Atanas Dimitrov
Homework #1
CSCI 6470
Dr. Cai

Solution to problem 1:

Input:

Two n-bit binary integers, stored in two n-element arrays A[1..n] and B[1..n]. In addition an integer array C[1..n+1] which will hold the result after the completion of the function.

Output:

C[1..n+1] contains the sum of the n-bit binary integers stored in the input n-element arrays A[1..n] and B[1..n]. In other words: C[i+1]=A[i]+B[i]+r where r{0,1} is the cary over from the previous binary bit addition, namely A[i-1] + B[i-1] and C[1]=r-carry over from the last addition made.

Pseudocode:

```
Binary_Add (int A[1..n], int B[1..n], int C[1..n+1])
        temp←0
        i←0
        remainder←0
        for i←n to 0
                temp ← A[i] + B[i] + c
                c ← (temp >> 1) & 1
                C[i+1] ← temp & 1
        C[i+1] ← remainder;
```

Solution to problem 2:

a) Prove or disprove that 2^(n+1)=O(2^n).

Thus we need to show that ∃ constants c>0, n0>0 such that:

$0 \leqq 2^{(n+1)} \leqq c*(2^n)$ for all $n \geqq n0$.

Let's look at 2^(n+1):

$2^{(n+1)}=(2^n)*(2^1)=2*(2^n)$

Thus we now must show ∃ constants c>0, n0>0 such that:

$0 \leqq 2*(2^n) \leqq c*(2^n)$

So if we choose c=2 and n0=1 the above inequality holds for all $n \geqq n0$.

This completes the proof.

b) Prove or disprove that 2^(2*n)=O(2^n).

Thus we need to show that ∃ constants c>0, n0>0 such that:

$0 \leqq 2^{(2*n)} \leqq c*(2^n)$ for all $n \geqq n0$.

Let's look at 2^(2*n):

$2^{(2*n)}=(2^n)^2$

Let's compare the two functions, namely (2^n)^2 and 2^n and determine if the following inequality can ever hold from some n0 to ∞:

$(2*n)^2 \leq c*(2^n)$

Let $Y=2^n$, then the same inequality can be transformed into:

$Y^2 \leq c*Y$

The first derivative of Y^2 is $2*Y$ and for Y it is 1. We can clearly see that as Y grows Y^2 continues to increase its rate of growth. The rate of growth of Y is a lot slower. Looking at the problem now we can now clearly see that since Y^2 is a very rapidly growing exponential function, no matter how large we choose c to be, as Y increases Y^2 will eventually become larger than $c*Y$, when Y is large enough.

Having in mind our previous substitution we have now shown that

$(2*n)^2 \leq c*(2^n)$ cannot hold for all $n \geq n0>0$ and some $c>0$

Thus $2^{(2*n)} \not\in O(2^n)$.

Solution to problem 3:

Input:

Positive integer x.

Output:

TRUE if x is prime. FALSE if x is not prime.

Pseudocode:

```
bool Prime_Test ( int x )
        a←x                             1
        b←0                             1
        bool prime←TRUE                 1

        for b←2 to a-1                  2*(a-1)
            if a%b==0                   2*(a-1)
                    prime←FALSE         1
                    break               1
        return prime                    1
```

a) if we let n be the value of x then the worst-case time complexity of the algorithm is T(n)=O(n):

$T(a) = 1+1+1+2*(a-1)+2*(a-1)+1+1+1=6+4*(a-1)=4*a-4+6$

$=4*a+2=O(a)$

Since in this case n=a, we have

$T(n)=O(n)$

b) If we let n be the length of x (number of digits in x), if x is a decimal number, then the worst-case time complexity is $T(n)=O(10^n)$.

