目录

[第1章. 算法在计算中的作用 3](#_Toc520409494)

[1.1 算法 3](#_Toc520409495)

[1.2 作为一种技术的算法 3](#_Toc520409496)

[思考题 3](#_Toc520409497)

[第2章. 算法基础 3](#_Toc520409498)

[2.1 插入排序 3](#_Toc520409499)

[2.2 分析算法 4](#_Toc520409500)

[2.3 设计算法 4](#_Toc520409501)

[思考题 6](#_Toc520409502)

[第3章. 函数的增长 7](#_Toc520409503)

[3.1 渐近记号 7](#_Toc520409504)

[3.2 标准记号与常用函数 8](#_Toc520409505)

[思考题 9](#_Toc520409506)

[第4章. 分治策略 11](#_Toc520409507)

[4.1 最大子数组 11](#_Toc520409508)

[4.2 矩阵乘法的Strassen 12](#_Toc520409509)

[4.3 用带入法求解递归式 12](#_Toc520409510)

[4.4 用递归树方法求解递归式 12](#_Toc520409511)

[4.5 用主方法求解递归式 12](#_Toc520409512)

[4.6 证明主定理 12](#_Toc520409513)

[思考题 12](#_Toc520409514)

[第5章. 概率分析和随机算法 13](#_Toc520409515)

[5.1 雇用问题 13](#_Toc520409516)

[5.2 指示器随机变量 14](#_Toc520409517)

[5.3 随机算法 15](#_Toc520409518)

[5.4 概率分析和指示器随机变量的进一步使用 16](#_Toc520409519)

[思考题 16](#_Toc520409520)

[第6章. 堆排序 17](#_Toc520409521)

[6.1 堆 17](#_Toc520409522)

[6.2 维护堆的性质 18](#_Toc520409523)

[6.3 建堆 18](#_Toc520409524)

[6.4 堆排序算法 18](#_Toc520409525)

[6.5 优先队列 18](#_Toc520409526)

[思考题 19](#_Toc520409527)

[第7章. 快速排序 21](#_Toc520409528)

[7.1 快速排序的描述 21](#_Toc520409529)

[7.2 快速排序的性能 21](#_Toc520409530)

[7.3 快速排序的随机化版本 22](#_Toc520409531)

[7.4 快速排序分析 22](#_Toc520409532)

[思考题 24](#_Toc520409533)

[第8章. 线性时间排序 28](#_Toc520409534)

[8.1 排序算法的下界 28](#_Toc520409535)

[8.2 计数排序 29](#_Toc520409536)

[8.3 基数排序 30](#_Toc520409537)

[8.4 桶排序 31](#_Toc520409538)

[思考题 32](#_Toc520409539)

1. 算法在计算中的作用

## 算法

## 作为一种技术的算法

Answer

Answer

## 思考题

#### 1-1运行时间的比较

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 sec | 1 min | 1 hour | 1 day | 1 month | 1 year | 1 century |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

1. 算法基础

## 插入排序

### 插入排序实例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 | 41 | 59 | 26 | 41 | 58 |
| 31 | 41 | 59 | 26 | 41 | 58 |
| 31 | 41 | 59 | 26 | 41 | 58 |
| 26 | 31 | 41 | 59 | 41 | 58 |
| 26 | 31 | 41 | 41 | 59 | 58 |
| 26 | 31 | 41 | 41 | 58 | 59 |

### 降序排列

Answer 将算法line5的改为即可实现降序排列。

### 线性查找

Linear-Lookup (A, v)

1. for i =1 to A.length
2. if A[i] == v
3. return i
4. return NIL

## 分析算法

Answer

### 选择排序

SELECT-SORT (A)

1. for i =1 to A.length-1
2. min = i
3. for j=i+1 to A.length
4. if A[j] < A[min] min = j
5. exchange A[i] with A[min]

循环不变式：每次line1处迭代开始时，是A中最小的i-1个元素的按序排列。

当时，是A中最小的n-1个元素的按序排列，此时一定是A中最大值。

运行时间在所有情况下都是。

### 线性查找次数

Answer 平均检查次数，最坏检查次数。

Answer 测试输入是否是一些特殊情况。如果是，输出预先计算好的结果。

## 设计算法

### 归并排序实例

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 41 | 52 | 26 | 38 | 57 | 9 | 49 |
| 3 | 41 | 26 | 52 | 38 | 57 | 9 | 49 |
| 3 | 26 | 41 | 52 | 9 | 38 | 49 | 57 |
| 3 | 9 | 26 | 38 | 41 | 49 | 52 | 57 |

