Assignment 1 | 4041 Algorithms and Data Structures
DUE: Sunday, Sept 18th, 10:00 pm | Fall 2016
Wyatt Kormick

1. Binary Addition. Write pseudocode to add two n-bit binary numbers. Use two n-element arrays to represent each of the operands and an n+1-element array to store the results. **The least significant bit is stored in the last element of the array.** Analyze the runtime as demonstrated on INSERTION-SORT(A) in Chapter 2. For each line, define the number of times it is executed, and write the equation for the best and worst case runtime of the entire algorithm.

```
Binary-Add(A, B)
    n = A.length                        1
    carry = 0                           1
    C =  new array, length n+1          1
    for i = 0 to n-1                    n
        sum = A[i]+B[i]+carry           n-1
        if sum == 3                     n-1
            carry = 1                   n-1
            C[i] = 1                    n-1
        else if sum == 2                n-1
            carry = 1                   n-1
            C[i] = 0                    n-1
        else                            n-1
            carry = 0                   n-1
            C[i] = sum                  n-1
    C[n] = carry                        1
```

T(n)=c1+c2+c3+c16+c5n+c6(n-1)…c15(n-1) = c5n+(c6+c7+…+c15)(n-1)+C

C=c1+c2+c3+c16

Best Case (Only first if occurs):
T(n)=c5n+(c6+c7+c8+c9)(n-1)+C

Worst Case (Only Else occurs):
T(n)=c5n+(c6+c7+c10+c13+c14+c15)(n-1)+C

The runtime of the loop doesn't get much better or worse. No matter the size of n (the length of A and B) the loop will run n times. There is some variation given the nature of the conditionals.

2. Write pseudocode for a THETA(nlgn) algorithm that, given a set S (i.e. unique numbers) of n integers and another integer x, determines whether or not there exist two numbers in S whose difference is exactly x. Briefly justify the runtime. Use a loop invariant to prove your algorithm is correct.

```
Set-Difference(S, x)
    n=S.length                      1
    i =  1                          1
    k = n-1                 1
    Merge-Sort(S, 0, n-1)           nlogn
    while i < k                     n
        if S[k] - S[i] == x                 n
            return true
        else if S[k] - S[i] < x     n
            k = k - 1
        else                n
            i = i + 1
    return false                    1
```

nlogn+cn+C. Merge-Sort has been proven to run in Θ(nlgn) time, and is the most expensive operation in this algorithm. Worst case, the while-loop runs in O(n) time.

S[k]-S[i] gets closer to x every iteration

LI: S[0]<=S[i] <= S[k]<=S[n-1],

Base Case: n=2, (S[0]=S[i])<(S[k]=S[1]),

Termination: i=k, S[0]<= S[i]=S[k]<=S[n-1] or S[k]-S[i]=x

3. Use mathematical induction to show that when *n* is an exact power of 2, the solution of the following recurrence is $T(n) = n \lg n$. Don't forget to provide a base case.

$$T(n) = \begin{cases} 2 & \text{if } n = 2 \\ 2T(n/2) + n & \text{if } n = 2^k\text{, for } k > 1 \end{cases}$$

