CSCI-UA.0310-004/005 Basic Algorithms February 4, 2015

Solutions to Problem 1 of Homework 1 (5 points)

Name: Jason Yao *Due: Wednesday, February 4*

You are given two binary integers a and b, where a has n bits, and b has m bits, where $n \geq m$. These integers are stored in two arrays $A[1 \dots n]$ and $B[1 \dots m]$, *in reverse order*. For example, if $a = 111000$ and $b = 1110$, then $A[1] = A[2] = A[3] = 0$, $A[4] = A[5] = A[6] = 1$, $B[1] = 0$, $B[2] = B[3] = B[4] = 1$. Your goal is to produce an array $C[1 \dots n+1]$ which stores the sum c of a and b. For example, $111000 + 1110 = 1000110$, meaning that $C[1 \dots 7] = 0110001$.

Write the pseudocode to produce C. Why was it convenient to store the arrays "backwards"?

Solution:

```
binAdd(a,b)

n = a.length
m = b.length
maxLength = abs(n - m)
A = new [n]
B = new [m]
C = new [maxLength + 1]

For i = n - 1 to 0
  A[i] = a % 10
  a /= 10
endFor

For j = m - 1 to 0
  B[j] = b % 10
  b /= 10
endFor

index = 0
carry = 0

while index <= n && index <= m
  if A[index] + B[index] == 2
    if carry == 1
      C[index] = 1
    endif
    if carry = 0
      C[index] = 1
    endif
    ++index
  endif

  if A[index] + B[index] == 1
    if carry == 1
      C[index] = 1
      carry = 0
    endif
    if carry == 0
      C[index] == 1
    endif
    ++index
  endif

  if A[index] + B[index] == 0
    if carry == 1
      C[index] = 1
      carry = 0
    endif
    if carry == 0
      C[index] = 0
    endif
    ++index
  endif
endwhile

while index <= n
  if carry == 1
    if A[index] == 0
      C[index] = 1
      carry = 0
    endif
    if A[index] == 1
      C[index] = 1
    endif
    ++index
  endif
  if carry == 0
    C[index] = A[index]
    ++index
  endif
endwhile

while index <= m
  if carry == 1
    if B[index] == 0
```

```
        C[index] = 1
        carry = 0
      endif
      if B[index] == 1
        C[index] = 1
      endif
      ++index
    endif
    if carry == 0
      C[index] = A[index]
      ++index
    endif
endwhile

if carry == 1
  C[maxLength] = 1
endif
```

It is convenient to store the arrays 'backwards' because we start binary addition from the last integer, allowing us to progress through the array normally instead of going from the other end. □

CSCI-UA.0310-004/005 Basic Algorithms February 4, 2015

Solutions to Problem 2 of Homework 1 (5 points)

Name: Jason Yao *Due: Wednesday, February 4*

In class we learned how to implement insertion sort by comparing the key element to the largest element in the sorted portion of the array, and moving that element to the right, if it was larger than the key, and then comparing the key to successively smaller elements until the right position is found.

Implement a variation of insertion sort, in which you instead compare the key to the smallest element in the sorted portion of the array and then iterate by comparing to successively larger elements. How does this algorithm compare in terms of efficiency to the traditional insertion sort?

Solution:

```
IS(A[])
for i = 1 to A.length - 1
   key = A[i]
   j = 0
   while j < i && A[j + 1] < key
      A[j] = A[j + 1]
      ++j
   endwhile
  A[j] = key
endfor
```

The algorithm is of comparable efficiency to that of normal insertion sort, due to the fact that the insertion sort is simply starting the comparison from the other end, while still having to go through O(n^2) □

CSCI-UA.0310-004/005 Basic Algorithms February 4, 2015

Solutions to Problem 3 of Homework 1 (10 Points)

Name: Jason Yao *Due: Wednesday, February 4*

For each of the following pairs of functions $f(n)$ and $g(n)$, state whether f is $O(g)$; whether f is $o(g)$; whether f is $\Theta(g)$; whether f is $\Omega(g)$; and whether f is $\omega(g)$. (More than one of these can be true for a single pair!)

(a) $f(n) = 13n^{19} + 92$; $g(n) = \frac{n^{24}-n^{23}+5}{5n^4+2000}$.

