SI 335 Spring 2015: Problem Set 2

Due: Wednesday, February 18

Your scheduled presentation time:

Group member:

Group member:

Group member:

Group member:

Instructions: Review the course honor policy: you may not use any human sources outside your group, and must document anything you used that's not on the course webpage.

This cover sheet must be the front page of what you hand in. Use separate paper for the your written solutions outline and make sure they are neatly done and in order. Staple the entire packet together.

Comments or suggestions about this problem set:

Comments or suggestions about the course so far:

Citations (be specific about websites):

Grading rubric:

A: Solution meets the stated requirements and is completely correct. Presentation is clear, confident, and concise.
B: The main idea of the solution is correct, and the presentation was fairly clear. There may be a few small mistakes in the solution, or some faltering or missteps in the explanation.
C: The solution is acceptable, but there are significant flaws or differences from the stated requirements. Group members have difficulty explaining or analyzing their proposed solution.
D: Group members fail to present a solution that correctly solves the problem. However, there is clear evidence of significant work and progess towards a solution.
F: Little to no evidence of progress towards understanding the problem or producing a correct solution.

Problem	Final assessment
1	
2	
3	
4	

1 Popularity contest

An ice cream store recently sampled many flavors for its top customers. Each customer then voted for their favorite. Now the ice cream store wants to know which flavors were really popular and received more than one-third of the votes.

So here's your task: given a list of numbers `A` of length n, find all numbers `x` that occur more than $n/3$ times in A. These are the popular numbers.

Note that there can be 0, 1, or 2 popular numbers. (It's impossible to have three things that all occur *more* than one-third of the time.)

a) The first, basic algorithm you think of is the following:

```
def popular_basic(A):
    '''Returns all elements that occur more than n/3 times'''
    apop = []
    for x in A:
        if count(A, x) > len(A)/3:
            apop.append(x)
    return apop

def count(A, x):
    '''Returns the number of times x occurs in A'''
    c = 0
    for a in A:
        if a == x:
            c += 1
    return c
```

What's the running time of this `popular_basic` algorithm, in terms of n?

b) Here's a cleverer algorithm for the same thing:

```
def popular_better(A):
    '''Returns all elements that occur more than n/3 times'''
    if len(A) <= 1:
        return A
    else:
        mid = len(A) // 2
        B = A[0 : mid]
        C = A[mid: len(A)]
        apop = []
        for x in popular_better(B):
            if count(A,x) > len(A)/3:
                apop.append(x)
        for x in popular_better(C):
            if x not in apop and count(A,x) > len(A)/3:
                apop.append(x)
        return apop
```

Convince me that this algorithm is still correct, i.e., it always returns all the popular elements in `A`.

c) Analyze the worst-case running time of `popular_better`, in terms of n, the length of `A`. I would like a big-Θ bound.

d) Suppose the list `A` were already sorted. Describe how that would allow you to find the popular numbers even faster, in $O(n)$ time.

e) **CHALLENGE**: Plus 1% to everyone in your section if you can come up with a $\Theta(n)$-time algorithm for this problem, even when `A` is not sorted. I promise that it's possible!

2 Tug of War

There are n Mids trying out for the varsity tug-of-war team. Some of them are strong and some of them are weak. For simplicity, assume that all the strong Mids have *exactly* the same strength, as do all the weak Mids. You can also assume that there is at least one strong Mid and at least one weak Mid. (In particular, n must be at least two.)

Your task is to determine who the strong Mids are, to decide who will be on the team. And the only tool you have to determine who is strong and weak is running *contests*. A contest involves pitting some Mids against some others in a tug-of-war, and the outcome can be either that one side wins, or the other side wins, or they tie. This pseudocode might help clarify:

```
# Calling this function represents a single contest.
def winner(group1, group2):
    if group1 wins:
        return group1
    elif group2 wins:
        return group2
    else:
        return 'tie'
```

There can be any number of Mids in any contest, but it should always be the same number on each side of the contest. The side with more strong Mids (or, equivalently, with fewer weak Mids) wins.

For example, here is an algorithm for $n = 3$:

```
def strongOf3(M0, M1, M2):
    w1 = winner({M0}, {M1})
    if w1 == 'tie':
        if winner({M1}, {M2}) == {M1}:
            return {M0, M1}
        else:
            return {M2}
    else:
        if winner(w1, {M2}) == 'tie':
            return w1 + {M2}
        else:
            return w1
```

An here's an algorithm for $n = 4$:

```
def strongOf4(M0, M1, M2, M3):
    w1 = winner({M0}, {M1})
    w2 = winner({M2}, {M3})
    if w1 == 'tie' and w2 == 'tie':
        return winner({M0,M1}, {M2,M3})
    elif w1 == 'tie':
        # w1 is a tie, but w2 is not
        if winner({M0}, w2) == 'tie':
            return {M0, M1} + w2
        else:
            return w2
    elif w2 == 'tie':
        # the opposite here; w1 is not a tie
        if winner({M2}, w1) == 'tie':
            return w1 + {M2, M3}
        else:
            return w1
    else:
        return w1 + w2
```

a) State the number of contests performed by `strongOf3` and by `strongOf4` in each of their worst cases.

b) Give a **lower bound** on the number of contests required to determine who the weak Mids are for an input of size n, in the worst case.

