CS5800: Algorithms — Iraklis Tsekourakis

Homework 1

Name: Christo Frank Franklin

1. (18 points) In the following, use a direct proof (by giving values for c and n_0 in the formal definition of big-O/Ω notation) to prove that:

(a) $n^2 + 7n + 1$ is $\Omega(n^2)$
Solution:

The given function, $f(n) = n^2 + 7n + 1$.
To Prove: $f(n)$ is $\Omega(n^2)$.

By the formal definition:
if $f(n)$ is $\Omega(n^2)$, then

$$f(n) \geq C \cdot g(n)$$

$$n^2 + 7n + 1 \geq C \cdot g(n)$$

Considering $g(n) = n^2$, we have

$$n^2 + 7n + 1 \geq C \cdot n^2$$

Taking the value of $C = 1$, we get

$$n^2 + 7n + 1 \geq n^2$$

$$7n + 1 \geq 0$$

$$n \geq -\frac{1}{7}$$

Hence, for all values above $n > 1$ here n = 1 and with $C = 1$, the equation holds good.

(b) $3n^2 + n - 10$ is $O(n^2)$
Solution:

The given function, $f(n) = 3n^2 + n - 10$.
By the formal definition:
If $f(n)$ is $O(n^2)$, then

$$f(n) \leq C \cdot g(n)$$

$$3n^2 + n - 10 \leq C \cdot g(n)$$

Considering $g(n) = n^2$, we have

$$3n^2 + n - 10 \leq C \cdot n^2$$

Taking the value of $C = 4$, we get

$$3n^2 + n - 10 \leq 4n^2$$

$$n^2 - n + 10 \geq 0$$

From this expression, for any value of n, the equation is satisfied.
Hence, for all values above $n > 1$, here n = 1 and with $C = 4$, the equation holds good.

(c) n^2 is $\Omega(n \lg n)$
Solution:

Given function: $f(n) = n^2$

To Prove: $f(n)$ is $\Omega(n^2)$ by the formal definition.

If $f(n)$ is $\Omega(n^2)$, then

$$f(n) \geq C \cdot g(n)$$

$$n^2 \geq C \cdot g(n)$$

Considering:

$$g(n) = n \log n$$

we have

$$n^2 \geq C \cdot n \log n$$

Dividing both sides by $n \log n$:

$$\frac{n}{\log n} \geq C$$

$$C \leq \frac{n}{\log n}$$

.

Verification: For $C = 4$ and $n = 2$, the given equation holds good.

---

2. (20 points) In the following, use the iteration method to find the asymptotic notation of the order of growth of the recurrences:

(a) $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + b & \text{if } n > 1 \end{cases}$
Solution:

Given Recurrence function is

$$T(n) = 2 \cdot T(n/2) + b$$

$$T(n) = 2 \cdot (2 \cdot (T(n/4) + b)) + b$$

$$T(n) = 4 \cdot T(n/4) + 3b$$

Substituting for $T(n/4)$,

$$T(n) = 4 \cdot (2 \cdot T(n/8) + b) + 3b$$

$$T(n) = 8 \cdot T(n/8) + 7b$$

If $n = k$,

$$T(k) = 2^k \cdot T(k/2^k) + (2^k - 1) \cdot b$$

Upon subsequent substitution, it reaches $T(1)$ where

$$\frac{k}{2^k} = 1$$

$$k = \log n$$

$$\begin{aligned} T(n) &= 2^{\log n} \cdot T(1) + b \cdot \left(2^{\log n} - 1\right) \\ &= n + b \cdot (n-1) \\ &= n(1+b) - b \end{aligned}$$

Hence, the asymptotic notation would be $\Theta(n)$.

(b) $T(n) = \begin{cases} c & \text{if } n = 0 \\ T(n-1) + n + b & \text{if } n > 1 \end{cases}$

Solution:

$$T(n) = T(n-1) + n + b$$

$$T(n) = [T(n-2) + (n-1) + b] + n + b$$

$$T(n) = [T(n-3) + (n-2) + b] + (n-1) + b + n + b$$

$$T(n) = T(n-3) + (n-2) + (n-1) + n + 3b$$

$$T(n) = T(n-k) + (n-k) + \cdots + (n-2) + (n-1) + n + 3b$$

$$T(n) = T(0) + \sum_{i=0}^{n} (n-i) + nb$$

$$= \frac{n(n+1)}{2} + nb + c$$

$$= \frac{n^2}{2} + \frac{n}{2} + nb + c$$

From this, it can be inferred that the growth of complexity is

$$O(n^2)$$

---

3. (20 points) Solve the following recurrences using the substitution method:

(a) $T(n) = T(n-3) + 3\lg n$. Our guess is $T(n) = O(nlgn)$. Show that $T(n) \leq cn\lg n$ for some constant $c > 0$ (Note that $\lg n$ is monotonically increasing for $n > 0$)
Solution:

Our guess: $O(n \log n)$
By formal definition,

$$T(n) \leq C \cdot n \log n \quad \text{for constant } C > 0$$

$$T(n) \leq C(n-3)\log(n-3) + 3\log n$$

Since the logarithm function is monotonically increasing,

$$\log n \geq \log(n-3)$$

$$T(n) \leq C(n-3)\log n + 3\log n$$

$$T(n) \leq Cn\log n - 3C\log n + 3\log n$$

$$T(n) \leq Cn\log n - (3C+3)\log n$$

For all $C \geq 1$, since the second term must be negative,

$$3C + 3 \leq 0 \quad \Rightarrow \quad C \geq 1$$

$$T(n) \leq Cn\log n - (3C+3)\log n$$

For $C = 1$, the equation holds true:

$$T(n) \leq n\log n - 6\log n$$

Hence, the complexity of $T(n)$ is $O(n\log n)$.

