《算法设计与分析》第1次作业

22373058　胡健

October 24, 2024

1　请给出 T(n) 尽可能紧凑的渐进上界并予以说明 (每小题 3 分，共 21 分)

1.1

$$\begin{cases} T(n) = 1\ ,\ n = 1 \\ T(n) = T(n-1) + n\ ,\ if\ n > 1 \end{cases}$$

解:

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= T(n-2) + n + (n-1) \\ &= T(n-3) + n + (n-1) + (n-2) \\ &= \cdots \\ &= n + (n-1) + (n-2) + ... + 2 + 1 \\ &= \frac{1}{2}n(n+1) \\ &= O(n^2) \end{aligned}$$

由此可知，原式渐进上界为$O(n^2)$。

1.2

$$\begin{cases} T(1) = 1, T(2) = 1 \\ T(n) = T(n-2) + 1,\ if\ n > 2 \end{cases}$$

解:

$$\begin{aligned} T(n) &= T(n-2) + 1 \\ &= T(n-4) + 1 + 1 \\ &= \cdots \\ &= 1 + \cdots + 1 + T(2 - n\%2) \\ &= \lceil \frac{n}{2} \rceil \\ &= O(n) \end{aligned}$$

1.3

$$\begin{cases} T(1) = 1 \\ T(n) = 4T(n/2) + n,\ if\ n > 1 \end{cases}$$

解:

由主定理法

$$\text{if } T(n) = aT(\frac{n}{b}) + O(n^d),$$

$$ßT(n) = \begin{cases} O(n^d), & \textbf{if } d > \log_b a \\ O(n^d \log n), & \textbf{if } d = \log_b a \\ O(n^{\log_b a}), & \textbf{if } d < \log_b a \end{cases}$$

这里有$a = 4, b = 2, d = 1$

那么$d < log_b a = 2$

所以$T(n) = O(n^d) = O(n^2)$

1.4

$$\begin{cases} T(1) = 1 \\ T(n) = 8T(n/2) + n^3\ ,\ if\ n > 1 \end{cases}$$

解:

由主定理法

$$\text{if } T(n) = aT(\frac{n}{b}) + O(n^d),$$

$$ßT(n) = \begin{cases} O(n^d), & \textbf{if } d > \log_b a \\ O(n^d \log n), & \textbf{if } d = \log_b a \\ O(n^{\log_b a}), & \textbf{if } d < \log_b a \end{cases}$$

这里有$a = 8, b = 2, d = 3$

那么$d = log_b a = 3$

所以$T(n) = O(n^d) = O(n^3 log n)$

1.5

$$\begin{cases} T(1) = 1, T(2) = 1 \\ T(n) = 2T(n/3) + n^2\ ,\ if\ n > 2 \end{cases}$$

解:

由主定理法

$$\text{if } T(n) = aT(\frac{n}{b}) + O(n^d),$$

$$ßT(n) = \begin{cases} O(n^d), & \textbf{if } d > \log_b a \\ O(n^d \log n), & \textbf{if } d = \log_b a \\ O(n^{\log_b a}), & \textbf{if } d < \log_b a \end{cases}$$

这里有$a=2,b=3,d=2$
那么$d=2>\log_b a$
所以$T(n)=O(n^d)=O(n^2)$

1.6

$$\begin{cases} T(1)=1 \\ T(n)=T(n/2)+\log n\ ,\ if\ n>1 \end{cases}$$

解:

$$\begin{aligned} T(n) &= T(n/2)+\log n \\ &= T(n/4)+\log\frac{n}{2}+\log n \\ &= \cdots \\ &= T(1)+\log 2+\log 4+\cdots+\log\frac{n}{2}+\log n \\ &= 1+\log(\Pi_{i=1}^{n}2^i) \\ &= 1+\log(\sqrt{\Pi_{i=1}^{\log n}2^i*2^{\log n-i}}) \\ &= 1+\log(\sqrt{\Pi_{i=1}^{\log n}2^{\log n}}) \\ &= 1+\frac{1}{2}\log(2^{\log n\times\log n}) \\ &= 1+\frac{1}{2}\log n\times\log n \\ &= O(\log^2 n) \end{aligned}$$