Base Case: $n=2^1$ (k=1), T(2) = 2

T(4) = 2T(2)+4 = 8 = 4*log_2_4

T(8) = 2T(4)+8 = 24 = 8*log_2_8

Guess: n log n

$T(2^k) = 2T((2^{k-1}))+2^k = n \log n$

$k=k+1 \text{ -> } T(2^{k+1}) = 2T(2^k)+2^{k+1}$

$=T(2^{k+1})=2(2T((2^{k-1}))+2^k)+2^{k+1}$

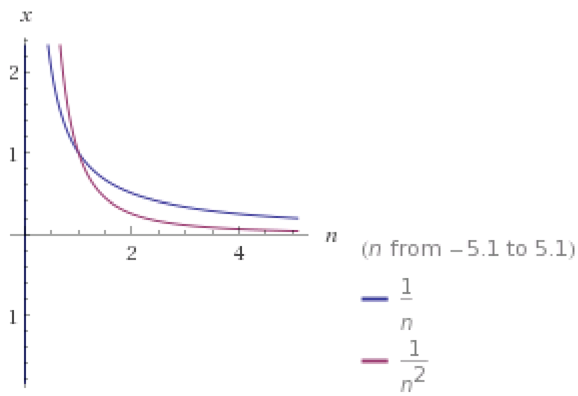
$=4T(2^{k-1})+2^{k+2} = n \log n$

4. Let f(n) and g(n) be asymptotically positive functions. Prove or disprove each of the following conjectures. To disprove a statement, you need to provide only 1 counterexample.

a. f(n) = O(g(n)) implies g(n) = O(f(n))
   a. Counterexample:
   b. $f(n) = n$, $g(n) = n^2$
   c. $f(n) = O(n^2)$
   d. g(n) != O(n)
   e. Because f(n) is smaller than n^2 (it can grow to n^2 with O(f(n))), but g(n) is already asymptotically larger than f(n)

b. f(n) = O(g(n)) implies lg(f(n)) = O(lg(g(n)), where lg(g(n)) ≥ 1 and f(n) ≥ 1 for all sufficiently large n

c. $f(n) = O((f(n))^2)$
   a. f(n) = 1/n
   b. f(n) is asymptotically larger than $f(n)^2$. f(n) is an upper bound on $f(n)^2$



5. Substitution Method

Using the master method in Section 4.5, you can show that the solution to the recurrence T(n) = 8T(n/4) + n is $T(n) = \Theta(n^{\log_4 8})$. Show that a substitution proof for establishing the upper bound with the assumption $T(k) \leq ck^{\log_4 8}$ fails. Then show how to subtract off a lower-order term to make a substitution proof work.

$n^{\text{log\_4\_8}}=n^{1.5}$, $O(n^{1.5})=n^2$, Guess: n^2

$T(n) <= cn^2$

$T(n) <= 8T(n/4) + n <= 8c(n^2/16)+n <= cn^2/2 + n <= cn^2+n$ which is not $<= cn^2$

We can subtract the lower order term n to fix this

New Assumption: $T(n) <= cn^2-xn$

$T(n) <= 8T(n/4) + n <= 8(cn^2/16 – xn/4)+n <= (cn^2/2-2xn+n = cn^2/2-n(2x+1)) <= cn^2+n(1-2x)$

When x = 1, $cn^2 – n$, and $cn^2 - n <= cn^2-xn$ which is our original assumption

6. Master Theorem

Use the Master Theorem to prove the bounds of the following. Justify your answer by defining f(n), a, b, and a bound for epsilon, if appropriate. If using rule 3, show that the second condition also holds.

a. $T(n) = 8T(n/3) + n^2$.
   a. a=8, b=3, $f(n)=n^2$
   b. log_b_a = log_3_8 = ~1.89
   c. $f(n) = n^2$ = Omega(n^C), ((c = 2) > 1.89), Case Three
   d. 8*f(n/3) <= kf(n)
      i. $(8n^2)/9 <= kn^2$, k<1, There exists a k that makes this true
   e. T(n) = Theta(f(n)) = Theta(n^2)

b. T(n) = 7T(n/3) + nlgn
   a. a=7, b=3, f(n)=nlgn
   b. log_3_7 = ~1.77
   c. f(n) = O(n^c) c< 1.77, case one
   d. T(n) = Theta($n^{log\_3\_7}$)

c. $T(n) = 4T(n/2) + n^2$.
   a. a=4, b=2, $f(n)=n^2$
   b. log_2_4 = 2
   c. f(n) = Theta($n^{log\_2\_4}$) = Theta (n^2), case two
   d. T(n) = Theta(n^2logn)