算法基础 2022 春 Homework 3

任课教师：陈雪 due: Mar 24, 15:30

（1）只准讨论思路，严禁抄袭

（2）只能阅读 bb 上的材料和教材算法导论。严禁网上搜寻任何材料，答案或者帮助

问题 1 (20 分)**.** 对下列每个递归式，给出尽量精确的界，并证明其正确性。假定对足够小的 n，$T(n)$ 是常数。

(a) $T(n) = 2T(n/2) + n/\log n$

(b) $T(n) = 4T(n/4) + nlogn$

(c) $T(n) = T(n/2) + 2T(n/4) + n$

(d) $T(n) = \sqrt{n}T(\sqrt{n}) + n$

Answer:

(a) $T(n) = \sum_{k=0}^{\log n-1} 2^k \cdot \frac{n/2^k}{\log(n/2^k)} = n \cdot \sum_{k=0}^{\log n-1} \frac{1}{\log n - k} = n \cdot \sum_{k=0}^{\log n-1} \frac{1}{k} = \Theta(n \cdot \log\log n)$

(b) $T(n) = \sum_{k=0}^{\frac{1}{2}\log n} 4^k \cdot \frac{n}{4^k}\log(\frac{n}{4^k}) = n \cdot \sum_{k=0}^{\frac{1}{2}\log n} (\log n - 2k) = n \cdot [\frac{1}{2}(\log n)^2 - \frac{(\log n)^2 - 2\log n}{2}] = \Theta(n \cdot \log^2 n)$

(c) Guessing $T(n) = \Theta(n \cdot \log n)$. $n\log n \geq \frac{n}{2}\log\frac{n}{2} + \frac{n}{2}\log\frac{n}{4} + n = n\log n - (\frac{3}{2}\log 2 - 1)n$, and $\frac{n}{2}\log n \leq \frac{n}{4}\log\frac{n}{2} + \frac{n}{4}\log\frac{n}{4} + n = \frac{n}{2}\log n - (\frac{3}{4}\log 2 - 1)n$. So $T(n) = \Theta(n \cdot \log n)$.

(d) $T(n) = \sum_{k=0}^{\log\log n-1} n^{1-2^{-k}} \cdot n^{-2^k} = \Theta(n \cdot \log\log n)$

问题 2 (20 分)**.** 在很多的应用如图像压缩中，我们会使用二维 FFT：

给定 $x \in \mathbb{R}^{N\times N}$，其中 N 是 2 的整数次幂。我们的目标是计算 $y \in \mathbb{R}^{N\times N}$，

$$y[k_1, k_2] = \sum_{j_1=0}^{N-1}\sum_{j_2=0}^{N-1} \exp[\frac{2\pi i \cdot (k_1 j_1 + k_2 j_2)}{N}] \cdot x[j_1, j_2]$$

给出一个能在 $O(N^2 \log N)$ 的时间内计算 y 的算法，并证明它的正确性和时间复杂度。

Answer:

$y[k_1,k_2]=\sum_{j_1=0}^{N-1}\{\exp[\frac{2\pi i\cdot k_1j_1}{N}]\cdot\sum_{j_2=0}^{N-1}\exp[\frac{2\pi i\cdot k_2j_2}{N}]\cdot x[j_1,j_2]\}.$

So we just need use 1-dimentional FFT in two dimention respectively.

Firstly, calculating $t[j_1,k_2]=\sum_{j_2=0}^{N-1}\exp[\frac{2\pi i\cdot k_2j_2}{N}]\cdot x[j_1,j_2]$ for all fixed j_1 by FFT.

Then calculating $y[k_1,k_2]=\sum_{j_1=0}^{N-1}\exp[\frac{2\pi i\cdot k_1j_1}{N}]\cdot t[j_1,k_2]$ by FFT.

$T(N)=2N\cdot O(N\log N)=O(N^2\log N)$, Specific algorithm is omitted.

问题 3 (30 分)**.** 设计分治算法；

(a) 给定一个长度为 N，元素各不相同的数组。给出一个计算数组逆序数的算法，且时间复杂度为 $O(N\log N)$，证明它的正确性和时间复杂度。逆序数即数组中满足 $i<j,a[i]>a[j]$ 的二元组 (i,j) 的个数。

(b) 将 (a) 中的逆序数替换为数组中满足 $i<j<k,a[i]<a[j],a[j]>a[k]$ 的三元组 (i,j,k) 的个数，同样给出一个算法，且时间复杂度为 $O(N\log N)$，证明它的正确性和时间复杂度。

Answer:

(a) Using a divide-and-conquer algorithm.