由此可知，原式渐进上界为$O(\log^2 n)$。

1.7

$$\begin{cases} T(1)=1 \\ T(n)=T(n-1)+2^n\ ,\ if\ n>1 \end{cases}$$

解:

$$\begin{aligned} T(n) &= T(n-1)+2^n \\ &= T(n-2)+2^{n-1}+2^n \\ &= \cdots \\ &= T(1)+2^2+2^3+\cdots+2^n \\ &= 2^{n+1}-3 \\ &= O(2^n) \end{aligned}$$

由此可知，原式渐进上界为$O(2^n)$。

2 游戏获奖问题 (19 分)

在一场射击比赛中，有 n 名选手参加比赛，每名选手的得分都不同，记为 $s[1...n]$，其中第 i名选手的得分为$s[i]$，且对于任意的$i \neq j$有 $s[i] \neq s[j]$。按照比赛规则，分数排名为前$\lfloor n/3 \rfloor$的选手可以获得与其得分相等的奖金（包括第 n/3 名）。

例如，当有 6 名选手参加比赛，得分为 s = [100, 50, 90, 60, 65, 80] 时，排名位于前 6/3 = 2的选手得分为 100, 90，则主办方应该发放奖金 190 元。

请你设计一个算法，计算这次比赛主办方一共要发放多少奖金。请描述算法的核心思想，给出算法伪代码并分析其对应的时间复杂度。

2.1 题目分析与核心思想

本题最直观的一个思路是对数组进行排序，然后取前$\lfloor \frac{n}{3} \rfloor$个元素求和即可。

1. 对于排序，使用基于比较的排序，如**归并排序**，那么时间复杂度为$O(n \log n)$
2. 这里可以认为选手得分均为非负整数，那么可以使用**桶排序(计数排序)**，然后取最大的$\lfloor \frac{n}{3} \rfloor$个元素求和，那么时间复杂度为$O(n+k)$，其中k为选手得分的最大值

然后考虑的一个点是，排序是把数组的每个元素的序都确定了，如果只是取出最大的$\lfloor \frac{n}{3} \rfloor$个元素，实际上不需要全部排序。那么就有使用类似快速排序的**快速选择**的算法，最好时间复杂度和期望时间复杂度均为$O(n)$，最坏时间复杂度为$O(n^2)$。

2.2 算法与伪代码

2.2.1 基于排序算法

Algorithm 1: 基于归并排序处理游戏获奖问题

Input: 正整数n，为数组长度，每名选手的得分数组：$A[1...n]$

Output: 一个整数Ans，表示主办方发放的奖金

```
function main(n, A):
    MergeSort(A, 1, n)
    Ans ← 0
    for i ← 1 to ⌊n/3⌋ do
        Ans+ = A[i]
    end
    return Ans
end
function MergeSort(A, left, right):
    if left ≥ right then
        return A[left..right]
    end
    mid ← ⌊(left+right)/2⌋
    MergeSort(A, left, mid)
    MergeSort(A, mid + 1, right)
    Merge(A, left, mid, right)
    return A[left..right]
end
function Merge(A, left, mid, right):
    A'[left, right] ← A[left, right]
    i ← left, j ← mid + 1, k ← 0
    while i ≤ mid and j ≤ right do
        if A'[i] ≤ A'[j] then
            A[left + k] ← A'[i]
            k ← k + 1, i ← i + 1
        end
        else
            A[left + k] ← A'[j]
            k ← k + 1, j ← j + 1
        end
    end
    if i ≤ mid then
        A[left + k..right] ← A'[i..mid]
    end
    else
        A[left + k..right] ← A'[j..right]
    end
    return A[left..right]
end
```

Algorithm 2: 基于计数排序处理游戏获奖问题

Input: 正整数n，为数组长度，每名选手的得分数组：$A[1...n]$

Output: 一个整数Ans，表示主办方发放的奖金

```
function main(n, A):
    max ← 0 for i ← 1 to n do
        if max < A[i] then
            max = A[i]
        end
    end
    CountingSort(A, n, max)
    Ans ← 0
    for i ← 1 to ⌊n/3⌋ do
        Ans+ = A[i]
    end
    return Ans
end
function CountingSort(A, n, max):
    T[0..max] ← [0] * (max + 1)
    for i ← 1 to n do
        T[A[i]] ← T[A[i]] + 1
    end
    j ← 0
    for i ← 0 to max do
        while T[i] > 0 do
            A[j] ← i
            T[i] ← T[i] − 1
            j ← j + 1
        end
        if j = n − 1 then
            Break
        end
    end
end
```