### 不使用哨兵的MERGE算法

MERGE(A, p, q, r)

1. n1 = q-p+r, n2 = r-q
2. let L[1..n1] and R[1..n2] be new arrays
3. for i=1 to n1
4. L[i] = A[p+i-1]
5. for j=1 to n2
6. R[j] = A[q+j]
7. i =1 , j = 1, k = p
8. while i <= n1 and j <= n2
9. if L[i] <= R[j]
10. A[k] = L[i]
11. i = i+ 1
12. else A[k] = R[j]
13. j= j + 1
14. k = k+ 1
15. if i == n1
16. while j <= n2
17. A[k] = R[j]
18. j = j + 1, k = k + 1
19. else
20. while i <= n1
21. A[k] = L[i]
22. i = i + 1, k = k + 1

Proof 时，。假设当时有，则当时，，归纳完成。故当是2的幂时。

Answer 插入已排序的数组在最坏情况下耗时，故递归式为

### 二分查找

Iterative-Binary-Search(A, low, high, key)

1. while low<=high
2. mid =
3. if A[mid] == key
4. return mid
5. else if A[mid] < key
6. high = mid - 1
7. else low = mid + 1
8. return NIL

运行时间在最坏情况下满足，故。

**Answer** 二分查找可以在时间找到的插入位置，但是依然需要进行元素移动，该步骤耗时，故运行时间不受影响。

### 2-SUM问题

2-SUM(S, x)

1. sort S
2. i =1, j = S.length
3. while i < j
4. if S[i] + S[j] == x
5. return True
6. else if S[i] + S[j] < x
7. i = i +1
8. else j = j – 1
9. return False

## 思考题

#### 2-1在归并排序中对小数组采用插入排序

1. 插入排序在长度为的子数组上最坏运行时间。个子数组最坏情况耗时。
2. 对m=n/k个子数组两两归并，得到个有序子数组进入下一轮两两归并直到只有一个数组。每一轮归并耗时，一共轮，故归并总耗时。
3. 为使，显然的阶不能超过。若，则。故的阶最大是。
4. 应当是使插入排序比归并排序快的最大输入规模。

#### 2-2冒泡排序

1. 还需证明是的一个排列
2. **循环不变式**：line2 for循环每次迭代开始时，是子数组的最小值且是line2开始前的子数组的一个排列。

**初始化**：j=n且仅有一个元素，循环不变式成立。

**保持**：考虑给定j的一次迭代，迭代开始前是子数组的最小值。接下来如果发现，则交换二者使得是子数组的最小值。若，那么自然是子数组的最小值。此过程唯一的变化就是与可能的交换并且是line2开始前的子数组的一个排列，故下一次迭代开始前是子数组的最小值且是line2开始前的子数组的一个排列。

**终止**：终止时。根据循环不变式，是子数组的最小值且是line2开始前的子数组的一个排列。

1. **循环不变式**：line1 for循环每次迭代开始时，是原始数组最小的i-1个元素的按序排列且是原始数组剩下的n-i+1个元素的一个排列。

**初始化**：，循环不变式显然成立。

**保持**：考虑给定的一次迭代。迭代开始前是原始数组最小的i-1个元素的按序排列。根据题b部分的结论，迭代后是子数组的最小值，故是原始数组最小的个元素的按序排列。同时迭代后是迭代开始前的子数组的一个排列，而迭代开始前的子数组是原始数组剩下的n-i+1个元素的一个排列。故迭代后的是原始数组剩下的个元素的一个排列。

