_3^{2-}
However, Ba^{2+} and CO_3^{2-} *cannot coexist in solution*, because they form insoluble precipitate:

$Ba^{2+} + CO_3^{2-} \rightarrow BaCO_3 \downarrow$

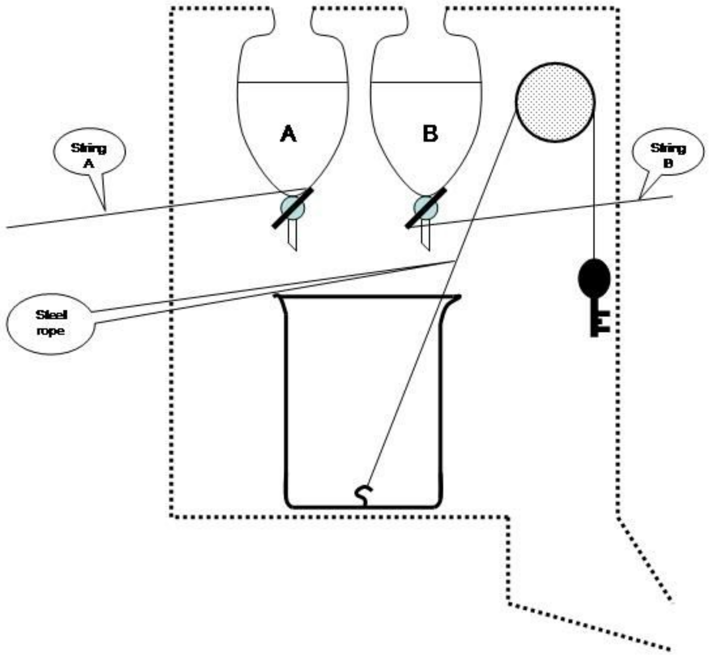
(the arrow pointing down means the product precipitates).
That means barium and carbonate ions remove each other from the solution. What is left? Just sodium and hydroxide ions, i.e. exactly what we would have obtained if we took a solid NaOH from the bottle, and dissolved it in water.
In summary:
When you mix aqueous solutions of two ionic compounds, these compounds may react with each other to produce two new compounds, however that will occur only if one of new compounds is insoluble and precipitates.

10 points:

Imagine you are playing the Escape the Room game. The key that (probably) opens the door is in the transparent plastic box shown on the figure below. The key is attached to a steel rope that hangs down a freely rotating wheel, and its opposite end is connected to the hook at the bottom of a glass beaker. Upon examining the box you realize the only way to get a key is to break the rope to let the key to fall down through the window at the bottom of the box. There are two dropping funnels (i.e. glass vessels with stopcocks at the bottom) filled with some unknown liquids. By pulling the string A you open a stopcock in the funnel A. By pulling the string B you open a stopcock in the funnel B. Once a stopcock is open, you cannot close it, so all the liquid from the funnel will be poured to the beaker. The necks of both funnels are open, as shown on the picture, so you can drop some small objects into the funnels (or take something from them).
The plastic box shown on the picture is attached to the table, you cannot move it.

In addition, in the room you found various things including:

- porcelain cups, dishes, and teapot;
- a pair of goggles and latex gloves;
- a silver spoon and German silver forceps;
- a bar magnet;
- two pipettes; they are long enough to withdraw a liquid from funnels A or B;
- a plastic model of the molecule of sulfuric acid; one oxygen atom was labeled with an asterisk;
- a book “Black holes and time warp” by Kip Thorne, there was a bookmark between pages 110 and 111, and the figure 2.5 on the page 110 was underlined with a yellow marker;
- a lighter;
- a candle;
- a bottle of vinegar;
- a flyer “SigmaCamp 2013”
- 10 pennies, two Canadian dollars, one 10 Russian roubles coin, one 500 Costa Rican colones coin;
- a globe where Australia, France and China capitals have been marked with a red marker;
- two small jars with metal shavings; both metals look light, silverish, and non-magnetic; your attempt to ignite the shavings from one jar was successful: the shavings burned producing bright white solid; when you withdrew (using a pipette) some liquid from each of two funnels and dropped it onto the shavings from the second jar, both liquids reacted with shavings to produce bubbles of some gas; when you withdrew some amount of the liquid from the funnel A, poured it to a cup, and added some amount of a liquid from the funnel B, the cup became warm.
- an odd box with 32 switches, not connected to other devices;
- a bottle of Poland spring sparkling water;
- and many other potentially useful things.