2.2.2 基于快速选择算法

Algorithm 3: 基于快速选择算法处理游戏获奖问题

Input: 正整数n，为数组长度，每名选手的得分数组：$A[1...n]$

Output: 一个整数Ans，表示主办方发放的奖金

```
1  function main(n, A):
2      price ← RandomizedSelection(A, 1, n, ⌊n/3⌋)
3      Ans ← 0
4      for i ← 1 to n do
5          if A[i] ≥ price then
6              Ans ← Ans + A[i]
7          end
8      end
9      return Ans
10 end
11 function RandomizedSelection(A, p, r, k):
12     q ← RandomizedPartition(A, p, r)
13     if k = (q − p + 1) then
14         x ← A[q]
15     end
16     if k < (q − p + 1) then
17         x ← RandomizedSelection(A, p, q − 1, k)
18     end
19     if k > (q − p + 1) then
20         x ← RandomizedSelection(A, q + 1, r, k − (q − p + 1))
21     end
22     return x
23 end
24 function RandomizedPartition(A, p, r):
25     s = Random(p, r)                                  ▷ 随机选择主元
26     exchange A[s] with A[r]
27     x ← A[r]
28     i ← p − 1
29     for j ← p to r − 1 do
30         if A[j] ≥ x then
31             exchange A[i + 1] with A[j]
32             i ← i + 1
33         end
34     end
35     exchange A[i + 1] with A[r]
36     q ← i + 1
37     return q
38 end
```

2.3 时间复杂度分析

基于**归并排序**处理游戏获奖问题时，算法主要分为排序阶段与求和阶段，排序阶段的时间复杂度递推式为 $T(n) = T(\frac{n}{2}) + O(n)$，由主定理可求得时间复杂度为$O(n\log n)$，求和阶段的时间复杂度为$O(n)$，因此算法的总时间复杂度为$O(n\log n)$

基于**计数排序**处理游戏获奖问题时，算法主要分为排序阶段与求和阶段，排序阶段为主要包括遍历原数组和遍历长度为数组元素最大值的数组两个部分，时间复杂度为$O(n+k)$,其中k为数组元素的最大值，求和阶段的时间复杂度为$O(n)$，因此算法的总时间复杂度为$O(n+k)$

基于**快速选择**算法处理游戏获奖问题时，算法的主要分为固定位置次序选择部分和遍历数组取大于第$\lfloor\frac{n}{3}\rfloor$的元素的所有元素，固定位置次序选择时调用了随机化的固定位置划分，最优解为第一次即找出了第$\lfloor\frac{n}{3}\rfloor$的元素，时间复杂度为$O(n)$，最坏情况为$T(n) = O(n) + O(n-1) + \cdots O(1) = \sum_{i=1}^{n} i = O(n^2)$，期望复杂度为$O(n)$。之后遍历的时间复杂度为$O(n)$，因此最终的期望时间复杂度为$O(n)$

下面证明基于快速选择期望复杂度为$O(n)$

随机选择主元，共n种情况(假设元素互不相同):

$$T(n) \leq \begin{cases} maxT(0), T(n-1) + O(n) = T(n-1) + O(n) \\ maxT(1), T(n-2) + O(n) = T(n-2) + O(n) \\ maxT(2), T(n-3) + O(n) = T(n-3) + O(n) \\ \cdots \\ maxT(n-1), T(0) + O(n) = T(n-1) + O(n) \end{cases}$$

每种情况出现的概率均为$\frac{1}{n}$，每个$T(i)$2!fi$i \geq \lfloor\frac{n}{2}\rfloor$

期望时间：

$$\begin{aligned} E[T(n)] &\leq E[\frac{2}{n} \cdot \sum_{i=\lfloor\frac{n}{2}\rfloor}^{n-1} (T(i) + O(n))] \\ &\leq \frac{2}{n} \cdot E[\sum_{i=\lfloor\frac{n}{2}\rfloor}^{n-1} (T(i) + O(n))] \\ &= \frac{2}{n} \cdot \sum_{i=\lfloor\frac{n}{2}\rfloor}^{n-1} E[(T(i))] + \frac{2}{n} \cdot \sum_{i=\lfloor\frac{n}{2}\rfloor}^{n-1} O(n) \\ &= \frac{2}{n} \cdot \sum_{i=\lfloor\frac{n}{2}\rfloor}^{n-1} E[(T(i))] + O(n) \end{aligned}$$

