1. Derive asymptotic upper bounds of work for each recurrence below.

W(n)=2W(n/3)+1

Level 0 = 1

Level 1 = 2

Level 2 = 4

Leaf dominated

Work at leaf level 2^log_3(n)

Thus O(2^log_3(n)) → O(n^log_3(2))

W(n)=5W(n/4)+n

Level 0 = n

Level 1 = 5/4 n

Level 2 = 25/16 n

Leaf Dominated

Leaf level = 5^(log_4 n)/ 4^(log_4 n) * n

(5/4) ^ log_4 n = n ^ log_4(5/4) * n

n^(log_4 5 - 1) * n = n^(log_4(5) -1 +1)

= n ^log_4 5

. O(5^(log_4 n)/ 4^(log_4 n)) = O(n^log_4 5)

.

.

.

W(n)=7W(n/7)+n

. Level 0 = n

. Level1 = n

Level 2 = n

Balanced

. depth = logn

. O(n logn)

.

* W(n)=9W(n/3)+n^2

. Level 0 = n^2

. Level1 = n^2

Level 2 = n^2

Balanced

. Depth = log_3 n

. O(n^2 logn)

.

* W(n)=8W(n/2)+n^3

. Level 0= n^3

. level1 = n^3

Level 2 = n^3

. Depth = log_2(n)

.   O(n^3 logn)

.

W(n)=49W(n/25)+n^{3/2}\log n

. Level 0 = n^3/2 * logn

. level 1 = (n^3/2)/125 * log(n/25)

Root dominated

. O(n^3/2 * logn)

.

.

W(n)=W(n-1)+2

. level 0: 2

. level 1 : 2

Depth is n

O(2n) -> O(n)

Thus O(n)

.

W(n)= W(n-1)+n^c, with c >= 1

. The depth here is n

work lvl 0: n^c

work lvl 1: (n-1)^c

work lvl 2: (n-2)^c.

While the work at each level decreases, it does not decrease geometrically and thus each level's work can be treated as n^c. Thus the total work is n^c * n. Thus, O(n^(c+1))

.

.

W(n)=W(\sqrt{n})+1

Because the input is decreased by a square root, not every input results in a nice base case. Most result in a floating number. The tree depth is n^(1/2k). When we want the final node to 1, we have to set n^(1/2k) = 1. We can take the log_base_n of each side to get 1/2k = 0 (because log base anything of 1 equals 0). However, this is only true as k = infinity. Thus our base case will be different for each input. So we can assume O(loglogn).

1. A. W(n) = 5W(n/2) + O(n) ->O(n^log_2(5))
    a. Leaf dominates as the work increases per level. To find total work, we must take the total work at the last level. Thus, work is O(n^log_2(5))
2. S(n) = S(n/2) + O(n) -> O(n)

a. Span here is just the longest branch. Depth of the tree is log_2(n). Because the input is halved and f(n) = n, the span is n + n/2 + n/4 + n/8… As this sum asymptotically approaches infinity, the span equals 2n. Thus O(n).

B. W(n) = 2W(n-1) + O(1) -> O(2^n)

Here, the function is leaf dominated. Thus to calculate the total work, its the work at each node times the number of nodes at the last level. The tree depth is n because W(n-1). The work at each node is 1. As each level decreases, the number of nodes doubles. As it goes down to n levels, the number of nodes becomes 2^n. Thus O(2^n)

S(n) = S(n-1) + O(1) -> O(n)

We know the depth to be n. The work at each node is 1. Span is the longest branch so 1 node per level. Thus O(n).

C. W(n) = 9W(n/3) + O(n^2) -> O(n^2logn)

Here the function is balanced. So the total work will have to be the work at each level multiplied by the tree depth. The tree depth here is log_3(n) and the work at each level is n^2. Thus O(n^2logn)

S(n) = S(n/3) + O(n^2) → O(n^2)

The span is the work of the longest branch. With S(n/3) + O(n^2), each level's input (n) decreases as n/3. Thus the span is root dominated and equals O(n^2)

Which Algorithm would I choose?

Time is money, so I want to choose the algorithm with the lowest span (assuming Infinite processors). This leaves us with algorithms A and B. The total work for A grows more slowly than the total work for algorithm B. Assuming I have many inputs (more than 6), I prefer algorithm A.