classfellow

Date: / / 20

Page No.

Algo HW 6

Q1)

A1) By definition, $f(n) = O(g(n))$ means there exists a constant $c > 0$ and an integer n_0 such that for all $n \geq n_0$,

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

Using inequality, $|f(n) + g(n)| \leq |f(n)| + |g(n)|$.

$$|f(n) + g(n)| \leq C \cdot |g(n)| + |g(n)|.$$

$$|f(n) + g(n)| \leq (C+1)\, |g(n)|$$

Since, $|f(n) + g(n)| \leq (C+1) \cdot |g(n)|$ for all $n \geq n_0$, we have shown that $f(n) + g(n) = O(g(n))$.

Here, k can be chosen as $C+1$, satisfying Big-O definition.

Q 2)

A2)

a) For $a(n) = \Theta(d(n)e(n))$,

from this $a(n)$ grows at same rate as product $d(n)e(n)$.

So, $a(n) \approx d(n)\, e(n)$. —(1)

b) For $a(n)d(n)e(n) = \Theta(n)^3$

From (1), $a(n) \approx d(n)\,(e(n))$. Substituting,

$$(d(n)e(n))(d(n)e(n)) = \Theta(n^3)$$

$$d(n)^2\, e(n)^2 = \Theta(n^3) \quad \text{—(2)}$$

Growth of $d(n)^2 (e(n)^2)$ is comparable to n^3.

c) For $b(n)^3 = \Omega(a(n)^2)$.

$b(n)^3$ grows at-least as fast as $a(n)^2$.

Substituting, $a(n)^2 \approx d(n)^2\, e(n)^2$ and $b(n)^3 = \Omega(d(n)^2 e(n)^2)$

$= b(n)^3 = \Omega(n^3)$, which gives

$b(n) = \Omega(n)$.

classfellow

Date: / / 20

Page No.

d) For $c(n)+d(n) = \Theta(b(n)^2)$.
($c(n)+d(n)$) grows asymptotically at same rate as $b(n)^2$.
Since, $b(n) = \Omega(n)$.
$b(n)^2 = \Omega(n^2)$.
So, $c(n)+d(m) = \Theta(n^2)$

e) For $d(n)^2 = \Theta(a(n)e(n))$.
We have $a(n) \approx d(n)e(n)$
So, $a(n)(e(n)) \approx d(n)\, e(n)^2$
So, $d(n)^2 \approx d(n)\, e(n)^2$
which gives $e(n)^2 \approx d(n)$

f) For $e(n)^2 = \Omega(b(n))$
We have, $b(n) = \Omega(n)$
So, $e(n)^2 = \Omega(n)$
So, $e(n) = \Omega(n)^{1/2}$

Q3→
A3→ Outer 'for' loop (line 3-11),
Iterates from $i=1$ to $i=n$, so it will run 'n' times.
Inner 'for loop (line 6-8),
For each value of j in 'while' loop, it runs $k=j$ to $k=n$
around $(n-j)$ times.
For 'while' loop (line 5-10),
For each value of i, variable j starts at i & is doubled
($j = 2*j$) on each iteration.
Loop condition is $i, 2i, 4i, 8i$ ------- until j exceeds (equals) n.
No of iterations of this 'while' loop is approximately
$O(\log(\frac{n}{i}))$, as j grows exponentially by '2' with
each iteration.

Outer 'for' loop = Runs 'n' times.
Inner 'while' loop = Runs 'log$(\frac{n}{i})$' times

Inner 'for' loop = Runs 'n - j' times.
Total no of iterations of m = m+1.
So, total iterations,

$$\sum_{i=1}^{n} \sum_{\text{while loop } j} (n-j).$$

So, time complexity is = $O(n^2 \log n)$.

Q4)
A4) Base case, $n=1$ (algorithm returns, as element is already sorted)
$n=2$ (checks if swap is needed, then returns sorted array)

Recursive calls (when $n>2$):
Algorithm calculates 'third' as $\lfloor n/3 \rfloor$.
It calls recursively 'ThirdSort' on line 11, 12, 13.
Each call handle $(\frac{2n}{3})$ subproblem size.
For $n>2$,

$$T(n) = 3T\left(\frac{2n}{3}\right) + O(1)$$

with base case, $T(1) = O(1)$

$$T(2) = O(1).$$

Q5)
A5) Recurrence for ThirdSort in Q4, $T(n) = 3T\left(\frac{2n}{3}\right) + O(1)$
In this $a = 3$, $b = \frac{3}{2}$, $f(n) = O(1)$.
Calculate $\log_b^a$,

$$\log_{\frac{3}{2}}(3) = \frac{\ln(3)}{\ln(3/2)}$$

This is case 1 of master Theorem,

$$f(n) = O(n^{\log_b(a) - \epsilon}) \text{ for } \epsilon > 0,$$

then $T(n) = O(n^{\log_b(a)})$.

classfellow

CPPL Date: / / 20

Page No.

So, $T(n) = O(n^{\log_{3/2}(3)})$.

$T(n) = O\, n^{\log_{3/2}(3)}$

So, $n^{\log_{3/2}(3)} \approx n^{1.71}$

So, $T(n) = \Theta(n^{1.71})$.

Selection Sort has time complexity of $O(n^2)$ in worst case. Third sort has $\Theta(n^{1.71})$, which is somewhat faster than $O(n^2)$ but slower than $O(n \log n)$.

Q6→

A6→

a) We will use hash map that tracks frequency of each element. This allows quick updates to frequencies when elements are added or removed.

We will also use max-heap that stores frequency of each element, making it possible to retrieve element with highest frequency in $O(1)$ time.

We can also use map of elements with max frequency and counter for current mode.

b) Algorithm: get mode

Input: none

Output: modeElement, element that appears most frequently

return modeElement.

It takes $O(1)$ time, worst-case.

c) Algorithm: addElement

Input: value, the element to add

Output: none

if value is in frequency map, then

frequencymap[value] += 1.

else

frequencymap[value] = 1

classfellow

Date: / / 20

Page No.

end if.

maxHeap. insert (frequency map[value]).

If frequency map[value] > mode frequency then

mode Element = value

mode Frequency = frequency map[value]

end if.

It takes O(log(n)) time, worst-case.

d)

Algorithm: removeElement

Input: value, the element to remove

Output: None.

if value is in frequencyMap then

frequency Map[value] -= 1

maxHeap. remove(frequency Map[value]+1).

if frequency Map[value] = 0, then

remove value from frequency Map

end if.

if frequency Map [mode Element] < maxHeap. peek() then

mode Frequency = maxHeap. peek()

mode Element = getElementWithFrequency(mode Frequency).

end if

end if.

This takes O(log(n)) time, worst-case.