通过代入法分析期望可得，假设：$\forall i < n, E[T(i)] < c \cdot i$

那么代入有$E[T(n)] \leq O(n) + \frac{2}{n}\sum_{i=\lfloor\frac{n}{2}\rfloor}^{n-1} c \cdot i \leq O(n) + \frac{2}{n} \cdot c \cdot \frac{3}{8}n^2 \leq c \cdot n$

综上即证

3 奇数因子和问题 (20 分)

定义 $od(i)$ 为整数 i 的最大奇数因子，例如 $od(3) = 3,\ od(14) = 7$。

请你设计一个高效算法，计算一个整数区间$[A, B]$内所有数的最大奇数因子和，即$\sum_{i\in[A,B]} od(i)$。请描述算法的核心思想，给出算法伪代码并分析其对应的时间复杂度。

例如，区间 [3, 9] 的计算结果为 $3 + 1 + 5 + 3 + 7 + 1 + 9 = 29$。

3.1 题目分析与核心思想

本题最直接的想法是直接暴力求解，对于每一个函数$od(n), n \in \mathbb{Z}$，设$t = abs(n) = |n|$有

$$od(n) = od(t) = \begin{cases} t, & t \mod 2 = 1 \\ \frac{t}{2}, & t \mod 2 = 0 \end{cases}$$

那么计算每个数n的$od(n)$的时间复杂度为$O(\log n)$，那么计算整数区间$[A, B]$的所有数的最大奇数因子和的时间复杂度为$(B - A) \log B$

通过观察函数的性质，我们发现可以将区间作为一个整体来进行考虑，于是有：

设所求整数区间$[A, B]$的所有数的最大奇数因子和为$S(A, B)$，这里我们发现有

$$S(A,B) = \begin{cases} S(0,B) - S(0,A-1), & 0 < A \le B \\ S(0,B) + S(A,0) = S(0,B) + S(0,-A), & A \le 0 \le B \\ S(A,0) - S(B+1,0) = S(0,-A) - S(0,-B-1), & A \le B < 0 \end{cases}$$

因此我们可以定义一个新的函数$f(n), n \in \mathbb{Z}^*$表示整数区间$[0, n]$的所有数的最大奇数因子和。

那么就有

$$S(A,B) = \begin{cases} f(B) - f(A-1), & 0 < A \le B \\ f(B) + f(-A), & A \le 0 \le B \\ f(-A) - f(-B-1), & A \le B < 0 \end{cases}$$

因此所求的$S[A, B]$，只需计算函数f的相应的值即可。

易知$f(0) = 0, f(1) = 1$，对于$f(n), n > 1 \ \wedge n \in \mathbb{Z}$,考虑如下递推式

$$\begin{aligned} f(n) &= \sum_{i=0}^{n} od(i) \\ &= (od(1) + od(3) + \cdots + od(2\lceil \frac{n}{2} \rceil - 1)) + (od(2) + od(4) + \cdots + od(2\lfloor \frac{n}{2} \rfloor)) \\ &= \sum_{i=1}^{\lceil \frac{n}{2} \rceil} od(2i-1) + \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} od(2i) \\ &= \sum_{i=1}^{\lceil \frac{n}{2} \rceil} (2i-1) + \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} od(2i) \\ &= \lceil \frac{n}{2} \rceil^2 + \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} od(i) \\ &= \lceil \frac{n}{2} \rceil^2 + f(\lfloor \frac{n}{2} \rfloor) \\ &= \lfloor \frac{n+1}{2} \rfloor^2 + f(\lfloor \frac{n}{2} \rfloor) \end{aligned}$$