Solution: f(n) $\approx n^{19}$, g(n) $\approx n^{20}$
$\lim_{n\to\infty} \frac{f(n)}{g(n)}$, $\lim_{n\to\infty} \frac{n^{19}}{n^{20}}$, $\lim_{n\to\infty} \frac{1}{n}$, $\lim_{n\to\infty} 0$

Thus f(n) = o(g(n)), and by definition, f(n) = O(g(n)) □

(b) $f(n) = \log(n^{19} + 3n)$; $g(n) = \log(n^{0.2} - 1)$.

Solution: f(n) $\approx log(n^{19})$, g(n) $\approx log(n^{0.2})$
$\lim_{n\to\infty} \frac{f(n)}{g(n)}$, $\lim_{n\to\infty} \frac{log(n^{19})}{log(n^{0.2})}$, $\lim_{n\to\infty} \frac{19*log(n)}{0.2*log(n)}$, $\lim_{n\to\infty} \frac{19}{0.2}$, $\lim_{n\to\infty} 95$

Thus f(n) = θ(g(n)) □

(c) $f(n) = \log(5^n + n)$; $g(n) = \log(n^{20})$.

Solution: f(n) $\approx log(5^n)$, g(n) $= log(n^{20})$
$\lim_{n\to\infty} \frac{f(n)}{g(n)}$, $\lim_{n\to\infty} \frac{log(5^n)}{log(n^{20})}$, $\lim_{n\to\infty} \frac{n*log(5)}{20*log(n)}$, $\lim_{n\to\infty} \infty$,

Thus f(n) = $\omega(g(n))$, and by definition, f(n) = $\Omega(g(n))$ □

(d) $f(n) = n^4 \cdot 3^n$; $g(n) = n^3 \cdot 4^n$.

Solution: $\lim_{n\to\infty} \frac{f(n)}{g(n)}$, $\lim_{n\to\infty} \frac{3^n \cdot n^4}{4^n \cdot n^3}$, $\lim_{n\to\infty} (\frac{3}{4})^n$ $\lim_{n\to\infty} 0$

Thus f(n) = o(g(n)), and by definition, f(n) = O(g(n)) □

(e) $f(n) = (n^n)^2$; $g(n) = n^{(n^2)}$.

Solution: f(n) $= n^{2n}$, g(n) $= n^{n^2}$
$\lim_{n\to\infty} \frac{f(n)}{g(n)}$, $\lim_{n\to\infty} \frac{n^{2n}}{n^{n^2}}$, $\lim_{n\to\infty} (\frac{2n*log(n)}{n^2*log(n)})^n$ $\lim_{n\to\infty} 0$

Thus f(n) = o(g(n)), and by definition, f(n) = O(g(n)) □

CSCI-UA.0310-004/005 Basic Algorithms February 4, 2015

Solutions to Problem 4 of Homework 1 (10 points)

Name: Jason Yao *Due: Wednesday, February 4*

The following two functions both take as arguments two n-element arrays A and B:

```
MAGIC-1($A, B, n$)
    For $i = 1$ to $n$
        For $j = 1$ to $n$
            If $A[i] \geq B[j]$ Return FALSE
    Return TRUE
```

```
MAGIC-2($A, B, n$)
    $temp := A[1]$
    For $i = 2$ to $n$
        If $A[i] > temp$ Then $temp := A[i]$
    For $j = 1$ to $n$
        If $temp \geq B[j]$ Return FALSE
    Return TRUE
```

(a) (2 points) It turns out both of these procedures return TRUE if and only if the same 'special condition' regarding the arrays A and B holds. Describe this 'special condition' in English.

Solution: The special condition is when A[i] ¡ B[j] □

(b) (5 points) Analyze the worst-case running time for both algorithms in the Θ-notation. Which algorithm would you chose? Is it the one with the shortest code (number of lines)?

Solution: As Magic-1 has a worst-case running time of $\Theta(n^2)$, and Magic-2 has a worst-case running time of $\Theta(n)$, that means that Magic-2 is much more efficient than Magic-1, and thus I would choose Magic-2 as the optimal solution. □

(c) (3 points) Does the situation change if we consider the best-case running time for both algorithms?

Solution: No, the situation does not change if we consider the best-case running time for the algorithms, since both running times would remain the same as their worst-case scenario counterparts. □