Propose the best strategy to get the key. 3 extra points may be given to those who solves the problem after obtaining a hint (8 pts maximum).

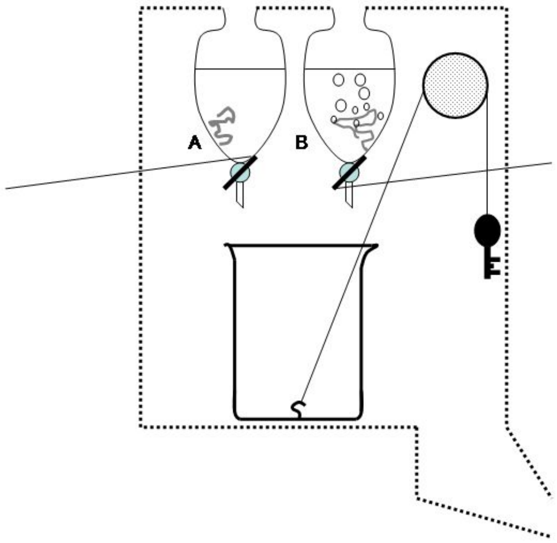
Hint:

The fact that the two liquids when mixed together produce heat means they are two different substances. Since both liquids react with metal shavings, these two liquids most likely are the solutions of some acid and some base, accordingly (two different acids very rarely react with each other; and one category of metals exists that reacts both with acids and bases). Unfortunately for us, iron reacts only with acids. Therefore, to dissolve the steel rope and to release the key, you need to pour an acid to the beaker. If you pull a wrong string, a base solution will be poured into the beaker, and it will be senseless to pull the second string after that, because the base (which is already in the beaker) will neutralise the acid, and the rope will not dissolve. Therefore, you have to decide which solution is an acid. To do that, try to guess what the shavings in the jars are. Which metals react both with acids and bases, and which metals react only with acids? Which metals can burn? May be, if you identify these metals you will know what to do.

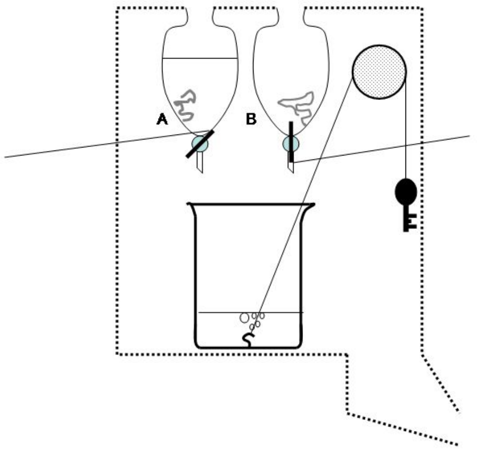
PS. It is highly likely that the shavings in the jars are not exotic or precious metals, but something very common and inexpensive....

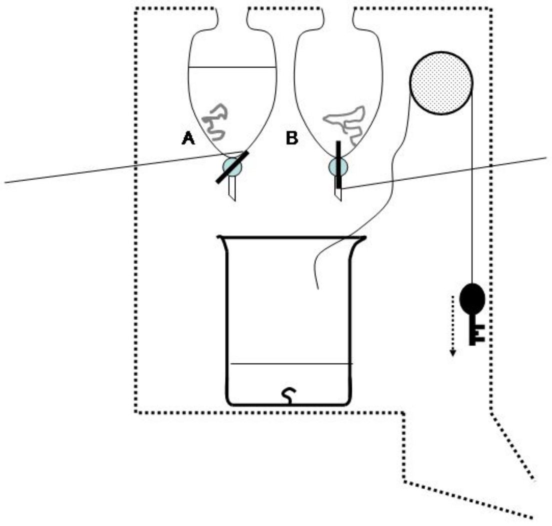
Answer:

Only few metals exist that are stable at open air and can burn; the most common is magnesium. Magnesium, as well as majority of other metals, reacts with acids, but it does not react with bases. In addition, few metals exist that react both with acids and with bases (these metals are called *amphoteric* metals). Two common amphoteric metals are aluminium and zinc (one of these metals was in the second jar; both metals cannot burn at open air). You know that either a funnel A or a funnel B contain acid, but you don't know which one. However, you know magnesium reacts with acids, but not with bases. That means you can make a simple test. You drop small piece of magnesium shavings into each funnel, and the reaction (bubble formation) will start just in one of them.