因此可以通过递推式直接求出$f(n), n \in \mathbb{Z}^*$的值，从而求得对应情况的$S[A, B]$

3.2 算法与伪代码

Algorithm 4: 奇数因子和问题

Input: 两个整数$A, B,\ A \leq B$

Output: 一个整数Ans，表示整数区间$[A, B]$的所有数的最大奇数因子和

```
function main(A, B):
    if 0< A ≤ B then
        return f(B) − f(A − 1)
    end
    else if A≤ 0 ≤ B then
        return f(B) + f(−A)
    end
    else
        return f(−A) − f(−B − 1)
    end
end
function f(n):
                                        ▷ 递归得到整数区间[0, n]的所有数的最大奇数因子和
    if n = 0 then
        return 0
    end
    else if n = 1 then
        return 1
    end
    else
        return ⌊(n+1)/2⌋^2 + f(⌊n/2⌋)
    end
end
function f′(n):
                                        ▷ 循环得到整数区间[0, n]的所有数的最大奇数因子和
    Ans ← 0
    while n > 0 do
        Ans ← Ans + ⌊(n+1)/2⌋
        n ← ⌊n/2⌋
    end
    return Ans
end
```

3.3 时间复杂度分析

该算法的核心复杂度为求$f(A)$和$f(B)$，考虑$f(n)$的递推式$f(n) = \lfloor \frac{n+1}{2} \rfloor^2 + f(\lfloor \frac{n}{2} \rfloor)$可知：

$f(n)$的时间复杂度有：

$$T(n) = T(\frac{n}{2}) + O(1)$$

由主定理法

$$\text{if } T(n) = aT(\frac{n}{b}) + O(n^d),$$

$$ß T(n) = \begin{cases} O(n^d), & \textbf{if } d > \log_b a \\ O(n^d \log n), & \textbf{if } d = \log_b a \\ O(n^{\log_b a}), & \textbf{if } d < \log_b a \end{cases}$$

有$a = 1, b = 2, d = 0$，那么$d = \log_b a = 0$，所以$T(n) = O(\log n)$

那么算法的时间复杂度即为$O(\log A + \log B)$，记$n = max(|A|, |B|)$，则算法的时间复杂度为$O(\log n)$

4 施工点对计数问题 (20 分)

在一个城市的修建工程中，有 n 个施工点，其中 $costs[i]$ 表示第 i 个施工点的成本。

为完成任务，工程师们需要选择两个不同的施工点进行联合施工，联合施工的成本为每个施工点的成本加和，并且需要保证联合施工的总成本不能超过预算上限。形式化地说：需要计算所有满足预算限制l的施工点对(i, j) 的数量，其中$1 \le i < j \le n$ 且 $costs[i] + costs[j] \le l$。

请你设计并实现一个算法来解决该问题，描述算法的核心思想，给出算法伪代码并分析其对应的时间复杂度。

4.1 题目分析与核心思想

这里只需要考虑联合施工点的数量，那么对$costs$数组进行修改不会影响数量的求解，同时为了方便求解，可以对$costs$数组进行排序，使用双指针方法，从头和尾对数组进行遍历即可。

1. 使用题目2中的归并排序对数组进行升序化处理，这里认为$costs$数组的值不一定为整数，因此不能够使用题目2的计数排序来进行处理
2. 使用双指针对数组进行遍历(此时数组元素已经按照从小到大排列)，初始时令$i = 1, j = n$(分别是头指针和尾指针)，

$$\begin{cases} costs[i] + costs[j] \le l & \Rightarrow \forall\ k \in [i+1..j], (i, k)\ is\ OK \\ costs[i] + costs[j] > l & \Rightarrow \forall\ k \in [i+1..j], (k, j)\ is\ not\ OK \end{cases}$$

然后将头指针往后移动或者将尾指针往前移动即可。持续进行上述操作，直到$i = j$即可。

4.2 算法与伪代码

Algorithm 5: 施工点对计数问题

Input: 两个正整数n, l，分别表示施工点数目和预算限制，一个$costs[1..n]$数组，表示每个施工点的成本

Output: 一个正整数Ans，表示满足题目要求的施工点对的数目

```
1  function main(n, l, costs):
2      MergeSort(costs, 1, n)                          ▷ 使用题目2中的MergeSort
3      i ← 1, j ← n
4      Ans ← 0 while i < j do
5          if costs[i] + costs[j] ≤ l then
6              Ans ← Ans + j − i
7              i ← i + 1
8          end
9          else
10             j ← j − 1
11         end
12     end
13     return Ans
14 end
```

