Ayman Al Zaatari
11 December 2014
Assignment 9 Solution

Assignment 9 Solution

Ex1:

a)

- 18 <= x ➔ 17x + 11 <= 18x +x <= x^2
  ➔ 17x + 11 <= C x^2
  ➔ f(x) ∊ O(x^2) for k=18 and C=1
- If f(x) ∉ Ω(x^2) ➔ 17x + 11 > C x^2 for C >0 and x >=k
  x > 18 ➔ 18x >= 17x + x > 17x + 11 ➔ 18x > Cx^2
  dividing by x, 18 > Cx => x < C_1 , for C1 = C/18, but x > k => impossible
  ➔f(x) ∉ Ω(x^2)

  ➔f(x) ∉ θ(x^2)

b)

- x^2 + 1000 ≤ 1000x^2 for x>1000
  ➔f(x) ∊ O(x^2) for C=1000 and k=1000

- x^2 + 1000 ≥ x^2 for x>1000
  ➔f(x) ∊ Ω(x^2) for C=1 and k=1000

  ➔f(x) ∊ θ(x^2) for k=1000

c)

- logx < x ➔ x logx ≤ x^2 for x > 1
  f(x) ∊ O(x^2) for k=1 and C=1

- Assume f(x) ∊ Ω(x^2) ➔ xlogx >= C x^2 such that C>0, |
  ➔ logx >= Cx ➔ logx/x >= C , but we know that the limit at infinity = 0,
  ➔ false Assumption
  ➔ f(x) ∉ Ω(x^2)

  ➔f(x) ∉ θ(x^2)

d)

- x^4/2 ≤ Cx^2 ➔ x^2 ≤ 2C ➔ x <Sqrt(2C)
  ➔f(x) ∉ O(x^2)

- x^4/2 ≥ Cx^2 => x^2 ≥ 2C for C=1 and k=2
  ➔f(x) ∊ Ω(x^2) for C=1 and k=2

➔ $f(x) \notin \theta(x^2)$

e)

- $2^x \leq Cx^2$ ➔ $x \leq C + 2\log x$ ➔ $x/2\log x <= (C+1)/2\log x$…At infinity, the limits would be $\infty <= 0$ which is false

  ➔ $f(x) \notin O(x^2)$

- $2^x \geq Cx^2 => x \geq C + 2\log x$ which is true for $x > 10$ and $C = 1$
  ➔ $f(x) \in \Omega(x^2)$ for C=1 and k=10

  ➔ $f(x) \notin \theta(x^2)$

f)

let $x=a+\varepsilon$ for $0 <= \varepsilon < 1$, ➔ $\mathbf{a^2 \leq \lfloor x \rfloor \cdot \lceil x \rceil \leq (a+1)^2}$

- $a \leq x \leq a+1$
  $a^2 \leq \lfloor x \rfloor \cdot \lceil x \rceil \leq (a+1)^2 \leq 3a^2 \leq Cx^2$ for $x>1$
  ➔ $f(x) \in O(x^2)$ for C=3 and k=1
- $x^2 = (a+\varepsilon)^2 = a^2 + 2a\varepsilon + \varepsilon^2 >= a^2$
  ➔ $x^2 \leq \lfloor x \rfloor \cdot \lceil x \rceil \leq (a+1)^2$
  ➔ $f(x) \in \Omega(x^2)$ for C=1 and k=1

➔ $f(x) \in \theta(x^2)$

Ex2:

1000 log n, sqrt(n), n log n, $n^2/1000000$, 2^n, 3^n, 2n!

Ex3:

We can pick the function with highest order when we have addition, and give the multiplaction of the functions when we have mutliplication

a) $f(x) = O(n^3) * O(\log n) + O(17\log n)*O(n^3) = O(n^3\log n)$

b) $f(x) = O(2^n)*O(3^n) = O(6^n)$

c) $f(x) = O(n^n)*O(n!) = O(n^n \times n!)$

Ex4:

a)

- $f(x) = x^2$, its obviously $\Theta(x^2)$
- $g(x) = 2x^2 + x – 7$, for $x > 7 > -7$ ➔ $2x > x-7 > 0$ and $x^2 > 2x$
  ➔ $2x^2 + x - 7 <= 2x^2 + 2x <= 4x^2$

➔ g(x) is Θ(x^2)
➔ f and g are of same order

b)

- f(x) = x which is obviously Θ(x)
- g(x) =⌊x+1/2⌋, consider 2 cases of x
    a. x = a + ε, and ε < 0.5, then ⌊x+1/2⌋ = a = x - ε, then its Θ(x)
    b. x = a + ε, and ε > 0.5, then ⌊x+1/2⌋ = a +1 = x +(1- ε), then its Θ(x)

  ➔f and g are of same order

c)

- f(x) = log(x^2) = 2logx which is obviously Θ(logx)
- g(x) = log(x^2+1)

Need to prove:

C_1 logx <= log(x^2+1) <= C_2 logx
2^{C1logx} <= $2^{log(x^2 +1)}$ <= 2^{C2logx}
X^{C1} <= x^2+1 <= X^{C2}, which is true C_1 = 0.5, C_2 = 3, and x > 3
➔ g is Θ(logx)
➔ f and g are of same order

d)

- f(x) = log_{10}x = log_2x/ $log_2$10, which is obviously Θ(lgx)
- g(x) = log_2x, which is obviously Θ(lgx)

  ➔f and g are of same order

Ex5:

Applying the following algorithm for getting x^{2^k} will be done as follows:

Step	Previous Value	Resulting Value
1	x^1	x^2
2	x^2	x^4
i	$x^{2^{i-1}}$	x^{2^i}
i+1	x^{2^i}	$x^{2^{i+1}}$
k	$x^{2^{k-1}}$	x^{2^k}

So, in K steps, we will havex^{2^k}, so to calculatex^n, we will need log(n) steps

However, the naïve way for calculatingx^n, starting with 1, and multiplying by x, will need n steps instead of log(n)…so the first way is more efficient

Ex6:
1 Day = 24 * 60 * 60 = 86400 seconds.

a) (10^{-11})*logn = 86400
   $n=2^{86400\ E11}$
b) (10^{-11})*1000n = 86400
   $n=86.4\times10^{11}$
c) (10^{-11})*n^2 = 86400
   n = sqrt(86400×10^{11})
d) (10^{-11})*1000n^2 = 86400
   n = sqrt(86.4×10^{11})
e) (10^{-11})*n^3 = 86400
   n = $\sqrt[3]{86400 \times 10^{11}}$
f) (10^{-11})*×2^n = 86400
   n = log(86400×10^{11})
g) (10^{-11})*×2^{2n} = 86400
   n = log(86400×10^{11})/2
h) (10^{-11})*×2^{2^n} = 86400
   n = log(log(86400×10^{11}))

Ex8:
procedure OccurMoreThanOnce ($a_1, a_2, . . . , a_n$: nondecreasing integers)

```
for i:= 1 to n-1
        while(i < n-1 && a_{i+1} = a_i)
                i++
        print(a_i)

if(n = 1 || a_n != a_{n-1})
        print(a_n)
```

The worst case complexity of this algorithm is $\Theta(n)$

Ex9:
To check if a set of n talks can be scheduled together, we need $\Theta(nlgn)$ – Sort, then check if there are conflicts

We have 2^n subsets for n talks. Thus the running time will be $O(nlgn * 2^n)$

Ex10:
-Linear search finds the value in O(n), so doubling the size will result in doubling the number of comparisons
-Binary search find the value in O(lgn), so doubling the size will result in 1 extra comparison