Assignment 2: Asymptotic Notation and Divide and Conquer

CSDS 310: Algorithms

1

For each of the following pairs of functions, write down the asymptotic relation between $f(n)$ and $g(n)$; i.e., if $f(n) \in x(g(n))$ where $x \in \{o, \theta, \omega, O, \Omega\}$. Assume that $k \geq 0, \epsilon > 0, c > 1$. Provide a justification for your answers.

a

$f(n) = \log^k n \quad g(n) = n^\epsilon$

✓ **Answer**

Unsure which would be asymptotically larger, so we will take the limit of its quotient to see its end behavior.

$$\lim_{n\to\infty} \frac{\log^k n}{n^\epsilon}$$
$$\implies \lim_{n\to\infty} \frac{k\log^{k-1} n}{\epsilon n^\epsilon} \text{ Via L'Hopital's Rule}$$
$$\implies \lim_{n\to\infty} \frac{\lceil k\rceil! \log^{k-\lceil k\rceil} n}{\epsilon^{\lceil k\rceil} n^\epsilon} \text{ After } \lceil k\rceil \text{ iterations of L'Hopitals Rule}$$
$$\implies \lim_{n\to\infty} \log^{k-\lceil k\rceil} n = 0$$
$$\implies \lim_{n\to\infty} n^\epsilon \to \infty$$
$$\implies \lim_{n\to\infty} \frac{\lceil k\rceil! \log^{k-\lceil k\rceil} n}{\epsilon^{\lceil k\rceil} n^\epsilon} = 0$$

$\log^k n < cn^\epsilon : n > n_0$
$c = 1$
$log^{k/\epsilon} n < n : n > n_0$
$n_0 > e^{-k/\epsilon W_{-1}(-\epsilon/k)}$ By definition of the Lambert W function

$\therefore \log^k n \in o(n^\epsilon)$
$\square$

b

$f(n) = n^k \quad g(n) = c^n$

Answer

Unsure which would be asymptotically larger, so we will take the limit of its quotient to see its end behavior.

$$\lim_{n\to\infty} \frac{n^k}{c^n}$$
$$\implies \lim_{n\to\infty} \frac{kn^{k-1}}{c^n \ln c} \text{ Via L'Hopital's Rule}$$
$$\implies \lim_{n\to\infty} \frac{\lceil k\rceil ! n^{k-\lceil k\rceil}}{c^n \ln^{\lceil k\rceil} c} \text{ After } \lceil k\rceil \text{ iterations of L'Hopitals Rule}$$
$$\implies \lim_{n\to\infty} \lceil k\rceil ! n^{k-\lceil k\rceil} = 0$$
$$\implies \lim_{n\to\infty} c^n \ln^{\lceil k\rceil} c \to \infty$$
$$\implies \lim_{n\to\infty} \frac{\lceil k\rceil ! n^{k-\lceil k\rceil}}{c^n \ln^{\lceil k\rceil} c} = 0$$

$\therefore n^k \in o(c^n)$
$\square$

c

$f(n) = \sqrt{n} \quad g(n) = n^{\sin n}$

Answer

Unsure how $g(n)$ behaves so we will analyse its limits.
$-1 \leq \sin n \leq 1$
$\implies n \geq n^{\sin n} \geq 0 : n > 1$

$$\lim_{n\to\infty} \frac{0}{\sqrt{n}} = 0$$
$$\lim_{n\to\infty} \frac{n}{\sqrt{n}} \to \infty$$

$f(n)$ is unable to be bounded by $g(n)$ as it oscillates to between $\theta(0)$ and $\theta(n)$ whilst $f(n)$ is asymptotically between those two limits at $\theta(\sqrt{n})$.