That will be an indication that funnel contains an acid. You pull the string attached to this funnel, the acid pours into the beaker, the steel rope starts to dissolve,





...and the key falls down.

BIOLOGY

5 points:

When you send your DNA to the sequencing service, like 23&me, they will often tell you that your ancestors from your mother's side have a particular heritage, and your ancestors from your father's side comes from a different heritage. How can they tell if genetic material came from your mother or father? Can this be determined for any genes in your genome?

Answer: Answer - X and Y chromosomes in males; mitochondrial DNA for females. For autosomes, cannot tell the origin.

10 points:

Under conditions of acute fear, the body prepares for fight or flight by activating the hypothalamic-pituitary-adrenal (HPA) axis, releasing a cascade of stress hormones such as adrenaline and cortisol. If chronically elevated through long-term emotional stress or sleep deprivation, stress (via cortisol) will actually cause people to increase their storage of abdominal fat. Likewise, people with a larger body-fat percentage generally produce more cortisol. What is the mechanism by which cortisol and fat-production are related? And how does this mechanism aid the fight or flight response?

Answer: Cortisol releases glucose from fat into the bloodstream, which allows for the energy required to fight or flee; people with more fat reduce more glucose, which is necessary since they have a larger mass to mobilize. Chronically-elevated cortisol stimulates appetite for carbs to make available more glucose, but emotional stress and sleep deprivation--unlike the fight or flight response for which cortisol's function evolved--do not consume glucose. Therefore, people tend to consume more food under these conditions, and to preferentially store it in the abdomen, where there are more cortisol receptors.

COMPUTER SCIENCE

- You can write and compile your code here: http://www.tutorialspoint.com/codingground.htm
- Your program should be written in C, C++, Java, or Python
- Any input data specified in the problem should be supplied as user input, not hard-coded into the text of the program.
- Please make sure that the code compiles and runs on http://www.tutorialspoint.com/codingground.htm before submitting it.
- Submit the problem in a plain text file, such as .txt, .dat, etc.
  No .pdf, .doc, .docx, etc!

5 points:

You are given a string of letters (ex: "asdkdjdhddjskqjjqoiu"). The string is entered interactively from the terminal. The task is to output the longest palindrome (a string that reads the same right to left and left to right) that you can form by rearranging (and/or removing some of) the letters in the string.

For example, if the input is: *aabbccddefg*

A valid output is: *abcdedcba*

Another valid output is: *bcdagadcb*

Solution:

```
word = raw_input("Type in word to palindrome:")
freqs = {}
for char in word:
    if char in freqs:
        freqs[char] += 1
    else:
        freqs[char] = 1
center_letter = None
palindrome_left_half = ''
for letter in freqs:
            if freqs[letter] % 2 != 0:
                        if center_letter == None:
                                    center_letter = letter
```

```
                palindrome_left_half += letter*((freqs[letter] - 1)/2)
        else:
                palindrome_left_half += letter*(freqs[letter]/2)
#reverse string
palindrome_right_half = palindrome_left_half[::-1]
#put the palindrome together
palindrome = palindrome_left_half
if center_letter != None:
        palindrome += center_letter
palindrome += palindrome_right_half

print "The palindrome is: " + palindrome
```

10 points:

N pirates found a treasure chest. There are M individual pieces inside (gold coins, silver medallions, pearls, diamonds, etc), and the pieces do not necessarily have the same value (for example, a pearl costs 1000 times less than a diamond). The pirates need to distribute the treasures as fairly as possible. Write a program to help them.

On the input, you receive the number of pirates, number of pieces of treasure, and the value of each piece:

N = 5, M = 1000, values = [10, 1, 1004, 12, 7, 34, 234 ...]

Feel free to type all values of treasure pieces directly into your program instead of reading them from user input or a file.

[Hint: begin by deciding for yourself: How do you define "fair"?]

Solution:

Like many things in life this problem is (intentionally) ambiguous. So first, let's make a few assumptions. What is fair? Many people consider a lottery to be fair. After all, everybody has an equal chance of winning. In this case we can come up with a very simple algorithm. Let's put all the pirates randomly in line and take 1 artifact at a time from the treasure chest to give the pirates sequentially. After all items have been distributed, each pirate would have a certain amount of treasure. These values most likely will not be equal for all pirates but since we chose them randomly we can claim this to be fair. Let's assume that the treasure artifacts can be represented by the natural numbers.