4.3 时间复杂度分析

1. 归并排序部分的递推公式为$T(n) = 2T(\frac{n}{2}) + O(n)$，可知时间复杂度为$O(n \log n)$
2. 双指针计算部分，每次对于都是对于数组中未访问元素进行访问，直到将数组元素全部访问一遍为止，因此时间复杂度为$O(n)$

因此本问题最终的时间复杂度为$O(n \log n) + O(n) = O(n \log n)$

5 最近村庄距离问题 (20 分)

在一个广阔的平原上，有 n个村庄。每个村庄 i都有一个独特的位置，记为 (x_i, y_i)，表示在平面上横纵坐标的位置。现在，政府打算优先在两个距离最接近的村庄之间修建一条公路，以便加强它们之间的交通联系，其中村庄之间的距离定义为欧几里得距离，即对于两个村庄 $A(x_1, y_1)$和 $B(x_2, y_2)$，它们之间的距离为：$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

请你设计一个高效算法，计算在平原上距离最近的两个村庄之间的距离。请描述算法的核心思想，给出算法伪代码并分析其对应的时间复杂度。

5.1 题目分析与核心思想

本题朴素的处理思路是遍历所有距离，那么时间复杂度是$O(n^2)$，但是这里计算了很多个距离，实际上有很多是不需要进行计算的，那么就可以有更加优的处理策略。

基本想法是使用分治策略，按照横坐标进行二分，得到左半区域与右半取悦的最小距离，然后在合并的时候进行处理，合并时如果计算两侧所有点的距离，那么跟朴素的暴力方法没有区别。一个想法是只需要计算部分点之间的距离即可。

具体而言：

本题实际上是一个最近点对问题，平面上有n个点，计算距离最近的两个点的距离。

设每个点的位置表示为$\text{Loc}(i) = (x_i, y_i), 1 \le i \le n$，有位置数组$\text{Loc}[1..n]$

1. 将Loc[1..n]按照横坐标的值进行归并排序

2. 采用分治算法计算Loc[$left..right$]部分的最近的两个点对的距离

   - 分：$mid = \frac{left+right}{2}$，分别计算Loc[$left..mid$]和Loc[$mid+1..right$]两个部分的最近的两个点对的距离，分别设为$d_l, d_r$
   - 治：对于每侧，若点的个数小于4，直接暴力遍历求出最近点对的距离；否则递归计算最近点对的距离
   - 合：在得到左右两个区间的点对距离最小值后，计算从这两个子数组中各自选出一个点计算最近点对距离d'，然后比较d'与$min(d_l, d_r)$的大小，更新Loc[$left..right$]的最近的两个点对的距离。

此时问题主要需要解决的是计算**合**的过程的最近点对距离。

设$d = min(d_l, d_r)$，那么只需要考虑$\text{Loc}_m = (x_m, y_m)(m = \frac{left+right}{2})$左右横坐标距离之差为$d$的所有点，也就是构造集合$B$：

$$B = \{\text{Loc}_i | x_m - d < x_i < x_m + d\}$$

对于B中的每个点Loc_i，当前目标是找到一个同样在B中、且到其距离小于h的点。为了避免两个点之间互相考虑，只考虑那些纵坐标小于y_i的点。显然对于一个合法的点Loc_j，$y_i - y_j$必须小于h。由此可定义集合$C(\text{Loc}_i)$：

$$C(\text{Loc}_i) = \{\text{Loc}_j \mid \text{Loc}_j \in B,\ y_i - h < y_j \le y_i\}$$

由此可得合并的步骤如下：

1. 构建集合B。

2. 将B中的点按照y_i排序。

3. 对于每个$\text{Loc}_i \in B$考虑$\text{Loc}_j \in C(\text{Loc}_i)$，对于每对$(\text{Loc}_i, \text{Loc}_j)$计算距离并更新当前所处集合的最近点对。

上文中的两次排序，由于点坐标全程不变，第一次排序可以只在分治开始前进行一次，每次递归返回当前点集按y_i排序的结果；对于第二次排序，上层直接使用下层的两个分别排序过的点集归并即可。

