高等理工学院《算法设计与分析》
(2021 年秋季学期)

第一次作业参考答案

1 请给出 $T(n)$ 尽可能紧凑的渐进上界并予以说明，可以假定 n 是 4 的整数次幂。(每小题 3 分，共 21 分)

1.

$$T(1) = 1$$
$$T(n) = 3T(n/2) + n^2 \quad if \quad n > 1$$

2.

$$T(1) = 1$$
$$T(n) = 4T(n/2) + n^2 \quad if \quad n > 1$$

3.

$$T(1) = 1$$
$$T(n) = T(n/2) + 2^n \quad if \quad n > 1$$

4.

$$T(1) = 1$$
$$T(n) = 16T(n/4) + n \quad if \quad n > 1$$

5.

$$T(1) = 1$$
$$T(n) = 2T(n/2) + n\log n \quad if \quad n > 1$$

6.

$$T(1) = 1$$
$$T(n) = \sqrt{2}T(n/2) + \log n \quad if \quad n > 1$$

7.

$$T(1) = 1$$

$$T(n) = T(n/2) + 2n \quad if \quad n > 1$$

答案:

1. $T(n) = O(n^2)$
2. $T(n) = O(n^2 \log n)$
3. $T(n) = O(2^n)$
4. $T(n) = O(n^2)$
5. $T(n) = O(n \log^2 n)$
6. $T(n) = O(\sqrt{n})$
7. $T(n) = O(n)$

2 部分有序数组排序问题 (19 分)

基于比较的排序算法时间复杂度下限为 $O(n \log n)$，但是限制输入数组部分有序可以降低算法的时间复杂度。

给定一个部分有序的数组 $A[1..n]$，和一个正整数 k，可以假定 $k|n$，满足 $a[i] \leq a[i+k], 1 \leq i \leq n-k$。

例如，数组 $[1, 2, 3, 6, 5, 8, 9, 10]$ 是 $k = 2$ 的部分有序数组。

请设计一个最差时间复杂度为 $n \log k$ 算法，对输入数组进行排序，使得该数组满足 $a[i] \leq a[i+1], 1 \leq i \leq n-1$。

答案:

观察题目条件可以发现 $a_1 \leq a_{k+1} \leq \cdots a_{t*k+1}$，$\cdots$，$a_{k-1} \leq a_{k+k-1} \leq \cdots a_{t*k+k-1}$，可以得到 k 个长度为 $\frac{n}{k}$ 的有序数组，该问题转化为合并 k 个有序数组。

一种高效的算法是把 k 个有序数组平均分为两份递归进行合并得到两个数组，然后再合并这两个数组, 算法实现请参考 **Algorithm** 1。

采用这种策略，可将原规模为 k 的问题分解为两个规模为 $k/2$ 的子问题，合并两个子问题的时间为 $O(n)$。故可列出递归式 $T(k) = 2T(k/2) + O(n)$。解得时间复杂度为 $T(n) = O(n \log k)$。

Algorithm 1 $k\_Merge(A, l, r)$

Input:
k 个有序数组, $A = \{A_1, A_2, \cdots, A_k\}$
递归区间左端点, l
递归区间右端点, r

Output:
归并后的有序数组

if $l = r$ **then**
  return A_l
end if
$m \leftarrow \lfloor \frac{l+r}{2} \rfloor$
return $Merge(k\_Merge(A, l, m), k\_Merge(A, m+1, r))$

3 局部最大值问题 (20 分)

给定一个由 $n(n \geq 3)$ 个互不相同的整数组成的数组 $A[1..n]$，其满足 $A[1] < A[2]$ 并且 $A[n-1] > A[n]$。我们定义数组的**局部最大值**为比它的两个相邻元素 (如果存在) 都大的整数。换言之，$A[x]$ 是局部最大值当且仅当它满足 $A[x] > A[x-1]$ 并且 $A[x] > A[x+1] (1 < x < n)$。例如，下图所示数组中存在两个个局部最大值，分别为 6 和 10。