Let's express this in pseudo-code.

```
Given: N - number of pirates,
       M - number of treasure artifacts,
       values[1..M] - treasure artifacts
Goal: find piratesPossession[1..N] - each element is a pirate's heap of goods

currentPirate := 1                          // start from the 1st pirate; we assume the pirates are
                                            // given a random sequence number
for each value from values                  // loop through all treasure pieces
                                            // (we didn't do any sorting of values, so we assume
                                            // the artifacts are there in random order)
  piratesPossession[currentPirate].add(value) // give this pirate a treasure artifact
  currentPirate := currentPirate+1          // select next pirate
  if currentPirate > N                      // if all pirates were given a piece
    currentPirate := 1                      // restart from the 1st pirate
  end if
end for
```

Now, let's express this in Python. See *pirateLottery.py*.
And in java: *pirateLottery.java*.

Run the programs, change the input and measure the time it takes to run.

Discussion.

It's always interesteting to know how much time it'll take to run a program. Well, we can just measure it. In our case, fix "values" and piratesCount and run the program. But what will happen with run time if we double piratesCount? Or double the number of treasure pieces? Of course, we can change the input and run the program again. But what happens if the run time takes too long? Maybe we can estimate it without actually running the program. Let's see. In our case the only input we've got is values and piratesCount. Try to change piratesCount, for example, 1, 2, 10, 100, 1000000. Does the time change significantly? Why?
Introduce a new variable treasureCount (M in the problem statement). Generate random integers "values" (the treasure artifacts), from 1 to treasureCount. Does the run time changes for different "values" for the same treasureCount?
Now, try different treasureCount: 10, 100, 1000, 1000000. How the run time changes? Can you see what the dependence of the run time on treasureCount looks like? Why? Take a look where we use "values". We have a loop that iterates (goes over) all the "values" 1 time. So, if we double treasureCount (number of elements in "values") the loop we'll become twice longer to run. The rest of the program almost doesn't change.

Thus, we can estimate that this program will run 10 times longer if we make treasureCount 10 times bigger. Such estimates are called "big O notation" and is written O(N), which basically means "order of" your input. In our case the algorithm complexity is O(N) where N means size of your input. As we discussed it's treasureCount. So, for us if we take twice as much input, i.e. 2*N then the run time will be "order of 2*N", i.e. twice as long as if we just took N.

Now, let's go back to our assumption of faireness. What if we take another approach? Let's say we consider fair if everybody gets the same dollar amount. Well, that's not always possible. Consider a treasure chest with 3 coins costing 2, 3 and 4 dollars, and only 2 pirates. The best solution in this case is {2, 3} and {4}. This is the closest to the true average of 4.5. So, let's try to define fair as the minimum deviation from the true average, i.e. we try to give each pirate almost the same amount, and we'll try to make the difference as little as possible.
This is actually a very difficult problem.

First, let's try the brute force approach. We'll try all possible variants and select the best one. For example, given the treasures { 2, 3, 4 } and 2 pirates, the possible solutions are:

```
1st pirate   2nd pirate
2,3,4 => 9      -  => 0
3,4   => 7      2  => 2
2,4   => 6      3  => 3
2,3   => 5      4  => 4   \ these two are
4     => 4    2,3  => 5   / the best solutions
3     => 3    2,4  => 6
2     => 2    3,4  => 7
-     => 0   2,3,4 => 9
```

In other words, let's give the 1st piece of treasure to the 1st pirate, then to the 2nd one, etc. For each case let's give the 2nd artifact to the 1st guy, then to the 2nd one, etc. For the above example it looks like

```
1st artifact ->      /                    \
                 1st pirate            2nd pirate
2nd artifact ->    /      \            /       \
                 1st guy  2nd guy   1st guy  2nd guy
3rd artifact ->   /  \     /  \      /  \      / \
                 1st 2nd  1st 2nd   1st 2nd  1st 2nd
```

Such structures are called Trees because they look like a tree: starting from a single trunk and then branching more and more down to the leaves. Albeit the picture is upside down. The place of splitting is called a Node. The single topmost node is called the Root. The "end" nodes are called terminal ones or leaves.