(b) $T(n) = 4T(n/3) + n$. Our guess is $T(n) = O(n^{\log_3 4})$. Show that $T(n) \leq cn^{\log_3 4}$ for some constant $c > 0$
Solution:

Our guess is $\mathrm{O}(\mathrm{n}^{\log_3 4})$
By Formal Definition,

$$T(n) \leq C \cdot n^{\log_3 4} \quad \text{for constant } C > 0$$

$$T(n) \leq 4 \cdot \left(C \cdot \left(\frac{n}{3}\right)^{\log_3 4} \right) + n$$

$$T(n) \leq 4C \cdot \frac{n^{\log_3 4}}{4} + n$$

$$T(n) \leq C \cdot n^{\log_3 4} + n$$

The n term:

$$T(n) \leq C \cdot n^{\log_3 4} + n$$

Since we have an additional linear term.
Improving the guess,

$$T(n) \leq C \cdot \left(n^{\log_3 4} - n\right)$$

$$T(n) \leq 4 \cdot C \cdot \left(\frac{n}{3}\right)^{\log_3 4} - n + n$$

$$T(n) \leq 4C \cdot \left(\frac{n^{\log_3 4}}{3^{\log_3 4}}\right) - 4C \cdot n + n$$

$$T(n) \leq 4C \cdot \left(\frac{n^{\log_3 4}}{4}\right) - 4C \cdot n + n$$

For C = 1:

$$T(n) \leq n^{\log_3 4} - 3n$$

Hence, the complexity would be

$$T(n) \leq n^{\log_3 4} - n$$

---

4. (20 points) You can also think of insertion sort as a recursive algorithm. In order to sort `A[1:n]`, recursively sort the subarray `A[1:n-1]`. Write pseudocode for this recursive version of insertion sort. Give a recurrence for its worst-case running time.

Solution:

Algorithm: InsertionSort

```
InsertionSort(arr, n):
    if n < 1:
        return

    InsertionSort(arr, n-1)
```

```
key = arr[n]
index = n - 1

while index > 0 and key < arr[index]:
    arr[index + 1] = arr[index]
    index -= 1

arr[index + 1] = key
```

---

5. (20 points) Let $f(n)$ and $g(n)$ be asymptotically nonnegative functions. Using the basic definition of Θ-notation, prove that $\max f(n), g(n) = \Theta(f(n) + g(n))$

Solution:

By Definition of $\Theta(f(n) + g(n))$,

$$\max\{f(n), g(n)\} = \Theta(f(n) + g(n))$$

$$c_1(f(n) + g(n)) \leq \max(f(n), g(n)) \leq c_2(f(n) + g(n))$$

Considering the Upper Bound,

$$\max(f(n), g(n)) \leq c_2(f(n) + g(n))$$

For $c_2 \geq 1$, then

$$\max(f(n), g(n)) \leq (f(n) + g(n))$$

Considering the Lower Bound,

$$\max(f(n), g(n)) \geq c_1(f(n) + g(n))$$

For $c_1 \leq \frac{1}{2}$, then

$$\max(f(n), g(n)) \geq 0.5(f(n) + g(n))$$

---

6. (20 points) Is $2^{n+1} = O(2^n)$? Is $2^{2n} = O(2^n)$? Use the formal definition of O-notation to answer these two questions.

Solution:

(a)

Given function is $f(n) = 2^{(n+1)}$.
Applying the formal definition:

$$f(n) \leq c \cdot g(n)$$

Let's consider $g(n) = 2^n$,

$$2^{(n+1)} \leq C \cdot 2^n$$

If $c \geq 2$:

Taking $c = 2$,

$$2^{(n+1)} \leq 2 \cdot 2^n$$

$$2^{(n+1)} \leq 2^{(n+1)}$$

For $c \geq 2$, taking $c = 4$,

$$2^{(n+1)} \leq 4 \cdot 2^n$$

$$2^{(n+1)} \leq 2^2 \cdot 2^n$$

$$2^{(n+1)} \leq 2^{(2+n)}$$

The equation is satisfied for the value of c.

(b)

Given:

$$f(n) = 2^{(2n)}$$
$$g(n) = 2^n$$

Applying the formal definition:

$$f(n) \leq c \cdot g(n)$$

$$2^{(2n)} \leq c \cdot 2^n$$

$$c \geq 2^n$$

There cannot be any constant c such that it is always greater than 2^n, as 2^n grows exponentially.