2	3	6	5	7	10	1

求局部最大值显然有一个 $O(n)$ 的做法，仅需要扫描一遍整个数组就可以找到所有的局部最大值。请你给出一个算法可以在 $O(\log n)$ 的时间复杂度内找出一个数组的局部最大值。如果局部最大值有多个，仅需要找出任意一个局部最大值即可。(提示：我们给出的限制条件保证数组至少有一个局部最大值。)

答案：

如果 n 等于 3，做法是显然的。考虑 n 大于 3 的情况, 此时令 $m = \lfloor \frac{n}{2} \rfloor$，来检查 $A[m-1], A[m], A[m+1]$ 之间的大小关系：

1. 如果 $A[m-1] < A[m]$ 并且 $A[m] > A[m+1]$，那么 A[m] 就是局部最大值，算法结束。
2. 如果 $A[m-1] > A[m] > A[m+1]$，那么 $A[1..m]$ 中一定存在局部最大值，我们递归处理数组 $A[1..m]$ 即可。
3. 如果 $A[m-1] < A[m] < A[m+1]$，那么 $A[m..n]$ 中一定存在局部最大值，我们递归处理数组 $A[m..n]$ 即可。
4. 如果 $A[m-1] > A[m]$ 并且 $A[m] < A[m+1]$，那么左右两个区间中都存在局部最大值，我们任选一个区间递归处理即可。

任何一种情况每次递归都缩减了一半的规模，因此可以得出递归式 $T(n) \leq T(n/2) + O(1)$，解出算法的时间复杂度为 $T(n) = O(\log n)$。

4 递归求和问题 (20 分)

给定一个整数集合 $A = \{a_1, a_2, \cdots, a_n\}$，每次可以对集合 A 进行以下操作，首先计算数组最大值与最小值的均值 $mid = (max(A) + min(A))/2$，然后进行以下两个操作之一：

1. 保留集合中小于 mid 的元素，即 $A = \{a_i | a_i \in A, a_i < mid\}$；
2. 保留集合中大于等于 mid 的元素，即 $A = \{a_i | a_i \in A, a_i \geq mid\}$。

给定整数 S，请设计一个高效算法，计算能否通过若干次操作，使得 $\sum_{a_i \in A} a_i = S$，并分析该算法的时间复杂度。

例如，给定 $S = 3$，集合 $A = \{9, 3, 1, 7\}$ 通过以下操作可以满足 $\sum_{a_i \in A} a_i = S$：

1. 计算 $mid = (1+9)/2 = 5$，保留集合中小于 5 的元素，$A = \{1, 3\}$；
2. 计算 $mid = (1+3)/2 = 2$，保留集合中大于等于 2 的元素，$A = \{3\}$。

答案：

该问题每次操作可以视为将问题分解为两个子问题，两个子问题任意一个存在解，则该问题存在解。

首先对集合 A 进行排序，便于查找切分点。

对于集合 A，在执行该操作之前，首先检查此数组是否已满足条件，如果满足则返回存在解；否则，若此数组已不可再分则停止递归。

然后计算 $mid = (max(A) + min(A))/2$，使用二分查找找到 mid 所在位置，并据此将数组 A 分为两个子集，分别是 $B = \{a_i | a_i \in A, a_i < mid\}$ 和 $C = \{a_i | a_i \in A, a_i \geq mid\}$，分别递归检查 B 和 C 是否满足条件。伪代码见 **Algorithm** 3。

复杂度分析：首先对数组进行排序，复杂度为 $O(n\log n)$。每次执行该操作，会至少分出一个元素，当集合长度小于等于 1 时停止递归，此时最大值等于最小值，因此最多切分 $n-1$ 次. 执行题目中操作，二分查找 mid 所在位置并切分集合，复杂度为 $O(\log n)$，因此总复杂度为 $O(n\log n)$。

需要注意的错误：每次执行该操作并不能使得数组规模减半，而是使得 $d(A)=max(A)-min(A)$ 减半，该递归树深度最大为 n，但是只有 $n-1$ 条边，而且每次合并复杂度为 $O(1)$，所以总复杂度为 $O(n\log n)$。