In our case we can look over all combinations (of artifacts and pirates) by traversing our tree in all possible paths.
Look at *BruteForce.java*, *Tree.java* or *BruteForce.py*.
Now, play with piratesCount (N) and treasureCount (M). See how quickly you can make this problem practically uncomputable. Let's see what our complexity is. We said that each artifact goes to each pirate. So, the 1st level of our tree has N nodes; the 2nd - N*N; the 3rd - N*N*N, etc. Thus, the last level will have N^M nodes. This is the number of leaves and it represents how many different allocations we can make for M artifacts between N pirates. So, we've got O(N^M). For example, for 5 pirates and 100 artifacts it's about 10^70, quite a number. For your 3 GHz CPU even if we assume all the operations can be computed in 3 CPU cycles (which is far from the truth), it'll still take 10^61 sec. That is many times over the age of our universe!

So what can we do? Let's be smarter. Instead of blindly trying all the cases let's try to foresee a better one. Here's a better algorithm. Let's sort from largest to smallest artifacts (by value). Take the 1st one (the largest) and give it to the 1st pirate. Take the 2nd one and give it to the next pirate, and so on. Once everybody gets a piece let's give the next one to the pirate who has the least total value. Then continue in this fashion (giving the next piece to the "poorest" pirate). This algorithm doesn't always find the best solution. For example, given {8,7,6,5,4} it'll distribute to 2 pirates like this: {8,5,4} and {7,6}. But it gives a reasonably good solution. And unlike the previous solution it works really fast. It scales like O(M), i.e. linearly with the number of treasure artifacts. This algorithm is called "greedy". The reason for this is at each step it "grabs" the best next step disregarding the fact that in the long run you may find a bigger benefit. Sounds familiar? ;-)
See *greedy.java* and *greedy.py*.

Solutions in Java:

PirateLotteryJ.java:
```
import java.util.ArrayList;

public class PirateLotteryJ {
  // our input
  private int[] values; // values of treasure pieces
  private int   piratesCount;
  // our output
  private ArrayList<ArrayList<Integer>> piratesPossessions;
  public void getInput() {
    values       = new int[] {4, 5, 6, 7, 8};
    piratesCount = 2;

    // add code here to randomly shuffle values

    System.out.printf("number of pirates = %d\n", piratesCount);
    System.out.printf("treasure chest: ");
    for(int i=0; i<values.length; i++)
      System.out.printf("%d, ", values[i]);
    System.out.printf("\n");
  }

  public void printOutput() {
    System.out.printf("pirates' treasures:\n");
    for(ArrayList<Integer> piratePossessions : piratesPossessions) {
      for(int item : piratePossessions) {
        System.out.printf("%d,  ", item);
      }
      System.out.printf("\n");
    }
  }

  public void test() {
    getInput();

    // initialize piratesPossessions
    piratesPossessions = new ArrayList<ArrayList<Integer>>(); // list of lists
(based on array)
    for(int i=0; i<piratesCount; i++) {
      piratesPossessions.add(new ArrayList<Integer>());
    }

    int currentPirate = 0;
    for(int value : values) {
      piratesPossessions.get(currentPirate).add(value);
      currentPirate++;
      if(currentPirate >= piratesCount)
        currentPirate = 0;
    }

    printOutput();
  }

  public static void main(String[] args) {
    long start = System.currentTimeMillis();
    PirateLotteryJ test = new PirateLotteryJ();
    test.test();

    long end = System.currentTimeMillis();
  }
}
```

BruteForce.java:

```
import java.util.ArrayList;

public class BruteForce {
  // our input
  private int[] values; // values of treasure pieces
  private int   piratesCount;
  // our output
  private ArrayList<ArrayList<Integer>> piratesPossessions;

  public void getInput() {
    piratesCount = 2;
    values       = new int[] {4,5,6,7,8};
    /*
    values = new int[10];
    for(int i=0; i<values.length; i++)
      values[i] = (int)(Math.random()*1000+1);
    */

    // add code here to randomly shuffle values

    System.out.printf("number of pirates = %d\n", piratesCount);
    System.out.printf("treasure chest: ");
    for(int i=0; i<values.length; i++)
      System.out.printf("%d, ", values[i]);
    System.out.printf("\n");
  }

  public void test() {
    getInput();

    // initialize piratesPossessions
    piratesPossessions = new ArrayList<ArrayList<Integer>>(); // list of lists
(based on array)
    for(int i=0; i<piratesCount; i++) {
      piratesPossessions.add(new ArrayList<Integer>());
    }

    Tree tree = new Tree(piratesCount);
    tree.add(values, 0); // populate the tree starting from 1st artifact
    // walk the tree and populate piratesPossessions
    tree.traverse(piratesPossessions);
  }

  public static void main(String[] args) {
    long start = System.currentTimeMillis();

    BruteForce test = new BruteForce();
    test.test();

    long end = System.currentTimeMillis();
    System.out.println("end.");
  }
}
```

