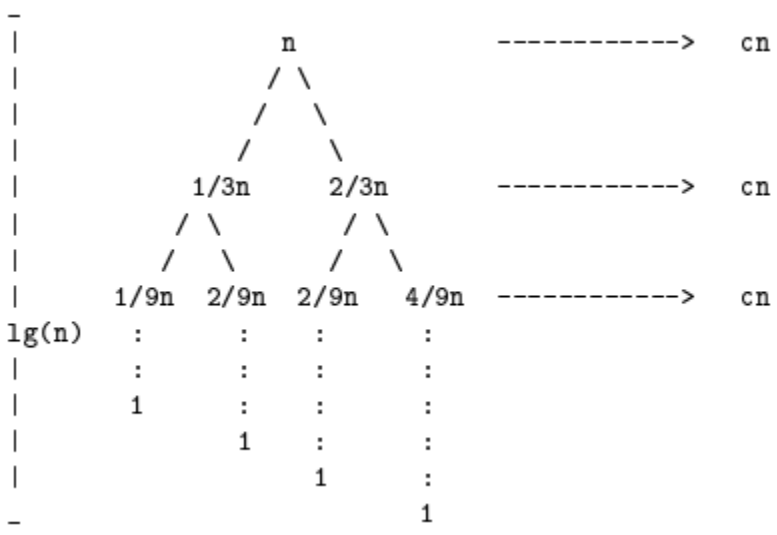
Solution 1 (a)

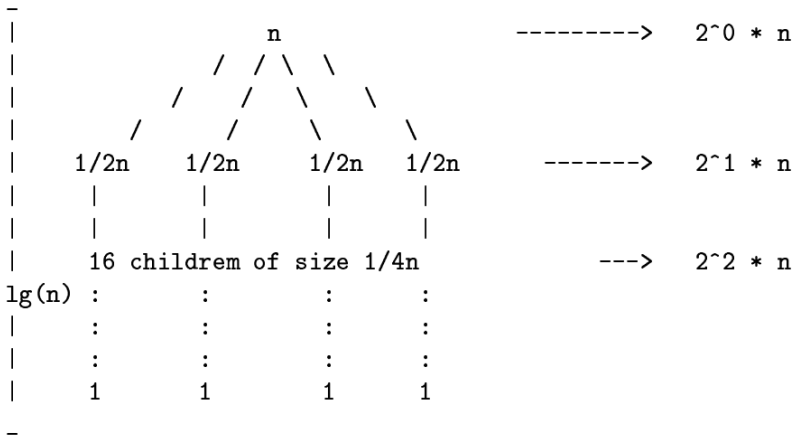
Drawing the recursion tree (shown on the next page) we see that the cost at every level is $\leq cn$, so the total cost is that multiplied by the height of the tree. At worst this height is $log_{3/2}n$, for which we have the identity: $\log_{3/2} n = \frac{\lg n}{\lg 3/2} = d \lg n$, where $d = \lg 3/2$. This gives us an upper bound on the runtime of $(cd)n \log n = O(n \lg n)$.

To obtain the lower bound we consider the shortest path in the recursion tree from the root to a leave. The depth of this shortest path is $\log_3 n$. Since all the levels up to this leave are full and have a cost of cn, using a similar base conversion as above, we obtain a lower bound on the runtime of $\Omega(n \log n)$ as well.



(b)

The following picture illustrates the recursion tree for the given recurrence relation.



From the recursion tree one can see that the tree has $\log n$ levels, and that at the i-th level the total work performed is $2^i n$. Thus, $T(n) = \sum_{i=0}^{\log n} 2^i n$.

To obtain a tight bound one can either evaluate this sum or apply the Master method. Since $a = 4$, $b = 2$, and $f(n) = n$, by setting $\varepsilon = 1$ we can see that case 1 is applicable, and hence $T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)$.

Solution 2

Hint : Write a method based on the following recursive formulation of 3^n **carefully**.

$$3^n = \begin{cases} 1 & \text{if } n = 0 \\ 3^{n/2} * 3^{n/2} & \text{if } n\%2 == 0 \\ 3^{n/2} * 3^{n/2} * 3 & \text{if } n\%2 == 1 \end{cases}$$

Solution 3

Think about the array B such that B**[i] =** A**[i]** − 2i . Given that A consists of n distinct even integers, the array B is also sorted. The idea is to carry out a binary search in B in order to locate the element **0**. An explicit construction of B, however, takes O(n) time. We instead work with the array A itself. Whenever we visit the i -th location in A , we compute B**[i] =** A**[i]** − 2i (or compare A**[i]** with 2i). Since the binary search visits **O(log** n**)** locations only, the elements of B are computed only **O(log** n**)** times.

Solution 4

We use the following divide-and-conquer approach to solve this problem. Here, we assume that pointer arithmetic can be used. If not, the following code can be easily rewritten by passing the start and last indices in a subarray.

```
void allbutme ( float *A, int n, float *B )
{
        int i, m, M;
        float s, t;
        if (n == 1) { B[0] = 1; return; }
        m = n / 2 ;
        allbutme(A, m, B);
        allbutme(A + m, n - m, B + m);
        s = A[0] * B[0]; /* We have s = a0a1 . . . am−1 */
        t = A[m] * B[m]; /* We have t = amam+1 . . . an−1 */
        for (i = 0; i < m; ++i) B[i] *= t;
        for (i = m; i < n; ++i) B[i] *= s;
}
```

Complexity

T(n) = 2T(n/2)+O(n)
This gives T(n) = O(nlogn)

Evaluation Notes: Please see the highlighted step. If any solution uses a loop for doing this, the solution has been definitely copied. In that case, 0 marks should be given.

The question did not forbid division by integers, but the solution on the internet did not allow division by integers, and hence a loop would have been used instead of the division.