Exercise 1

Exercises for Algorithms by Nengjun Zhu.

Name: 郭同 **Student ID:** 24721537 **Email:** teri@shu.edu.cn

答案 AI 含量声明: 第四题将 c++ 代码转换为伪代码, 第五题步骤 e

1. Suppose $a_0 = 1, a_1 = 2, a_2 = 3, a_k = a_{k-1} + a_{k-2} + a_{k-3}$ for $k \geq 3$. Use strong principle of mathematical induction to prove that $a_n \leq 2^n$ for all integers $n \geq 0$.

   Proof: When $0 \leq n \leq 4$, the conclusion is obviously true.

   Induction Step: For $k \geq 5$, assume $a_k \leq 2^k$, then we have:

$$\begin{aligned} a_{k+1} &= a_k + a_{k-1} + a_{k-2} \\ &\leq 2^k + 2^{k-1} + 2^{k-2} \\ &= 2^{k-2}(4+2+1) \\ &\leq 2^{k+1}. \end{aligned}$$

   Conclusion: Therefore, the conclusion holds.

2. Consider the sorting algorithm shown in Alg.1, which is called BUBBLESORT.

   (a) What is the minimum number of element comparisons? When is this minimum achieved?

   Answer: The minimum number of comparisons is $n-1$ when the elements are already in ascending order.

   (b) What is the maximum number of element comparisons? When is this maximum achieved?

   Answer: The maximum number of comparisons is $\frac{n(n+1)}{2}$ when the elements are in descending order.

   (c) Express the running time of Alg.1 in terms of the O and Ω notations.

   Answer: Let $g(n) = n - 1$, $h(n) = \frac{n(n+1)}{2}$, thus $f(n) = O(h(n))$, $f(n) = \Omega(g(n))$. The time complexity is $O(n^2)$ and $\Omega(n)$.

   (d) Can the running time of the algorithm be expressed in terms of the Θ notation? Explain.

   Answer: In the average case, the elements are randomly ordered, requiring about half the swaps, so the time complexity remains $\Theta(n^2)$. Both the worst-case and average-case complexities are $\Theta(n^2)$, making this notation suitable.

3. Fill in the blanks with either true (T) or false (F):

$f(n)$	$g(n)$	$f = O(g)$	$f = \Omega(g)$	$f = \Theta(g)$
$2n^3 + 3n$	$100n^2 + 2n + 100$	F	T	F
$50n + \log n$	$10n + \log\log n$	T	T	T
$50n \log n$	$10n \log\log n$	F	T	F
$\log n$	$\log^2 n$	T	F	F
$n!$	5^n	F	T	F

4. Design a divide-and-conquer algorithm to determine whether two given binary trees T_1 and T_2 are identical.

```
Input: Two binary trees, root1 and root2
Output: Boolean value indicating whether the trees are identical
function isSameTree(root1, root2)
    if root1 = NULL and root2 = NULL then
        return true
    end if
    if root1 = NULL or root2 = NULL then
        return false
    end if
    if root1.val ≠ root2.val then
        return false
    end if
    return isSameTree(root1.left, root2.left) and isSameTree(root1.right, root2.right)
end function
```

5. You are given two sorted lists of size m and n in ascending order. Give an $O(\log m + \log n)$ time algorithm for computing the k-th smallest element in the union of the two lists.

Algorithm Idea

1. **Base Case Handling**:
   - If one of the lists is empty, the k-th smallest element is simply the k-th element of the other list.
   - If $k = 1$, return the smaller of the first elements of the two lists.
2. **Recursive Strategy**:
   - Assume the length of list A is always less than or equal to the length of list B (swap if necessary). This ensures the first list is never longer than the second, optimizing recursive calls.
   - Choose $i = \min(\frac{k}{2}, m)$ and $j = \min(\frac{k}{2}, n)$ as the midpoints to split the lists A and B.
   - Compare $A[i-1]$ and $B[j-1]$:
     * If $A[i-1] < B[j-1]$, it means that the k-th smallest element cannot be in the first i elements of A. Recursively search for the $(k-i)$-th smallest element in the remaining part of A and the entire B.

* Otherwise, the k-th smallest element cannot be in the first j elements of B. Recursively search for the $(k-j)$-th smallest element in A and the remaining part of B.

3. **Time Complexity**:

– The algorithm halves the search space in each recursive step, achieving a time complexity of $O(\log m + \log n)$.

Detailed Steps

1. **Input:** Two sorted lists A of size m and B of size n, and an integer k.
2. **Output:** The k-th smallest element in the union of A and B.
3. **Procedure:**
   (a) If $m > n$, swap A and B to ensure A is the smaller list.
   (b) If $m = 0$, return $B[k-1]$.
   (c) If $k = 1$, return $\min(A[0], B[0])$.
   (d) Set $i = \min(\frac{k}{2}, m)$ and $j = \min(\frac{k}{2}, n)$.
   (e) Compare $A[i-1]$ and $B[j-1]$:
      – If $A[i-1] < B[j-1]$, recursively find the $(k-i)$-th smallest element in $A[i:]$ and B.
      – If $A[i-1] \geq B[j-1]$, recursively find the $(k-j)$-th smallest element in A and $B[j:]$.