CMSI 282 - Homework 3

Janine Leano and Justin Sanny

April 1st 2015

1.
```python
# Help taken from Java implementation of Bozo-Sort:
# http://www.dreamincode.net
# /forums/blog/114/entry-595-bozosort-definitive-c-c%23-vbnet-java-php/

import random

def BozoSort(array):
        while (not is_sorted(array)):
                index1 = random.randint(0, len(array) - 1)
                index2 = random.randint(0, len(array) - 1)

                array[index1], array[index2] = array[index2], array[index1]
        return array;

def is_sorted(array):
        for i in range(0, len(array) - 1):
                if (array[i] > array[i + 1]):
                        return False
        return True
```

Given array [6, 10, 123, 1, 3, 23, 4, 90, 2, 53, 44]

Table length	**1st time**	**2nd**	**3rd**	**4th**	**5th**	**Average runtime**
2	0.1s	...	...	...	...	0.01s - 0.1s
3	0.07s	...	...	...	...	0.01s - 0.1s
4	0.1s	...	...	...	...	0.01s - 0.1s
5	0.1s	...	...	...	...	0.01s - 0.1s
6	0.13s	0.12s	0.12s	0.13s	0.12s	0.122s
7	0.4s	0.18s	0.2s	0.13s	0.31s	0.244s
8	0.3s	0.2s	1.6s	0.4s	0.4s	0.58s
9	5.1s	1.6s	4.3s	15.3s	3.1s	5.88s
10	228.2s	24.1s	3.8s	44.0s	76.9s	75.4s

2.
```python
def AutoKeyVigenere (plain_text, old_key):
        if (len(plain_text) < len(old_key)):
                raise ValueError("Key length must be <= to message length")
        key = list(old_key.lower())
        plain_text = list(plain_text.lower())
        ciphertext = []
        for i in range(len(plain_text) - len(old_key)):
                key.append(plain_text[i])
        for j in range(len(key)):
```

```
            code = ((ord(plain_text[j]) + ord(key[j]) - 64) % 26) + 65
            ciphertext.append((chr(code)))
        return ciphertext

#ciphertext = "JUKVKVOZCOHMDSFUMZCTNHZVQPFOJWCOOTWYVVBHUBYHYSWFU"
def main():
        a = raw_input("Enter message to be encrypted: ")
        b = raw_input("Enter key: ")
        cipher = AutoKeyVigenere(a, b)
        string = ""
        for k in range(len(cipher)):
                string += cipher[k]
        print("Encryption: " + string)

main()
```

3. Let us change our traditional attitude to the construction of programs. Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.

4. Computer science is no more about computers than astronomy is about telescopes.

5. To find the private key, we have to solve for d, where d = modular inverse of e relative to (p -1)(q - 1)
   Given: n = 729880581317 and e = 5. n = pq, where p and q are some primes
   p can be computed by testing all primes (going down from the square root of n) to see the largest possible value that is a factor of n.
   q = n / p. So far, p = 822893 and q = 886969
   We assign Φ or (p-1)(q-1), which in this case is equal to 729878871456, to variable r.
   We look for a candidate value K which is equal to1 mod r and must include e as one of K's factors.
   Here, we chose K = 2919515485825
   We divide K by e, which will yield us d. d = K/e = 583903097165
   Private key is (n, d) or (729880581317, 583903097165)

6. Problem 1.45
   a.) We need digital signatures because it allows us to authenticate the encrypted message and ensures its integrity is preserved and that it came from our desired user.
   b.) n = pq, where p and q are some primes.
   Since e and d are multiplicative inverses and $(M^d)^e \equiv M \pmod{n}$ then it is true that $(M^d)^e = M^{de}$.
   Since by definition, e and d are inverses, then M is just raised to 1. So $M \equiv M \pmod{n}$.
   c.) Given: p = 101, q = 61, we calculate: n = 6161, e = 19 where n = pq and e is relatively prime to (p - 1)(q - 1). d can be calculated using the same method as Problem 5. So d = 1579
   Let m = "JANINE", and we will use ASCII to map strings to integers in [0, n - 1]. The first letter is 74. Let m = 74.
   The signature for the first number (m0) is $m0 \bmod n$ or $74^{19} \bmod 6161 = 5015$. To verify, we check if $5015^{19} \bmod 6161 = 74$.
   d.) Given n = 391 and e = 17, p = 17 and q = 23 by factoring. So $(p-1)(q-1) = 16 \times 22 = 352$.
   d = $(e)^{-1} \bmod 352 = 145$. Answer is 145.
   We verify that $145 \times 17 = 2465 \equiv 1 \pmod{352}$.

