Homerwork3

Tuesday, February 22, 2022 8:25 PM

1. Recurrence

a) Algorithm A:

$$T(n) = 4T\left(\frac{n}{2}\right) + O(n)$$

"four subproblems" — "half the size" — "combining solutions in linear time"

Using the Master Theorem:

$a = 4$, $b = 2$, $d = 1$

$a = 4 > b^d = 2$

$\Rightarrow T(n) = O(n^{\log_2 4}) = O(n^{\log_2 2^2}) = \boxed{O(n^2)}$

For Algorithm B:

$$T(n) = 2T(n-1) + O(1)$$

$$= 2(2T(n-2) + O(1)) + O(1) = 2^2 T(n-2) + (2^1 + 2^0) O(1)$$

$$= 2(4T(n-2) + 2O(1) + O(1)) + O(1) = 2^3 T(n-2) + (2^2 + 2^1 + 2^0) O(1)$$

$$= 2^n T(1) + (2^{n-1} + 2^{n-2} + \dots + 2^0) O(1)$$

(geometric series)

$$= \sum_{k=0}^{n-1} 2^k = \frac{2^n - 1}{2 - 1} = 2^n - 1$$

$$\Rightarrow T(n) = 2^n T(1) + (2^n - 1) O(1)$$

$$\Rightarrow T(n) = O(2^n) + O(2^n)$$

$$\Rightarrow \boxed{T(n) = O(2^n)}$$

For Algorithm C:

$$T(n) = 5T\left(\frac{n}{3}\right) + O(n^2)$$

"dividing into 5 problems" — "of size $\frac{n}{3}$" — "combining solutions in $O(n^2)$ time"

$\Rightarrow$ Using Master's Theorem:

$a = 5$, $b = 3$, $d = 2$

$a = 5 < b^d = 9$

$\Rightarrow T(n) = \boxed{O(n^2)}$

$\Rightarrow$ If we want to solve the problem efficiently, we can either use algorithms A or C since they both have a time complexity of n^2, and they are therefore more time efficient than algorithm B, which runs in $O(2^n)$ time, and $\lim_{n \to \infty} \frac{2^n}{n} = \infty$

b) $T(n) = T(n-1) + 3 \log n$

$= (T(n-2) + 3 \log(n-1)) + 3 \log n$

b) • $T(n) = T(n-1) + 3\log n$

$= \left(T(n-2) + 3\log(n-1)\right) + 3\log n$

$= (T(n-3) + 3\log(n-2)) + 3\log(n-1) + 3\log n$

$= T(1) + 3(\log 2 + \log 3 + \log 4 + \dots + \log(n-1) + \log n)$

$= T(1) + 3\log(2 \times 3 \times 4 \times \dots \times n-1 \times n)$

$= 1 + 3\log(n!)$

$\Rightarrow T(n) = O(\log(n!))$

• $T(n) = (\log n)\, T\left(\frac{n}{2^1}\right) + 1$

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

$= (\log n)(\log n - 1)(\log n - 2)\, T\left(\frac{n}{2^3}\right) + (\log n)(\log n - 1) + (\log n) + 1$

$= (\log n)(\log n - 1)(\log n - 2)(\log n - 3)\, T\left(\frac{n}{2^4}\right) + (\log n)(\log n - 1)(\log n - 2) + (\log n)(\log n - 1) + \log n + 1$

$= (\log n)(\log n - 1) \times \dots \times 1 \times T(1) + (\log n)(\log n - 1) \times \dots \times 2 + (\log n) \times (\log n - 1) \times \dots \times 3 + (\log n)(\log n - 1) \times \dots \times 4 + \log n + 1$

$= (\log n)! + \frac{(\log n)!}{1!} + \frac{(\log n)!}{2!} + \frac{(\log n)!}{3!} + \dots + \frac{(\log n)!}{(\log n - 1)!} + \frac{(\log n)!}{(\log n)!}$

$= O((\log n)!)$

2. Sort

a) Worst case time complexity: $O(n^2)$

This occurs when the array is sorted in reverse. In that case, everytime a new index is reached when iterating forward, we would have to iterate again to the beginning of the array because the next element will always be smaller than all the previous elements.

Best case time complexity: $O(n)$

This occurs when the input is a sorted array. In this case, the algorithm would only have to make one forward iteration through all n elements.

b) The recurrence relationship is:

$$T(n) = \underbrace{2T\left(\frac{n}{2}\right)}_{\text{two recursive calls on an input with size } \frac{n}{2}} + \underbrace{T(n-1)}_{\text{One recursive call on an input with size } n-1.} + \underbrace{O(1)}_{\text{remaining operations happen in constant time.}}$$

The time complexity of this algorithm does not change no matter the input. The only line that would not be evaluated if the input is already sorted is the one that swaps the two elements, and since it runs in constant time, the time complexity of the algorithm will always be the same.

To find $T(n)$:

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

Using the Master Theorem:

$a = 2,\ b = 2,\ d = 0$

$a > b^d$

$\Rightarrow \delta(n) = O(n^{\log_2 2})$

$= O(n)$

$\Rightarrow T(n) = T(n-1) + n$

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

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

$= T(1) + 2 + 3 + 4 + \dots + (n-1) + n$

$= 1 + 2 + 3 + 4 + \dots + n - 1 + n$

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

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

$= O(n^2)$

Therefore the time complexity of this algorithm is always $O(n^2)$, so in the best case it is slower than bubble sort which can run in $O(n)$ time for a sorted array.