Suppose the number of digits is n, when x is decimal, then having in mind the algorithm (brute force), then the most number of devisions we will make is the number of permutations of 0 -9 in n places. In other words:

X X X X X X X X..... X X X X X X X

|---------- n ----------|

Where each $X \in \{0,1,2,3,4,5,6,7,8,9\}$, Thus, the number of permutations is:

10*10*10*10.....*10*10*10*10 = c*10^n
|---------- n ----------|

, for some c>0 which I use to represent the rest of the operations involved in the loop and outside the loop. We can exclude 0, 1, and the number itself. Thus the maximum total number of checks is:

c*(10^n)-3∈O(10^n)

If we let n be the length of x (number of digits in x), if x is a binary number, then the worst-case time complexity is T(n)=O(2^n).

Suppose the number of digits is n, when x is binary, then having in mind the algorithm (brute force), then the most number of devisions we will make is the number of permutations of 0 and 1 in n places. In other words:

X X X X X X X X..... X X X X X X X
|---------- n ----------|

Where each X∈{0,1}, Thus, the number of permutations is:

2*2*2*2.....*2*2*2*2 = c*2^n
|---------- n ----------|

, for some c>0 which I use to represent the rest of the operations involved in the loop and outside the loop. We can exclude 0, 1, and the number itself. Thus the maximum total number of checks is:

c*(2^n)-3∈O(2^n)

Running times are different and it will be faster to determine if x is prime when x is in its binary form. for example checking 5 for primality takes at most 10^1-3 = 7 divisions, but checking 101 for primality takes at most only 2^3-3 = 5 divisions (ignoring other operations).

Solution to problem 4:

Show that T(n)=2T(floor(n/2) - 17) + n = O(n*lgn)

Proof:

T(n) =2T(floor(n/2) + 17) + n
≦2T(n/2 + 17) + n

Inductive hypothesis:

Assume T(n)≦c*n*lgn holds for (n/2+17)

Then we have:

T(n/2+17)≦c*(n/2+17) * lg(n/2+17)

Inductive step:

T(n) ≦2*c*(n/2+17) * lg(n/2+17) + n
=2*c*((n+34)/2)*lg(n/2+17) + n
=c*(n+34)*lg(n/2+17) + n
≦c*(n+34)*lg(n/2+n/3) + n for n≧51
=c*(n+34)*lg(5*(n/6)) + n
=c*(n+34)*(lgn-lg(6/5)) + n
=(c*n+c*34)*(lgn-lg(6/5)) + n
=c*n*lgn + 34*c*lgn - log(6/5)*c*n - 34*lg(6/5)*c + n lg(6/5)≈0.3
≦c*n*lgn + 34*c*lgn - 0.3*c*n - 0.3*34*c + n
≦c*n*lgn + (34*c*lgn - 0.3*c*n - 10.2*c + n)

Now we must show when is

$A = 34*c*\lg n - 0.3*c*n - 10.2*c + n \leqq 0$

This also holds if

$34*c*\lg n - n*(0.3*c-1) - 10.2*c \leqq 0$, since $-10.2*c \leqq 0$

$34*c*\lg n - n*(0.3*c-1) \leqq 0$

$34*c*\lg n \leqq n*(0.3*c-1)$

$(34*c)/(0.3*c-1) \leqq n/\lg n$ $\quad c \not\asymp 3.3(3)$

Let $c=1$, $n \geqq 2$ then $(34*c)/(0.3*c-1) \approx -49$, thus

$-49 \leqq n/\lg n$ always holds.

Thus we have shown that $A \leqq 0$, and consequently:

$c*n*\lg n + A \leqq c*n*\lg n$

So $T(n) \leqq c*n*\lg n$.

Base case:

Let $T(35)=1$.

Then,

$T(36) = 2*T(36/2 + 17) + 36 = 2*T(35) + 36 = 2*1 + 26 = 38$

Checking for upper boundary:

$T(36) \stackrel{?}{\leqq} c*n*\lg n = 1*36*\lg 36 = 38*5.17 = 186.12$

$38 \leqq 186.12$

Thus our base case is T(36).

So for $n \geqq 36$, $c \geqq 1$ $T(n)=2T(\text{ floor}(n/2) - 17) + n = O(n*\log n)$