Tree.java:

```
import java.util.ArrayList;

class Node {
  private int value; // treasure artifact
  private ArrayList<Node> children;

  public Node(int value) {
    this.value = value;
  }

  public void addChildren(int value, int branchCount) {
    children = new ArrayList<Node>(branchCount);
    for(int i=0; i< branchCount; i++) {
      children.add(new Node(value));
    }
  }

  public int getValue() {
    return value;
  }

  public ArrayList<Node> getChildren() {
    return children;
  }
}

public class Tree {
  private Node root;
  private int  branchCount;
  private double average;
  private int nVariants;
  public Tree(int branchCount) {
    this.branchCount = branchCount;
    root = new Node(-1);
    nVariants = 0;
  }

  public void add(int[] values, int index) {
    // calculate the average
    average = 0;
    for(int i=0; i<values.length; i++)
      average += values[i];
    average /= branchCount;

    add(values, index, root);
  }

  private void add(int[] values, int index, Node node) {
    if(index >= values.length)                          // we processed all
artifacts
      return;                                           // nothing left to do
    node.addChildren(values[index], branchCount);      // we'll try all the
variants,
    ArrayList<Node> children = node.getChildren();
    for(Node child : children) {                        // i.e. give this "value"
to all pirates
      add(values, index+1, child);                      // we call ourselves with
the next item. This is called recursion! A very important concept!
    }
  }

  /**
   * walk the tree collecting artifacts into pirates' bags
   * @param [IN/OUT] piratesPossessions
   */
  public void traverse(ArrayList<ArrayList<Integer>> piratesPossessions) {
    traverse(piratesPossessions, root);
    System.out.printf("number of variants = %d\n", nVariants);
  }

  public void traverse(ArrayList<ArrayList<Integer>> piratesPossessions, Node
node) {
    ArrayList<Node> children = node.getChildren();
    if(children == null) {
      nVariants++;
      printOutput(piratesPossessions);
      // reset piratesPossessions
      int piratesCount = piratesPossessions.size();
      piratesPossessions = new ArrayList<ArrayList<Integer>>();
      for(int i=0; i<piratesCount; i++) {
        piratesPossessions.add(new ArrayList<Integer>());
      }
      return;
    }
    for(int i=0; i<children.size(); i++) {
      Node child = children.get(i);
      int value = child.getValue();
      //System.out.printf("%d goes to %d\n", value, i);
      ArrayList<Integer> piratePossessions = piratesPossessions.get(i);
      piratePossessions.add(value);
      traverse(piratesPossessions, child); // recursion again

      piratePossessions.remove(piratePossessions.size()-1); // delete the last
added item
    }
  }

  public void printOutput(ArrayList<ArrayList<Integer>> piratesPossessions) {
    double delta = 0; // total discrepancy from the average

    System.out.printf("pirates' treasures: ");
    for(int i=0; i<piratesPossessions.size(); i++) {
      double sum = 0;
      if(i > 0)
        System.out.printf("; ");
      System.out.printf("{");
      ArrayList<Integer> piratePossessions = piratesPossessions.get(i);
      for(int j=0; j<piratePossessions.size(); j++) {
        int item = piratePossessions.get(j);
        if(j > 0)
          System.out.printf(", ");
        System.out.printf("%d", item);
        sum += item;
      }
      delta += Math.abs(sum - average);
      System.out.printf("}");
    }
    System.out.printf("   delta=%.1f%s\n", delta, (delta<=1) ? " *** best
solution" : "");
  }
}
```

