CS310: Advanced Data Structures and Algorithmns

Assignment 1 – Solution

Thanks to past students for parts of this solution.

1. (a) SUM $= 2^1 + 2^2 + \ldots 2^{10} = 2046$. By special argument: This is binary 111 1111 1110. If we add 2 to it, it rolls over to 1000 0000 0000 $= 2^{11} = 2K = 2048$, so it must be 2046.

   By geometric series: SUM $= 2^1 + 2^2 + \ldots 2^{10} = 2 * (2^0 + \cdots + 2^9)$

   SUM $= a * (1 + r + r^2 + r^n) = a * (1 - r^{(n+1)})/(1 - r)$ sum formula where here r = 2, a=2, n=9 so SUM $= 2 * (1 - 2^{10})/(1 - 2) = 2 * (1K - 1) = 2046$

   (b) SUM $= 2/3 + 4/9 + 8/27 + 16/81 + ...$
   Need to subtract the "to infinity ..." part out.
   $3/2 * SUM = 1 + 2/3 + 8/27 + 16/81 + \ldots$ and therefore:
   3/2*SUM − SUM $= 1 + (2/3 + 8/27 + ... - 2/3 - 8/27 - \ldots)$. The part in parentheses cancels out, we're left with: 1/2SUM = 1. Thus, SUM = 2

2. A number N has $floor(\log_2(N)) + 1$ binary digits, where floor(x) denotes the largest integer not greater than x. So:

   $\text{floor}(\log_2(2^{100})) + 1 = \text{floor}(100) + 1 = 101$,
   $\text{floor}(\log_2(5^{100})) + 1 = \text{floor}(100 \log_2(5)) + 1 = 232 + 1 = 233$
   $\text{floor}(\log_2(10^{100})) + 1 = \text{floor}(100 \log_2(10)) + 1 = 332 + 1 = 333$.

   How are these answers related? $2^{100} * 5^{100} = 10^{100}$ and $101 + 233 = 334 = approx.333$

3. (a) We have $\log_B(N) = \log_2(N)/\log_2(B)$ by the change-of-base formula, for any base B.
   So $\log_b(N) = \log_2(N)/\log_2(b)$ – relate base b to base 2
   and $\log_a(N) = \log_2(N)/\log_2(a)$ – relate base a to base 2
   and thus $\log_b(N)/\log_a(N) = \log_2(a)/\log_2(b) = const.$

   (b) We have $\log_{10}(N) = \log_2(N)/\log_2(10)$, and $1/\log_2(10) = 0.3010$, so $\log_{10}(N) = 0.3010 * \log_2(N)$
   Number of decimal digits = ceiling($\log_{10}(N)$) (within 1 of $\log_{10}(N)$)
   Number of binary digits = ceiling($\log_2(N)$) (within 1 of $\log_2(N)$)
   So numbers in base 10 are about 3/10 length of the same numbers in base 2.

4. (a) Functions ranked in order of increasing growth rate. Most of these are easily decided by looking at the limit of the ratio as N grows. Numerical evidence isn't really needed.

   - $2/N = O(1/N)$ – does not grow at all, it shrinks as $N \to \infty$
   - $39 = O(1)$ – constant
   - sqrt(N) $= N^{0.5}$
   - N
   - $N \log\log N$. $\log\log N$ grows **very** slowly. It's just 10 when $N = 2^{1000}$.
   - $N \log N$. This and the next are tied
   - $N \log(N^2) = 2N \log N$
   - $N^{1.5}$

- N^2
- $N^2(\log N)$
- $N^2(\log N)^2$
- N^3
- $2^{(N/2)}$
- 2^N this is **not** a tie with $2^{(N/2)}$

(b) $\log N$ and $\log(N^2)$ are tied because $\log(N^2) = 2 * \log N$. $\log^2 N = \log N * \log N$ grows faster than $\log N$. You can divide the two and get $\log N >> const$.

5. (a) The outer loop is executed n times, and the inner loop, in the worst case, is also executed n times giving time complexity of $O(n^2)$.

   (b) In terms of big-O both functions are $O(N^2)$.

