CSCI-GA.1170-001/002 Fundamental Algorithms September 9, 2014

Solutions to Problem 1 of Homework 1 (16 (+4) points)

Name: Sahil Goel *Due: Tuesday, September 10*

A degree-n polynomial $P(x)$ is a function

$$P(x) = a_0 + a_1x + \ldots + a_{n-1}x^{n-1} + a_nx^n = \sum_{i=0}^{n} a_ix^i$$

(a) (2 points) Express the value $P(x)$ as

$$P(x) = a_0 + a_1x + \ldots + a_{n-2}x^{n-2} + b_{n-1}x^{n-1} = \sum_{i=0}^{n-1} b_ix^i$$

where $b_0 = a_0, \ldots, b_{n-2} = a_{n-2}$. What is b_{n-1} as a function of the a_i's and x?

Solution:

$$P(x) = a_0 + a_1x + \ldots + a_{n-1}x^{n-1} + a_nx^n = \sum_{i=0}^{n} a_ix^i - 1$$

$$P(x) = a_0 + a_1x + \ldots + a_{n-2}x^{n-2} + b_{n-1}x^{n-1} = \sum_{i=0}^{n-1} b_ix^i - 2$$

Equating equations 1 and 2, since $a_0 = b_0, a_1 = b_1, ..., a_{n-2} = b_{n-2}$

$$a_{n-1} * x^{n-1} + a_n * x^n = b_{n-1} * x^{n-1}$$

$$b_{n-1} = a_{n-1} + a_n * x$$

$$P(x) = b_0 + b_1x + \ldots + b_{n-2}x^{n-2} + b_{n-1}x^{n-1} = \sum_{i=0}^{n-1} b_ix^i$$

where $a_0 = b_0, a_1 = b_1, ..., a_{n-2} = b_{n-2}$ and $b_{n-1} = a_{n-1} + a_n * x$

□

(b) (5 points) Using part (a) above write a recursive procedure $\mathsf{Eval}(A, n, x)$ to evaluate the polynomial $P(x)$ whose coefficients are given in the array $A[0 \ldots n]$ (i.e., $A[0] = a_0$, etc.). Make sure you do not forget the base case $n = 0$.

Solution: There are 3 solutions to this problem.
1. With base case n=0
2. With base case n=-1
3. With base case n=degree of polynomial (Assumption : if degree of polynomial is accessible

inside the function and initially calling the function as eval(A,x,0)). This is the most optimal solution amongst the three

1. Base case n=0

Eval(A,x,n)
{
if(n==0)
return A[0];
return $x^n * A[n]$ + Eval(A,x,n-1);
}

2. Base case n=-1

Eval(A,x,n)
{
if(n==-1)
return 0;
return $x^n * A[n]$ + Eval(A,x,n-1);
}

3. Base case n=degree of polynomial
Call from main(), Eval(A,x,0)
Assumption: Degree of polynomial known inside function
Horner's Rule using recursion

Eval(A,x,n)
{
if(n==DegreeOfPolynomial)
return A[n];
return $A[n] + x * Eval(A, x, n + 1)$;
}

□

(c) (3 points) Let $T(n)$ be the running time of your implementation of Eval. Write a recurrence equation for $T(n)$ and solve it in the $\Theta(\cdot)$ notation.

Solution: Assuming calculation of x^n takes $O(n)$ times
For the first 2 solutions of 1-(c), the recurrence equation will become

$$T(1) = c_1$$
$$T(n) = T(n-1) + c * n$$

Solving the above recurrence equation

$$T(n) = T(n-2) + c*(n-1) + c*n$$
$$T(n) = T(n-3) + c*(n-2) + c*(n-1) + c*n$$

Solving in the similar manner

$$T(n) = T(1) + c*(n + (n-1) + (n-2) + \ldots + 1)$$

$$T(n) = c1 + c*((n)(n+1)/2)$$

Therefore $T(n) = O(n^2)$

Now calculating the complexity for horner's rule method (iii) Part

$$T(1) = c_1$$
$$T(n) = T(n-1) + c$$

Solving for the recurrence equation above

$$T(n) = T(n-2) + c + c$$

Solving similarly

$$T(n) = n*c$$

Therefore $T(n) = O(n)$ which is much better than $O(n^2)$

□

(d) (6 points) Assuming n is a power of 2, try to express $P(x)$ as $P(x) = P_0(x) + x^{n/2}P_1(x)$, where $P_0(x)$ and $P_1(x)$ are both polynomials of degree $n/2$. Assuming the computation of $x^{n/2}$ takes $O(n)$ times, describe (in words or pseudocode) a recursive procedure Eval_2 to compute $P(x)$ using two recursive calls to Eval_2. Write a recurrence relation for the running time of Eval_2 and solve it. How does your solution compare to your solution in part (c)?