Greedy.java:

```
import java.util.ArrayList;
import java.util.Collections;

public class Greedy {
```

```
// our input
private int[] values; // values of treasure pieces
private int   piratesCount;
// our output
private ArrayList<ArrayList<Integer>> piratesPossessions;

private double average;
private ArrayList<Integer> sums;

public void getInput() {
  piratesCount = 2;
  values       = new int[] {4,5,6,7,8};
  /*
  values = new int[10];
  for(int i=0; i<values.length; i++)
    values[i] = (int)(Math.random()*1000+1);
  */

  // add code here to randomly shuffle values

  System.out.printf("number of pirates = %d\n", piratesCount);
  System.out.printf("treasure chest: ");
  for(int i=0; i<values.length; i++)
    System.out.printf("%d, ", values[i]);
  System.out.printf("\n");

  // calculate the average
  average = 0;
  for(int i=0; i<values.length; i++)
    average += values[i];
  average /= piratesCount;
}

public void printOutput() {
  double delta = 0; // total discrepancy from the average

  System.out.printf("pirates' treasures: ");
  for(int i=0; i<piratesPossessions.size(); i++) {
    double sum = 0;
    if(i > 0)
      System.out.printf("; ");
    System.out.printf("{");
    ArrayList<Integer> piratePossessions = piratesPossessions.get(i);
    for(int j=0; j<piratePossessions.size(); j++) {
      int item = piratePossessions.get(j);
      if(j > 0)
        System.out.printf(", ");
      System.out.printf("%d", item);
```

```
        sum += item;
      }
      delta += Math.abs(sum - average);
      System.out.printf("}");
    }
    System.out.printf("   delta=%.1f%s\n", delta, (delta<=1) ? " *** best
solution" : "");
  }

  public void test() {
    getInput();

    // initialize piratesPossessions
    piratesPossessions = new ArrayList<ArrayList<Integer>>(); // list of lists
(based on array)
    sums = new ArrayList<>();
    for(int i=0; i<piratesCount; i++) {
      piratesPossessions.add(new ArrayList<Integer>());
      sums.add(0);
    }

    // sort values in reverse order; quite cumbersome in java
    ArrayList<Integer> a = new ArrayList<Integer>();
    for(int i=0; i<values.length; i++)
      a.add(values[i]);
    Collections.sort(a, Collections.reverseOrder());
    for(int i=0; i<values.length; i++)
      values[i] = a.get(i);

    // distribute artifacts
    for(int item : values) {
      int minIndex = sums.indexOf(Collections.min(sums)); // find the poorest
pirate
      piratesPossessions.get(minIndex).add(item);         // assign the new
artifact to him
      int newSum = sums.get(minIndex) + item;             // calculate his new
sum
      sums.set(minIndex, newSum);                         // update it
    }

    printOutput();
  }

  public static void main(String[] args) {
    long start = System.currentTimeMillis();

    Greedy test = new Greedy();
    test.test();
    long end = System.currentTimeMillis();
    System.out.println("end.");
  }
}
```

Solutions in Python:

PirateLottery.py:

```
def get_input():
  values = (4, 5, 6, 7, 8) # values of treasure pieces # () means tuples which
are like lists but are immutable
  N = 2                    # number of pirates
  # M = len(values)        # we don't need M explicitly; we can get it from
values length

  # add code here to randomly shuffle values

  print("number of pirates = {}".format(N))
  print("treasure chest: {}".format(values))
  return N, values

def print_output(pirates_possessions):
  print("pirates\' treasures:")
  for pirate_possessions in pirates_possessions:
    for item in pirate_possessions:
      print("{}, ".format(item), end=' ')
    print("")
  pass

def test():
  N, values = get_input()

  # initialize pirates_possessions
  pirates_possessions = [] # list of lists
  for i in range(0, N):
    a = []
    pirates_possessions.append(a)

  current_pirate = 0 # python indices start from 0
  for value in values:
    pirates_possessions[current_pirate].append(value)
    current_pirate += 1
    if current_pirate >= N:
      current_pirate = 0
  pass

  print_output(pirates_possessions)
```

```
if __name__ == "__main__":
  # test()
  import timeit
  print(timeit.timeit("test()", setup="from __main__ import test", number=1))
```