2

Considering functions $f(n) \geq 0, g(n) \geq 0, c > 0$, indicate whether each of the following statements is true. Prove the statements that are true by providing a formal argument that is based on the definition of asymptotic notation. For statements that are false, provide a counter-example to prove that they are false.

a

$f(n) \geq 1 \to f(n) + c \in O(f(n))$

✓ **Answer**

True

$$0 \leq f(n) + c \leq 2cf(n)$$
$$\therefore f(n) + c \in O(f(n))$$
$$\square$$

b

$f(2n) \in \theta(f(n))$

✓ **Answer**

$$f(n) = 2^x$$

$$\lim_{n\to\infty} \frac{2^{2x}}{2^x}$$
$$\implies \lim_{n\to\infty} 2^x \to \infty$$

$$\therefore f(2n) \notin \theta(f(n))$$
$$\square$$

c

$f(n) \in O(nc) \to f(2n) \in O(nc)$

✓ **Answer**

$$f(n) \in O(nc)$$
$$\implies 0 \leq f(n) \leq cn$$
$$0 \leq f(2n) \leq 2cn$$
$$0 \leq f(2n) \leq c_1 cn$$

$$\therefore f(2n) \in O(nc)$$
$$\square$$

3

Let $0 < \lambda < 1 < a < b$ be constants. Solve the following recurrences using Master Method, noting the case that applies.

a

$T(n) = bT(n/a) + \theta(n)$

Answer

$c_{crit} = log_a b$
$a < b \implies c_{crit} > 1$
$c = 1$
$\implies c < c_{crit}$

Case 1
$T(n) = \theta(n^{\log_a b})$

b

$T(n) = a^2 T(n/a) + \theta(n^2)$

Answer

$c_{crit} = \log_a a^2 = 2$
$c = 2$
$c_{crit} = c = 2$

Case 1/3
$T(n) = \theta(n^2)$

c

$T(n) = T(\lambda n) + n\lambda$

Answer

$c_{crit} = \log_{\frac{1}{\lambda}} 1 = 0$
$c = 1$
$c > c_{crit}$

Case 3
$T(n) = \theta(n)$

4

You are given an array of k sorted arrays each of which has a length n/k elements. Describe an efficient algorithm to merge these arrays to obtain one sorted array of length n.

✓ **Answer**

```
procedure mergeArrays(k, n, a[k][n/k])
        # Extract the first item and its location in the array and sort them via their value
        # theta(k log(k))
        minimums ← sort(
                [a[i][1], i, 1] for i in 1..k,
                i ⇒ i[1]
                )

        output = Array<int>(n)

        # repeats n times
        for each i in 1..n
                # Remove the minimum from the minimum array and put it into the output array
                min ← minimums.pop(1)
                output[i] ← min[1]
                # Iterate index of minimum in relevant subarray
                newIndex ← min[2] + 1
                # This if statement will only be false k times
                if newIndex ⩽ n/k
                        # Insert a new item into the sorted minimum array using binary insertion
                        # theta(log(k))
                        minimums.insertSort(
                                [a[min[1]][newIndex], min[1], newIndex],
                                i ⇒ i[1]
                                )
        # total loop will be of the complexity of:
        # theta((n-k) log(k) + n) ⇒ theta((n-k) log(k))

        return output
        # overall complexity will be theta((n+k) log(k))
```

Initialization

First we initialize our k sized array the minimums of each subarray, which will be the first elements.

We sort this array in order to find the global minimum

Loop Invariant

The loop invariant is that the `minimums` array is a sorted array that contains the minimum unused value from each subarray, thus guaranteeing that the current unused minimum is first in this array.

Maintenance

First, we remove the minimum of the `minimums` array, this is guaranteed to be the smallest unused value from the entire array as the `minimums` array contains the smallest of each subarray. We append this value into the output array.

Now that we are missing the minimum from the subarray we just moved to the output array, we add the minimum unused value from that subarray back to the `minimums` array, using binary insertion to maintain the sorted status of the `minimums` array.

Termination

The loop will only run n times as this is a for loop. We are guaranteed to go through all values of the original array as once a subarray has been fully depleted, the other arrays will still be in the `minimums` array, and thus will be depleting.

Is is impossible to overdeplete a subarray due to the index check, and if we remove one value from the pool of valid values each iteration, all values will be used by the end of n iterations.

Time Complexity

The most intensive operations within the algorithm are the initial sorting of the k sized array, which is of complexity $\theta(k \log k)$

The loop, which runs n times, has its most intensive operation as the binary insertion, with time complexity of $\theta(\log k)$, which runs all except for k loops. Therefore the entire loop will be of complexity $\theta((n-k)\log k + n)$

When we add the complexities of the entire algorithm together, we are left with $\theta((n+k)\log k)$