State your exact lower bound as a function of n, showing all your work. Then state what the asymptotic big-Ω bound is that results, simplified as much as possible.

For example of what I'm asking for, in class we showed that sorting requires at least $\lg(n!)$ comparisons (the exact bound), which is $\Omega(n \log n)$.

c) Give an algorithm for any n that determines who the strong Mids are. In describing your algorithms, you can call the Mids by their number like an array: `M[0]`, `M[1]`, ..., `M[n-1]`.

Analyze the number of contests that your algorithm performs in the worst case (NOT the number of primitive operations, just the number of contests).

d) Is your algorithm asymptotically optimal? Say why or why not. (Hint: there is an asymptotically optimal algorithm for this problem!)

e) **Ultra bonus**: An algorithm is *exactly optimal* (not just asymptotically optimal) if the number of contests is exactly equal to the lower bound, for all values of n. You need to either develop an exactly optimal algorithm for the tug-of-war problem, or prove that no such algorithm could possibly exist.

3 RSA Vulnerability

There is a website called `secretbook.com` that offers free accounts to anyone in the world, and allows its members to send secret (encrypted) messages to each other. The way it works is, every user gets assigned a public/private key pair when they sign up for their free account, and the website publishes everybody's public key along with their username and public profile. Then any member can post encrypted messages and files with anyone else's public key, so only that person can read the message.

Here's the problem: you notice that every public key issued by the website has *exactly the same* modulus n. The website owners thought this was OK, since they only issue the public key (e, n) and private key (d, n) to its users, never the prime factors p and q of n or any other information.

But you notice there is a vulnerability there. What is the vulnerability? Describe how an attacker could do something that shouldn't be allowed, more easily than they would be able to do otherwise. Say exactly what can be done, and how easy it would be to do it.

Note: For full credit, there is an attack that can be performed in polynomial-time against this system, if you are clever enough! But you can still earn partial credit by describing any vulnerability that makes the attacker's job easier than it would be normally if different moduli n were used for each account.

4 I am rich and you are NIST.

I have developed a brand new public-key cryptosystem called DAN and the government has agreed to adopt DAN as a new encryption standard to replace RSA. Your task is to make recommendations to the government on what key lengths to use with the DAN cryptosystem in various settings.

All that you know about the DAN cryptosystem is that it consists of four functions, KEYGEN, ENCRYPT, DECRYPT, and CRACK, which are the best ways the creator of DAN (me!) has come up with to do those four tasks. All of these functions have a running time which depends on the key length k as follows:

- KEYGEN runs in $O(k^4)$ time.
- ENCRYPT runs in $O(k \log k)$ time.
- DECRYPT runs in $O(k^2)$ time.
- CRACK runs in $O((1.1)^k)$ time.

You also have timing information on these algorithms, running on my laptop, using only a single CPU core, and with keylength $k = 100$:

- KEYGEN took 10 seconds with keylength $k = 100$.
- ENCRYPT took 30 seconds with keylength $k = 100$.
- DECRYPT took 1 second with keylength $k = 100$.
- CRACK took 20 hours with keylength $k = 100$.

Based on the information above, and any other research you do, decide on key length recommendations for the following scenarios. There is of course not a single correct answer; rather I am interested in your logical reasoning and how you came up with the lengths you recommend.

You might need to do some outside research on things like the relative speed of supercomputers, cell phones, and laptops, or the projected costs of computing in the future. That's great, but be sure to cite any sources that you used.

Feel free to be somewhat speculative, but don't expect to get credit for "cute" answers such as "key length 0, because I already installed a keylogger" or "tell the government that DAN is not properly vetted and shouldn't be used yet". You have to come up with plausible, numerical recommendations!

a) What keylength do you recommend for a smartphone app for anonymously sending pictures to a specified individual, which are deleted as soon as they are viewed? Assume that a new key is generated for each picture that is sent.

b) What keylength do you recommend for the security certificate of a comic book dealer's website in Arkansas? The total revenue over this website is 1 million dollars per year, and the same certificate will be used for 10 years before it expires.

c) What keylength do you recommend for nuclear weapons launch codes? Assume these are re-generated every day and old ones won't ever work again.