Figure 1: 构造集合$C(\text{Loc}_i)$

对于集合$C(\text{Loc}_i)$的构造，并不需要遍历B中所有的元素，考虑这么一个x方向长度为$2d$，y方向长度为d的长方形，将其对称分为如上图:

5.2 算法与伪代码

Algorithm 6: 最近村庄距离问题

Input: 一个正整数n，表示村庄数目，一个所有村庄位置数组Loc

Output: 一个数Ans，表示这n个村庄中最近的两个村庄之间的距离

```
function main(n, Loc):
    MergeSort(Loc.x, 1, n)                    ▷ 使用题目二中的归并排序对Loc按照x坐标进行排序
    return minDist(Loc, 1, n)
end
function minDist(Loc, left, right):
    if right − left ≤ 3 then
        D ← ∞
        for i ← left to right − 1 do
            for j ← i + 1 to right do
                D ← min(√((Loc_i.x − Loc_j.x)² + (Loc_i.y − Loc_j.y)²), D)
            end
        end
        MergeSort(Loc.y, left, right)         ▷ 使用题目二中的归并排序对Loc按照y坐标进行排序
        return D
    end
    mid ← (left+right)/2
    D_1 ← minDist(Loc, left, mid)
    D_2 ← minDist(Loc, mid + 1, right)
    D = mid(D_1, D_2)
    Merge(Loc.y, left, mid, right)            ▷ 使用题目二中的归并对Loc进行按照y坐标排序后的合并
    B ← CreateArray()                                              ▷ 新建一个数组B
    for i ← left to right do
        if |Loc_i.x − Loc_mid.x| < D then
            B.add(Loc_i)
        end
    end
    for i ← 2 to B.length do
        for j ← i − 1 to max(0, i − 7) do
            D ← min(√((Loc_i.x − Loc_j.x)² + (Loc_i.y − Loc_j.y)²), D)
        end
    end
    return D
end
```

5.3 时间复杂度分析

该算法主要分为几个部分

1. 最开始对数组进行按照x坐标进行归并排序，时间复杂度为$O(n \log n)$
2. 使用分治法进行最短点对距离的求解

第二次分治时，考虑有$T(n) = 2T(\frac{n}{2}) + f(n)$，其中$f(n)$表示合的过程的中的时间复杂度。

计算合并操作中更新跨数组最近点对的时间复杂度：

由于$C(\text{Loc}_i)$中的所有点的纵坐标都在$(y_i - h, y_i]$范围内；且$C(\text{Loc}_i)$中的所有点和Loc_i本身的横坐标都在$(x_m - d, x_m + d)$范围内。这构成了一个$2d \times d$的矩形。

根据$Figure1$所示,我们再将这个矩形拆分为两个$d \times d$的正方形，不考虑Loc_i，其中一个正方形中的点为$C(\text{Loc}_i) \cap A_1$，另一个为$C(\text{Loc}_i) \cap A_2$，由于它们来自同一下层递归，因此两个正方形内的任意两点间距离大于等于d。

接下来即可证明每个$d \times d$的正方形内最多有4个点：

将一个$d \times d$的正方形拆分为四个$\frac{d}{2} \times \frac{d}{2}$的小正方形。则每个小正方形中最多有1个点：因为该小正方形中任意两点最大距离是对角线的长度，即$\frac{d}{\sqrt{2}}$，该数小于d。

根据**鸽巢原理**，若有大于等于5个点，则必有两个点在同一个小正方形中，那么这两个小正方形之间的距离必然小于d，与两个正方形内的任意两点间距离大于等于d矛盾。由此，每个正方形中最多有4个点，矩形中最多有8个点，去掉Loc_i本身，$\max(C(\text{Loc}_i)) = 7$。

综上可知，合并操作中，更新跨数组最近点对的时间复杂度为7即$O(1)$。

1. Merge操作的时间复杂度为$O(n)$
2. 遍历构造数组B的时间复杂度为$O(n)$
3. 最后二重循环遍历B中元素时，由于最多只有7个点，因此时间复杂度为$O(B.length) * O(7) = O(n)$

因此整个合并操作的时间复杂度为$O(n)$。又根据分治算法的时间复杂度为：

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

由主定理法知算法时间复杂度为$O(nlogn)$