

You may want to embed an image. If so, use a figure code block.

$$T(n) = T(n-1) + O(1) \quad (16)$$

$$= T(n-2) + 2 * O(1) \quad (17)$$

$$= T(n-3) + 3 * O(1) \quad (18)$$

$$= T(n-4) + 4 * O(1) \quad (19)$$

$$\dots \quad (20)$$

$$= T(n-n) + n * O(1) \quad (21)$$

$$= T(0) + n * O(1) \quad (22)$$

$$= 1 + n * O(1) \quad (23)$$

$$= O(n) \quad (24)$$

4. (15 points) What is the best, worst and average running time of quick sort algorithm?

Best case:

$$O(n \log(n)) \quad (25)$$

Average case:

$$O(n \log(n)) \quad (26)$$

Worst case:

$$O(n^2) \quad (27)$$

(a) Provide recurrence relations. For the average case, you may assume that quick sort partitions the input into two halves proportional to c and $1 - c$ on each iteration.

Best:

$$T(n) = \begin{cases} 2T(n/2) + O(n), & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases} \quad (28)$$

Average:

$$T(n) = \begin{cases} T(cn) + T((1-c)n) + n, & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases} \quad (29)$$

Worst:

$$T(n) = \begin{cases} T(n-1) + n, & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases} \quad (30)$$

(b) Solve each recurrence relation you provided in part (a)

Will occur $\log n$ times

$$\begin{cases}
 2T(n/2) + n \cdot 1 = T(n) \\
 4T(n/4) + 2 \cdot \frac{1}{2}n + n = 4T(n/4) + 2n \\
 8T(n/8) + 4 \cdot \frac{1}{4}n + 2 \cdot \frac{1}{2}n + n = 8T(n/8) + 3n \\
 \dots
 \end{cases}$$

$$nT(n/n) + n \log n = n \cdot 0 + n \log n = n \log n$$

is $O(n \log n)$

Figure 4: Solution to best case recurrence relation

□ indicates important

○ = answer

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

$$T(c^2n) + T((1-c)^2n) + 2n$$

Case of $c > \frac{1}{2}$

$$T(c^k n) + T((1-c)^k n) + kn$$

$$n = \left(\frac{1}{c}\right)^k, \quad k = \log n \quad \text{and} \quad \left(\frac{1}{c}\right)^k = n$$

$$T(1) + T((1-c)^k n) + k\left(\frac{1}{c}\right)^k = 0 + 0 + k\left(\frac{1}{c}\right)^k = n \log n$$

If $c > \frac{1}{2}$ $T((1-c)^k n)$ will have already reached 0 because $(1-c)^k < c^k$ and the ratio of the two will be < 1 .

Case of $c < \frac{1}{2}$

$$n = \left(\frac{1}{1-c}\right)^k, \quad k = \log n, \quad \left(\frac{1}{1-c}\right)^k = n$$

$$T\left(\left(\frac{1}{1-c}\right)^k\right) + T(1) + k\left(\frac{1}{1-c}\right)^k = 0 + 0 + k\left(\frac{1}{1-c}\right)^k = n \log n$$

If $c < \frac{1}{2}$ $T(cn)$ will have already reached 0 because $c^k < (1-c)^k$ and the ratio of the two will be < 1 .

Case of $c = \frac{1}{2}$

This is just best case scenario, which was calculated to be $n \log n$

Final

Since all three cases were $n \log n$, the average runtime complexity function must be $n \log n$.

Figure 5: Solution to average case recurrence relation

$$\begin{aligned}
 T(n) &= T(n-1) + n \\
 &\quad T(n-2) + 2n \\
 &\quad T(n-3) + 3n \\
 &\quad T(n-4) + 4n \\
 &\quad \dots \\
 &\quad T(n-n) + n^2 \\
 &\quad 0 + n^2
 \end{aligned}$$

$T(1) = 0$

is $O(n^2)$

Figure 6: Solution to worst case recurrence relation

- (c) Provide an arrangement of the input array which results in each case. Assume the first item is always chosen as the pivot for each iteration.

Best	{5, 2, 3, 4, 1, 7, 8, 9, 6}
Average	{9, 6, 7, 4, 5, 8, 3, 1, 2}
Worst	{1, 2, 3, 4, 5, 6, 7, 8, 9}

5. (15 points) Write a C++ function that counts the total number of nodes with two children in a binary tree (do not count nodes with one or none child). You can use a STL container if you need to use an additional data structure to solve this problem.

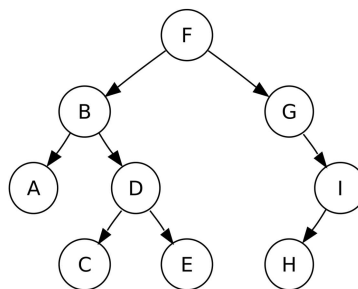


Figure 7: Calling `count_filled_nodes` on the root node F returns 3

- (a) Complete the function below. The function will be called with the root node (e.g. `count_filled_nodes(root)`). The tree may be empty. Do not modify the tree.

```

1 #include <vector>
2

```