BruteForce.py:

```
class Node:
  def __init__(self, value, children = []):
    self.value    = value
    self.children = children

  def add_children(self, value, branch_count):
    self.children = [Node(value) for _ in range(branch_count)]

class Tree:
  def __init__(self, branch_count):
    self.branch_count = branch_count
    self.root         = Node(-1)
    self.average      = 0
    self.n_variants   = 0

  def add(self, values, index):
    self.average = sum(values) / self.branch_count

    self.add2(values, index, self.root)

  def add2(self, values, index, node):
    if index >= len(values):                             # we processed all
artifacts
      return                                             # nothing left to do
    node.add_children(values[index], self.branch_count)  # we'll try all the
variants,
    for child in node.children:                          # i.e. give this
"value" to all pirates
      self.add2(values, index+1, child)                  # we call ourselves
with the next item. This is called recursion! A very important concept!

  def traverse(self, pirates_possessions):
    self.traverse2(pirates_possessions, self.root);
    print("number of variants = {}".format(self.n_variants))

  def traverse2(self, pirates_possessions, node):
    if not node.children:
      self.n_variants += 1
      print_output(pirates_possessions, self.average)
      # reset piratesPossessions
      pirates_count = len(pirates_possessions)
      pirates_possessions = []
      for _ in range(pirates_count):
        pirates_possessions.append([])
      return

    for i in range(len(node.children)):
      child = node.children[i]
      value = child.value
      #print("{} goes to {}".format(value, i))
      pirate_possessions = pirates_possessions[i]
      pirate_possessions.append(value)
      self.traverse2(pirates_possessions, child) # recursion again

      del pirate_possessions[-1] # delete the last added item
  pass

##########

def get_input():
  values = (4,5,6,7,8)     # values of treasure pieces # () means tuples which
are like lists but are immutable
  N = 2                    # number of pirates
  # M = len(values)        # we don't need M explicitly; we can get it from
values length

  # add code here to randomly shuffle values

  print("number of pirates = {}".format(N))
  print("treasure chest: {}".format(values))
  return N, values

def print_output(pirates_possessions, average):
  delta = 0 # total discrepancy from the average
  print("pirates\' treasures: ", end='')
  for i in range(len(pirates_possessions)):
    pirate_possessions = pirates_possessions[i]
    summ = 0
    if i > 0:
      print("; ", end='')
    print("{", end='')
    for j in range(len(pirate_possessions)):
      item = pirate_possessions[j]
      summ += item
      if j > 0:
        print(", ", end='')
      print("{}, ".format(item), end='')
    print("}", end='')
    delta += abs(summ - average);
  print("   delta={}{}".format(delta, " *** best solution" if delta<=1 else ""))

def test():
  N, values = get_input()

  # initialize pirates_possessions
  pirates_possessions = [] # list of lists
  for _ in range(N):
    pirates_possessions.append([])

  tree = Tree(N)
  tree.add(values, 0); # populate the tree starting from 1st artifact

  # walk the tree and populate piratesPossessions
  tree.traverse(pirates_possessions)

if __name__ == "__main__":
  # test()
  import timeit
  tm = timeit.timeit("test()", setup="from __main__ import test", number=1)
  print("it took {} sec to run the program".format(tm))
```

Greedy.py:

```
def get_input():
  values = (4,5,6,7,8)      # values of treasure pieces # () means tuples which
are like lists but are immutable
  N = 2                     # number of pirates
  # M = len(values)         # we don't need M explicitly; we can get it from
values length

  # add code here to randomly shuffle values

  average = sum(values) / N

  print("number of pirates = {}".format(N))
  print("treasure chest: {}".format(values))
  return N, values, average

def print_output(pirates_possessions, average):
  delta = 0 # total discrepancy from the average
  print("pirates\' treasures: ", end='')
  for i in range(len(pirates_possessions)):
    pirate_possessions = pirates_possessions[i]
    summ = 0
    if i > 0:
      print("; ", end='')
    print("{", end='')
    for j in range(len(pirate_possessions)):
      item = pirate_possessions[j]
      summ += item
      #if j > 0:
      #  print(", ", end='')
      print("{}, ".format(item), end='')
    print("}", end='')
    delta += abs(summ - average);
  print("   delta={}{}".format(delta, " *** best solution" if delta<=1 else ""))

def test():
  N, values, average = get_input()

  # initialize pirates_possessions
  pirates_possessions = [] # list of lists
  sums = []
  for _ in range(N):
    pirates_possessions.append([])
    sums.append(0)

  # sort values in reverse order
  values = sorted(values, reverse=True)

  # distribute artifacts
  for item in values:
    min_index = sums.index(min(sums))            # find the poorest pirate
    pirates_possessions[min_index].append(item)  # assign the new artifact to
him
    sums[min_index] += item                      # update his sum

  print_output(pirates_possessions, average)

if __name__ == "__main__":
  # test()
  import timeit
  tm = timeit.timeit("test()", setup="from __main__ import test", number=1)
  print("it took {} sec to run the program".format(tm))
```