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, $f(n) = n^2+7n+1$.

By the formal definition of the lower bound (Big-Omega notation), there exist positive constants c and n_0 such that for all $n \geq n_0$:

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

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

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

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

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

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

$$7n+1 \geq 0$$

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

Since we consider asymptotic behavior for large values of n, we take $n_0 = 1$. Thus, for all $n \geq 1$ and with $c = 1$, the inequality holds.

$$\therefore f(n) = \Omega(n^2)$$

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

Solution:

The given function, $f(n) = 3n^2 + n - 10$.

By the formal definition of Big O:

Let $f(n) = O(g(n))$ if and only if there exist positive constants c and n_0 such that for all $n \geq n_0$,

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

Now applying this to the problem:

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 for $n = 1$ and with $c = 4$, the equation holds good.

Now let's check for a particular value:

We check if the inequality holds true for $n = 2$ and $c = 4$.

For $f(n) = 3n^2 + n - 10$ and $g(n) = n^2$, we check if:

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

Substituting $n = 2$ and $c = 4$, we get:

$$3(2)^2 + 2 - 10 \leq 4 \cdot (2)^2$$

$$3(4) + 2 - 10 \leq 4 \cdot 4$$

$$12 + 2 - 10 \leq 16$$

$$4 \leq 16$$

This inequality holds true for $n = 2$ and $c = 4$.

(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.

By the formal definition of Big Omega:
Let $f(n) = \Omega(g(n))$ if and only if there exist positive constants c and n_0 such that for all $n \geq n_0$,

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

Now applying this to the problem:
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 $n = 10$, the inequality holds true:

$$\frac{10}{\log 10} = \frac{10}{2.3026} \approx 4.34 \geq 4$$

Thus, the inequality holds for $n = 10$ for $c = 4$.

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

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

Upon next iteration,

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

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

Upon next iteration, 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 the base case $T(1)$ when

$$\frac{k}{2^k} = 1$$
$$k = \log n$$
$$T(n) = 2^{\log n} \cdot T(1) + b \cdot \left(2^{\log n} - 1\right)$$
$$= n + b \cdot (n - 1)$$
$$= n(1 + b) - b$$

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

$$\theta(n^2)$$

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,

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

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

To verify, Let consider $C = 1$, 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

$$O(n^{\log_3 4})$$

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

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

Substituting the guess to the given function

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

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

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

For C = 1 and d = 1:

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

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

Hence, the complexity would be O (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
```

To calculate the recurrence relation for the worst case,

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

By using iteration method,
1st Iteration :

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

2nd Iteration :

$$T(n) = T(n-2) + c(n-2) + d + c(n-1) + d$$

3rd Iteration :

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

(n-1)th Iteration :

$$T(n) = T(1) + c(n-(n-1)) + c(n-(n-2)) + ... + (n-1).d$$

$$T(n) = T(1) + c(1+2+3+4.. + (n-1)) + ... + (n-1).d$$

$$T(n) = T(1) + c(\frac{n(n-1)}{2}) + (n-1).d$$

$$T(n) = k + c(\frac{n(n-1)}{2}) + (nd-d)$$

$$T(n) = k + \frac{cn^2}{2} - \frac{cn}{2} + (nd-d)$$

We combine, k, $-\frac{cn}{2}$, -d to a single term to parse the logic better

$$T(n) = \frac{cn^2}{2} + nd + a$$

where a = $k - \frac{cn}{2} - d$
Therefore, the expression we can understand that the term n^2, which increases faster than the other linear terms. Thus proving that $T(n) = O(n^2)$

---

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.