DD2448
Andreas Tarandi
890416-0317

Homework I

February 25, 2012

No study group

1

gcd(a, b) is all the common factors in a and b.

- The $b > a$ check ensures that a always is at least as big as b.
- b == 0 check returns a (gcd(n, 0) = 0)
- If b and a are both even, then 2 is a factor in both and thus also a factor in the gcd.
- If one of a or b but not both is even then 2 is not a factor in the gcd and the even operand may be divided with 2
- This is almost the same step as in the Euclidean algorithm, except that we only remove one b at a time (in the euclidean algorithm floor(a/b) b:s are removed each iteration.
  Taking gcb(a,b) = gcb(b, a-b) is valid since the difference between a and b must be a factor of the gcd.

The number of recursive calls are at most the number of factors in the biggest operand.

2

To find X we use the chinese remainder theorem. We have the system
$X \equiv a_1 mod N$
$X \equiv a_2 mod (N + 1)$

Which gives us $X \equiv a_1U_1 + a_2U_2 mod N(N + 1)$ (from the chinese remainder therem). From Euclid's extended algorithm we get $1 = aN + b(N - 1)$ which in this case is trivial, $a = -1$, $b = 1$. Where $U_1 = a * N$, $U_2 = b * (N - 1)$.

This gives us the result:
$U_1 = -8904160317$
$U_2 = 8904160318$
$X \equiv 123456789 * -8904160317 + 987654321 * 8904160318 \ mod\ N(N - 1)$
$X \equiv 7694953371471391965 \ mod\ N(N - 1)$

That is X = 7694953371471391965 (both a_1 and a_2 are less than N and (N+1)

3

For this radix sort would be a good choice:

```
        def sort(list_num)
            current_divisor = 1
            (0..Math.log(list.max)).each do |digit_num|
                sublists = [[], [], [], [], [], [], [], [], [], []]
                list.each do |num| # Iterate through, in order
                    digit = ( num / current_divisor ) % 10
                    # append to sublist (in order of appearance)
                    sublists[digit] << num
                end
# Merge lists:
                list = sublists.flatten
                # List is now sorted up to digit digit_num
                current_divisor*=10 # Increase divisor
            end

            list #Return list
        end
```

This will work since we sort be each digit and keep the order from the previous digits.

This algorithm has the complexity O(n*k) where k is log_{10} list.max. In this case the the magnitude of the numbers are n^{10}, which gives us O($n * log_{10}(n^{10}) = n * log_{10}n * 10$) = O($n * log_{10}n$) worth noticing here is that it is log_{10}, and not log_2 which is normaly intended when talking complexity. $log_{10}(n)$ is negligible since it's much smaller than n (ex $n = 1000000000$ gives $log_{10}(n) = 9$ The algorithm can therefore be concidered to run in linear time.

4

Here is a nice quick and dirty solution:

```
std::vector<int> sort(std::vector<int> list) {
    std::vector<int> out;
    std::map<int, int> set;
    std::vector<int>::iterator it;
    std::map<int,int>::iterator it2;
    for(it = list.begin(); it != list.end(); ++it) {
        ++set[*it];
```

```
    }
    for(it2 = set.begin(); it != set.end(); ++it) {
        for(int i = 0; i < it2->second; ++i) {
            out.push_back(it2->first);
        }
    }
    return out;
}
```

This is probably not what you expected here, but it does the job in the required time. We have two for-loops over all elements O(2n) = O(n) and the insertion (and the lookup if the element already exists, done in the same call) into the map takes O(log [number of elements in the map]) which is at most O(log(m)), this gives us O(n*log(m)) which is the required complexity.