Exercis 1

Exercises for Algorithms by Nengjun Zhu.

Name: ___郭同___   **Student ID:** ___24721537___   **Email:** ___teri@shu.edu.cn___

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 $n \geq 0$.

   证： 当 $0 \leq n \leq 4$ 时，结论显然成立。

   假设 对于 $5 \leq k$，假设有 $a_k \leq 2^k$，则我们有：

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

   结论 因此结论成立。

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?

   答： 当元素为升序时比较次数最少, 为 $n-1$ 次

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

   答： 当元素为降序时比较次数最多, 为 $\frac{n(n+1)}{2}$ 次

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

   答： 令 $g(n) = n-1$, $h(n) = \frac{n(n+1)}{2}$, 可得 $f(n) = O(h(n))$, $f(n) = \Omega(g(n))$, 时间复杂度为 $O(n^2), \Omega(n)$。

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

   答： 平均情况下，数组元素是随机排列的，需要进行大约一半的交换操作，时间复杂度依然是 (n^2)。无论是平均情况还是最坏情况，冒泡排序的时间复杂度都是 (n^2)。因此，Θ 表示法是合适的。

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.

Algorithm 1 isSameTree: Check if two binary trees are identical

---

1: **Input:** Two binary trees, *root1* and *root2*

2: **Output:** Boolean value indicating whether the trees are identical

3: **function** isSameTree(root1, root2)

4:      **if** root1 = NULL and root2 = NULL **then**

5:          **return** true

6:      **end if**

7:      **if** root1 = NULL or root2 = NULL **then**

8:          **return** false

9:      **end if**

10:      **if** root1.val $\neq$ root2.val **then**

11:          **return** false

12:      **end if**

13:      **return** isSameTree(root1.left, root2.left) and isSameTree(root1.right, root2.right)

14: **end function**

---

5. You are given two sorted lists of size m and n in ascending order. Given 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 $B[j:]$.