   (c) In mysterySum, we have the inner loop adding i tos i times, so adding $i * i$ in all.

```
int mysterySum(int n)
{
   int i, j, s=0;
   for(i=0; i<n; i++)
      s += i*i;
}
```

   That's n passes of O(1) complexity. So the time complexity is $O(n)$.

   (d) The result is $sum(i^2) = n(n+1)(2n+1)/6$. This formula is available in the cs220 textbook. Or find it by searching for "sum of squares"

6. Calculating power2

```
int power2(int n)
{
 if (n==0) return 1;
 return power2(n-1)+power2(n-1);
}
```

We have a double recursion here, so obviously the runtime is bad. The recurrance formula is $T(n) = C + 2 * T(n-1)$. So $T(n-1) = C + 2T(n-2)$, and together:$T(n) = C + 2 * (C + 2T(n-2)) = C(1+2) + 2^2 * T(n-2)$. Similarly $T(n) = C(1+2+4) + 2^3 * T(n-3)$, ... $T(n) = C(1+2+...+2^(n-1)) + 2^n * T(1) = C * 2^n + 2^n * T(1) = O(2^n)$. The factors of 2 accummulate so that at the k^{th} stage we have $2^k * C$. Overall we have n stages.

A linear time algorithm would be the following:

```
int power2(int n)
{
 if (n==0) return 1;
 return 2*power2(n-1);
}
```

We simply run the recursion once and multiply by 2... You can easily show that the recurrence formula is linear, since it's the same as the factorial function we showed in class: $T(n) = C + T(n-1)$. Another way: a simple for loop (maybe not such a small change, though):

```
int power2(int n)
{
   int power = 1;
   for (int i=0; i < n; i++) {
      power *=2;
   }
}
```

7. Code for combinational lock. Note there are no getters or setters for a, b, c, since the combination is "secret". To change the combination, you need to supply the old combination.

```
/*
 * CombinationLock.java
 * @author -- Ruchi Dubey
 * ruchi@cs.umb.edu
 */

class CombinationLock
{
   private int a,b,c;

   CombinationLock(int a1,int a2, int a3)
   {
      a = a1;
      b = a2;
      c = a3;
   }

   public boolean open(int x,int y, int z)
   {
      return x == a && y == b && z == c;
   }

   /* The old combination is xyz and the new to be changed is pqr */
   public boolean changeCombo(int x, int y, int z, int p, int q, int r)
   {
      if(this.open(x, y, z))
      {
            a = p;
            b = q;
            c = r;
            return true;
      }
      else
        return false;
  }
}
```

8. (a) equals, hashCode, toString.

   (b)

   ```
   if (s.equals(t))
   ```

   This equals call compares 'a' vs. 'a', and since they are equal, goes on to compare 'b' vs. 'b', and since equal, goes on to compare 'c' vs 'x' and returns false. For (s == t), the string references are compared (memory addresses), which must be different, so again evaluates to false (in this case).

   (c) hashCode value for "xyz" is the int 119193

   toString value for "xyz" is "xyz"

   hashCode value for Integer 8 is int 8

   toString value for Integer 8 is "8"

9. (a) An interface is a Java file that provides an API, a set of method descriptions (technically, method headers) of public methods, but no implementation whatsoever of that API. It also can specify

constants and nested interfaces. More technically, it provides ”abstract methods”, but this just means method declarations without implementation. Java 8 has introduced ”default methods,” with implementation, allowed in interfaces (but not equals, etc. Object methods), but we are not covering this new feature, which is intended for fixing up previously published APIs. Search ”default interface methods Java 8” if interested.

(b) methods, constants, (and nested interfaces, but not expected as an answer here)

(c)

```java
// It would be good to put comments on each of these methods--
public interface UnionFind {
  void union(int p, int q);
  int find(int p);
  boolean connected(int p, int q);
  int count();
}
// new top line for UF.java:
public class UF implements UnionFind {
```

(d) Methods/constructors of UF that are not in the interface: the constructor $UF(intn)$ and $main$.

(e) Yes, WeightedQuickUF qualifies for implementing the interface because it implements all the methods of the interface (and those method headers match the ones in the interface).