7. Problem 1.46
   a.) Encrypted message = $M^e \bmod N$. If Eve asks Bob to sign it for her, she can obtain his private key (d). To get M, she can just compute $(M^e)^d \bmod N$.
   b.) Eve can use a value relatively prime to N (call it x). Eve can ask Bob to sign $M^e \times x^e \bmod N$. She can get M by multiplying x^{-1} mod N with Mx mod N (which was obtained – $M^e \times x^e mod N = (Mx)^{ed} \bmod N = Mx \bmod N$).

8. Problem 2.4
   Using the Master Theorem – Algorithm A is $5T(n/2) + n. \ln 5 > 1$. Running time is $O(n^{2.322\cdots})$
   Algorithm B is $2T(n-1) + c$ (c is some constant). $2 * 2T(n-1-1) + c + c$. Pattern is 2^n. So $O(2^n)$.
   Algorithm C is $9(n/3) + n^2 \log_3 9 = 2$. So $O((n^2) logn)$.
   Algorithm A is preferable.

9. Problem 2.12
   source: http://www.cs.rpi.edu/ goldsd/docs/spring2013-csci2300/hw1-solns.txt
   $T(n) = 2 * T(n/2) + 1$ with $n = 2^k$
   Solving for T(n) in [1...n], we find that the computations follow the pattern of n-1. Since $n = 2^k$, $T(2^k) = 2 * T(2^{k-1}) + 1 = 2^k - 1 = n - 1 = \Theta(n)$

10. Problem 2.23
    a.) Algorithm: Split array (A) into two (A1 and A2) of n/2 (n = length of A). Assign boolean value (B1 and B2) on whether subarrays have in fact a majority element. If B1 or B2 are true, assign majority element to integer ME1 and ME2 respectively. If B1 and B2 are true and ME1 == ME2, return ME1. If B1 and B2 are false, then there is no majority element (return False). If B1 and B2 are True and False (or vice versa), assign an integer value (C1 and C2) that keeps count on the occurrences that ME1 appears in A1 and A2 and ME2 appears in A1 and A2. If either sub-majority appears more than n/2, return that sub-majority. Otherwise, return False.
    b.) source: http://stackoverflow.com/questions/4325200/find-majority-element-in-array
    Using Boyer's algorithm, we can find the majority element in O(n) time. The algorithm is as follows:

```
int findMajorityElement(int* arr, int size) {
    int count = 0, i, majorityElement;
    for (i = 0; i < size; i++) {
        if (count == 0)
            majorityElement = arr[i];
        if (arr[i] == majorityElement)
            count++;
        else
            count--;
    }
    count = 0;
    for (i = 0; i < size; i++)
        if (arr[i] == majorityElement)
            count++;
    if (count > size/2)
        return majorityElement;
    return -1;
}
```

11. Problem 3.2(a)
    tree edge - (A, B), (B, C), (C, D), (D, H), (H, G), (G, F), (F, E)
    forward edge - (A, F), (B, E)
    back edge - (D, B), (E, D), (E, G), (F, G)
    cross edge - none

    vertex - (pre, post)
    A - (1, 16)
    B - (2, 15)
    C - (3, 14)
    D - (4, 13)
    H - (5, 12)
    G - (6, 11)

F - (7, 10)
E - (8, 9)

12. Problem 3.8

a. The graph is a directed graph in the form of a depth-first tree. Define each node by (a0, a1, a2) where a0 is at most or equal to 10. The question is whether there exists a path between (0, 7, 4) and (a0, 2, a2) or (a0, a1, 2) so long as the variable values are acceptable for their respective a-coordinates.
b. The algorithm is depth-first search.
c. The answer is $(0,7,4) \rightarrow (7,0,4) \rightarrow (10,0,1) \rightarrow (10,1,0) \rightarrow (6,1,4) \rightarrow (6,5,0) \rightarrow (2,5,4) \rightarrow (2,7,2)$