最差情况：最差情况为 $A=\{2^i, 1\le i\le n\}$。

Algorithm 2 $Check(A,S,l,r)$

Input:
数组 A，集合元素和 S，下标 l,r

Output:
是否存在满足条件的集合

if $preSum[r]-preSum[l-1]=S$ **then**
  return True
end if
if $l\ge r$ **then**
  return False
end if
$mid \leftarrow \frac{A[l]+A[r]}{2}$
$mid \leftarrow lower\_bound(A+l, A+r+1, mid)-A$
return Check(A,S,l,mid) **or** Check($A,l,mid+1,r$)

Algorithm 3 $RecursiveSum(A,S)$

Input:
集合 A，集合元素和 S

Output:
是否存在满足条件的集合

$A \leftarrow list(A)$
$A \leftarrow sorted(A)$
$preSum[0] \leftarrow 0$
for $i=1$ **upto** n **do**
  $preSum[i] \leftarrow preSum[i-1]+A[i]$
end for
return Check($A,S,1,A.length$)

5 数字消失问题 (20 分)

给定一长度为 n 的数组 $A[1..n]$，其包含 $[0,n]$ 闭区间内除某一特定数 (记做消失的数) 以外的所有数字（例如 $n=3$ 时，$A=[1,3,0]$，则消失的数是 2）。这里假定 $n=2^k-1$。

1. 请设计一个尽可能高效的算法找到消失的数，并分析其时间复杂度。(8 分)
2. 若假定数组 A 用 k 位二进制方式存储（例如 $k=2$，$A=[01,11,00]$ 则消失的数是 10），且不可以直接访存。目前**唯**一可以使用的操作是 `bit-lookup`(i,j)，其作用是用一个单位时间去查询 $A[i]$ 的第 j 个二进制位。请利用此操作设计一个尽可能高效的算法找到消失的数，并分析其时间复杂度。(12 分)

答案：

1. 考虑目前数组对应的值域为 $[L,R]$，则可以利用中位数 $mid=\lfloor\frac{L+R}{2}\rfloor$ 将数组划分成两部分: $\le mid$ 的部分和 $> mid$ 的部分。若 $\le mid$ 的部分的元素个数少于 $mid-L+1$，则说

明消失的数在 $[L, mid]$ 之中，递归考虑，否则我们递归考虑 $> mid$ 的部分。算法伪代码请参考 **Algorithm** 4。

每次仅有至多一半的元素需要进入下一层递归，故递归式可写为 $T(n) = T(\frac{n}{2}) + n$。由主定理可知，时间复杂度为 $T(n) = O(n)$。

2. 类似第一问的思路可以考虑将数组按照二进制位逐位的进行划分，然后迭代的考虑按位划分后每一位个数较少的一部分。算法伪代码请参考 **Algorithm** 5。

   注意到每次循环迭代 S 的过程时，都对 S 集合进行了一次折半操作。故 `bit-lookup` 的操作次数为 $T(n) = \sum_{i=0}^{k-1} \frac{n}{2^k}$。求解该式可得时间复杂度为 $O(n)$。

Algorithm 4 $MissingInteger(A, i, j)$

Input:
数组 A，待寻找的值域 $[i, j]$
Output:
消失的数

```
mid ← ⌊(i+j)/2⌋
if mid 不在 A 数组中 then
    return mid
end if
把 A 数组划分成 B(≤ x) 和 C(> x) 两个部分
if Size(B) < x − i + 1 then
    return MissingInteger(B, i, x)
else
    return MissingInteger(C, x + 1, j)
end if
```

Algorithm 5 $MissingInteger2(A, k)$

Input:

数组 A 及其对应的值域 $[0, 2^k - 1]$

Output:

消失的数

```
S ← 1, 2, ··· , n
S_0 ← S_1 ← ∅
count0 ← count1 ← 0
for posn = k downto 1 do
   for i ∈ S do
      bit ← bit-lookup(i, posn)
      if bit ← 0 then
         count0 ← count0 + 1
         S_0 ← S_0 ∪ {i}
      else
         count1 ← count1 + 1
         S_1 ← S_1 ∪ {i}
      end if
   end for
   if count0 > count1 then
      missing[posn] ← 1
      S ← S_1
   else
      missing[posn] ← 0
      S ← S_0
   end if
   S_0 ← S_1 ← ∅
   count0 ← count1 ← 0
end for
return missing
```