Solution: If n is power of 2 $P(x)$ can be split into $P_0(x)$ and $P_1(x)$. For example suppose

$$P(x) = a_0 + a_1 * x + a_2 * x^2 + a_3 * x^3 + a_4 * x^4$$

It could be divided as

$$P(x) = P_0(x) + x^2 * P_1(x)$$

where $P_0(x) = a_0 + a_1 * x + a_2 * x^2$ and $P_1(x) = a_3 * x + a_4 * x^2$
Now both the polynomials are 2 degree polynomials and both can be recursively called to evaluate the value of expression

```
Eval2(A,x,l,h)                                   %Where l is lower bound and h is higher bound
{
if(l==h)
return A[l];
return Eval2(A,x,l,((l+h)/2)) + [x^((h-l)/2+1)]*Eval2( A, x, ((l+h)/2)+1 , h);
}
```

Here First call to Eval2 is for first half of polynomial i.e. from 0 to n/2 i.e. (l+h)/2. Second call to Eval2 is for second half of polynomial i.e. from (n/2)+1 i.e. ((l+h)/2 + 1) to h.
Recurrence Equation will become:

$$T(1) = c_1$$

since calculation of $x^{n/2}$ takes O(n) time, time consumed in that step is $c_2 * n$

$$T(n) = 2 * T(n/2) + c_2 * n;$$

By substitution

$$T(n) = 2 * (2 * T(n/4) + c_2 * (n/2)) + c_2 * n;$$
$$T(n) = 4 * T(n/4) + 2 * c_2 * n$$

Substituting in similar fashion

$$T(n) = n * T(1) + log(n) * n$$
$$T(n) = n * c + log(n) * n$$

Therefore $T(n) = O(n * log(n))$

□

(e) (**Extra Credit.**) Explain how to fix the slow "conquer" step of part (d) so that the resulting solution is as efficient as "expected".

Solution: Slow conquer step i.e. calculation of x^n can be fixed by calculating x^n in $log(n)$ time.
Algorithm for the same will be

```
POW(x,n)
{ if(n==0)
return 1;
temp=POW(x,n/2);
if(n%2==0);
return temp*temp;
else
return x*temp*temp;
}
```

Total complexity of algorithm will become $O(n)$ □

CSCI-GA.1170-001/002 Fundamental Algorithms September 9, 2014

Solutions to Problem 2 of Homework 1 (10 Points)

Name: Sahil Goel *Due: Tuesday, September 10*

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) $f(n) = 32n^{21} + 2$; $g(n) = \frac{n^{22}+3n+4}{111} - 52n$.

Solution: f is $o(g)$
f is $O(g)$
since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

□

(b) $f(n) = \log(n^{21} + 3n)$; $g(n) = \log(n^2 - 1)$.

Solution: f is $\Omega(g)$
f is $O(g)$
f is $\Theta(g)$

□

(c) $f(n) = \log(2^n + n^2)$; $g(n) = \log(n^{22})$.

Solution: f is $\omega(g)$
f is $\Omega(g)$

□

(d) $f(n) = n^3 \cdot 2^n$; $g(n) = n^2 \cdot 3^n$.

Solution: f is $o(g)$
f is $O(g)$

□

(e) $f(n) = (n^n)^3$; $g(n) = n^{(n^3)}$.

Solution: f is $o(g)$
f is $O(g)$

□

CSCI-GA.1170-001/002 Fundamental Algorithms	September 9, 2014

Solutions to Problem 3 of Homework 1 (10 points)

Name: Sahil Goel *Due: Tuesday, September 10*

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: If every value in the array A is smaller than any of the values in array B then the result is TRUE. In other words, if the maximum of array A elements is smaller than the minimum of array B elements, than the result will be true. □

(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: In worst case, considering MAGIC-1 code, the outer loop will run n times and the inner loop will run n times for every outer loop.
So total number of steps will be n^2
Hence the worst case complexity of algorithm is $\Theta(n^2)$

In worst case, considering Magic-2 Code, Finding the maximum of array A will take n comparisons i.e. if the element is at the end of array A.
In next step worst case will be when maximum of array A is smaller than every element of B except the last element. It will also take n comparisons.
Total Complexity will become $\Theta(n)$.
More optimal solution is the second code with more lines of code, so it is not the one with shortest lines of code.

□

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

Solution: Yes the situation changes if we consider the best case running time algorithm for both algorithms because in Magic-1, there will be only one comparison in best case that is when first element of A will be larger than the first element of array B, it will exit and return false.
Whereas in Magic-B, there will be minimum n comparisons to find out the maximum of array A.
Therefore, Magic-1 best case running time is better than Magic-2 best case running time.

□