Algorithm 1 inverse number's algorithm

```
function MERGE(A, l, r)
    if l ≥ r then
        return 0
    else
        count = Merge(A, l, (l + r)/2) + Merge(A, (l + r)/2 + 1, r)
        i = l, j = (l + r)/2 + 1, k = l
        while i ≤ (l + r)/2 and j ≤ r do
            if A[i] ≤ A[j] then
                B[k] = A[i], i = i + 1
            else
                B[k] = A[j], j = j + 1
                count = count + ((l + r)/2 − i + 1)
            k = k + 1
        while i ≤ (l + r)/2 do
            B[k] = A[i], i = i + 1, k = k + 1
        while j ≤ r do
            B[k] = A[j], j = j + 1, k = k + 1
        for i = l to r do
            A[i] = B[i]
    return count
```

Assuming **Merge**$(A, l, (l+r)/2)$ and **Merge**$(A, (l+r)/2+1, r)$ have counted all two-tuples in $a[l, ..., (l+r)/2]$ and $a[(l+r)/2, ..., r]$ respectively, the remaining two-tuples in $a[l, ..., r]$ must satisfy $l \leq i \leq (l+r)/2, (l+r)/2+1 \leq j \leq r$. So we just need find out those two-tuples. When we insert $A[j]$, all elements in $A[i, ..., (l+r)/2]$ precede $A[j]$ and is greater than $A[j]$, which need to be counted. And there's no more tuple that meets the conditions. Therefore, we count all tuples in $A[l, ..., r]$. So the algorithm 1 is similar to MergeSort algorithm, and time complexity is $O(N \log N)$.

$T(n) = N \log N$.

(b) $\sum_{1 \leq i<j<k \leq N}^{N} I(a[i] < a[j], a[j] > a[k]) = \sum_{1<j<N} \{[\sum_{1 \leq i<j} I(a[i] < a[j])] \cdot [\sum_{j<k \leq N} I(a[j] > a[k])]\}$.

Algorithm 2 inverse number's algorithm

```
function MERGE(A, count, orgin, l, r, inverse)
    if l ≥ r then
        return 0
    else
        Merge(A, count, l, (l + r)/2, inverse)
        Merge(A, count, (l + r)/2 + 1, r, inverse)
        i = l, j = (l + r)/2 + 1, k = l
        while i ≤ (l + r)/2 and j ≤ r do
            if inverse · A[i] ≤ inverse · A[j] then
                B[k] = A[i], i = i + 1
                tmp[k] = orgin[i]
            else
                B[k] = A[j], j = j + 1
                count[orgin[j]]+ = ((l + r)/2 − i + 1)
                tmp[k] = orgin[j]
            k = k + 1
        while i ≤ (l + r)/2 do
            B[k] = A[i], i = i + 1, k = k + 1
        while j ≤ r do
            B[k] = A[j], j = j + 1, k = k + 1
        for i = l to r do
            A[i] = B[i]
            orgin[i] = tmp[i]
    return 0
function MAIN((A, N))
    for i = 1 to N do
        orgin[i] = i, count[i] = 0
    Merge(A, count, orgin, 1, N, 1)
    for i = 1 to N do
        orgin[i] = i, countInv[i] = 0
    Merge(A, count, orgin, 1, N, −1)
    t = 0
    for i = 1 to N do
        t+ = count[i] ∗ countInv[i]
    return t
```

The new array *orgin* is to find out orginal location of elements. And *count* becomes a array to record how many tuples $A[j]$ is contained exactly. All analysis is similar to (a).

问题 4 (10 分)**.** 如果班上的同学存在两人生日相同的概率超过 1/2，班上至少要有多少名同学？不考虑闰年的情况，给出精确数值解。

Answer:

The probabilities of event that none of the class have the same birthday in the class of k is $\prod_{i=1}^{k}(1-\frac{k-1}{365})$. It's less than 1/2 when $k \geq 23$. Therefore, The probabilities of event that none of the class have the same birthday is greater than 1/2 when 23 students are in the class.

问题 5 (20 分)**.** 我们在课上学习了最大 2 叉堆，可以将它推广到最小 d 叉堆，其中的每个非叶结点有 d 个孩子，而不是仅仅 2 个。

(a) 如何在一个数组中表示一个 d 叉堆?（数组索引从 1 开始，写出推导过程）

(b) 请给出 **EXTRACT-MIN** 在最小 d 叉堆的一个有效实现，并用 d 和 n 表示出它的时间复杂度。

(c) 给出 **DECREASE-KEY**(A, i, k) 在最小 d 叉堆的一个有效实现（其中 i 是要修改的元素现在在堆中的位置，k 是修改后的值），并用 d 和 n 表示出它的时间复杂度。

Answer:

(a) The parent of i-th elements in heap is $\lfloor (i+d-2)/d \rfloor$-th elements, and k-th child is $d \cdot (i-1) + k + 1$,

(b) $T(n) = O(d \log_d n)$

(c) $T(n) = O(\log_d n)$

Algorithm 3 heap's algorithm

```
function EXTRACT-MIN(A)
    if A[0] ≤ 0 then
        raise error // A[0] stores the length of heap
    t = A[1]
    A[1] = A[A[0]]
    A[A[0]] − −
    i = 1
    while d · (i − 1) + 2 ≤ A[0] do
        j = arg min A[d · (i − 1) + 2, ..., min(d · i + 1, A[0])]
        if A[i] ≤ A[j] then
            break
        else
            exchange(A[i], A[j]), i = j
    return t

function DECREASE-KEY(A, i, k)
    if A[i] < k then
        raise error
    else
        while i > 1 do
            p = ⌊(i + d − 2)/d⌋
            if A[p] > A[i] then
                Exchange(A[p], A[i])
                i = p
            else
                break
```