**终止**：终止时。根据循环不变式，是原始数组最小的n-1个元素的按序排列且是原始数组剩下的1个元素的一个排列。故必是原始数组最大的元素。

1. line3~line4执行次数为。故冒泡排序运行时间恒为。

#### 2-3 Horner规则

1. 该算法计算需次乘法，累计需次乘法，然后所有的累和需次加法。故运行时间为。
2. **初始化**：，，循环不变式显然成立

**保持**：考虑给定的一次迭代。若迭代前，则在line3中，故循环不变式保持成立。

**终止**：，按照循环不变式，。

1. 根据题c部分，Horner规则正确性显然。

#### 2-4逆序对

1. (1, 5)，(2, 5)，(3, 4)，(3, 5)，(4, 5)
2. 按降序排列时具有最多的逆序对，逆序对数为。
3. 线性关系，插入排序代码中line6-line7的执行次数恰为逆序对数。这是因为，line6~line7是把所有排在A[j]之前且比A[j]大的元素右移一格，移动次数就是排在A[j]之前且比A[j]大的元素个数。当j从2遍历到A.length，总的移动次数就是总的逆序对数。
4. 算法思想：设算法COUNT-INVERSION(A, p, r)返回数组的逆序对数。的逆序对数是的逆序对数加上的逆序对数再加上中某个较大元素和中某个较小元素构成的逆序对数。前二者可由COUNT-INVERSION递归计算，最后一个可通过修改练习2.3-2的MERGE(A, p, q, r)实现。在MERGE(A, p, q, r)代码中，在line7与line8之间初始化逆序对计数。一旦line12执行，就意味着，此时的每个元素都大于，从而由贡献的逆序对数为，故在该条件分支中添加。修改后的归并算法记为MERGE-INVERSION (A, p, q, r)。

COUNT-INVERSION (A, p, r)

1. invert = 0
2. if p < r
3. q =
4. invert = invert + COUNT-INVERSION (A, p, q)
5. invert = invert + COUNT-INVERSION (A, p+1, r)
6. invert = invert + MERGE-INVERSION (A, p, q, r)
7. return invert
8. 函数的增长

## 渐近记号

Proof 当n充分大时，且，故且。故。

Proof 当时有，故成立

从而。

Answer 记运行时间为。表示，是集合中的某个函数。注意到函数也在集合中，故等价于。由于总是非负，这是平凡的结论。

Answer ，。

### 证明定理3.1

Proof 必要性显然。下面考虑充分性。表示，存在正常数和，只要，就有。表示，存在正常数和，只要，就有。现在令，则当时，，故。

Proof 必要性显然。下面考虑充分性。假设算法的运行时间，则。设某个，则存在正常数和，只要，就有。设某个，则存在正常数和，只要，就有。现在令，则当时，成立。故。

**Proof** 任取，下面证明。若，则对任意正常量，存在正整数，对任意，都有。于是对于给定的正常量，对于任意正整数（无论它多大），只需取，就存在，使得，即。故。

**Answer**

## 标准记号与常用函数

**Proof** 若则，。于是，，即和也是单调递增的。若，，则，即也是单调递增的。

**Proof**

**Proof** 根据Stirling公式，一方面，当n充分大时

另一方面，对任意满足，当时

故。

当n充分大时，对任意正常数，且。故且。

**Answer** 不是多项式有界，是多项式有界的。注意到多项式有界当且仅当。

Answer 。若（个2），，则，。

Proof 显然。

Proof 时结论成立。归纳假设：对，当时都有。则当时，

即当时结论也成立。

Proof

## 思考题

#### 3-1 多项式的渐近行为

以下记

1. 。当时

故

1. 。当时

故。

1. 由题a和题b立马得到。
2. 。对任意正常数，当时

故。

1. 。根据题b，当且时

故。

#### 3-2 相对渐近增长

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

#### 3-3 根据渐近增长率排序

1. 按照阶由高到低排列为，，，，，，，，，(，，，和，，，，，，，，，，，和1
2. 构造一个在0到之间取值且函数值随n增大始终波动的函数。

#### 3-4 渐近记号的性质

1. 错误。
2. 错误。考虑，。
3. 正确。
4. 正确。
5. 错误。考虑。
6. 正确。
7. 错误。考虑。
8. 正确。

#### 3-5 与的一些变形

#### 3-6 多重函数

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | 0 |  |
|  | 1 |  |
|  | 1 |  |
|  | 2 |  |
|  | 2 |  |
|  | 1 |  |
|  | 2 |  |
|  | 2 |  |

1. 分治策略

## 最大子数组

Answer 所有元素都是负值时，返回数组最大值。

Answer 对所有的满足的序偶，求子数组的和。

Max-Subarray(A)

1. max\_sum = -
2. for i = 1 to A.length-1
3. cur\_sum = 0
4. for j = i to A.length
5. cur\_sum = cur\_sum + A[j]
6. if cur\_sum > max\_sum
7. max\_sum = cur\_sum
8. return max\_sum

Answer 假设修改前二者性能在问题规模时恰好相同。修改后当问题规模小于时，如果忽略因修改引入的条件判断和函数调用等常量时间开销，递归算法与暴力求解性能相同，考虑该开销时递归算法略差一点。大于时递归算法优于暴力求解。

Answer 修改FIND-CROSSING-SUBARRAY line15 和FIND-MAXIMUM-SUBARRAY line 2的return语句。对返回值作条件判断，若返回的和小于0，则返回(-1,-1,0)，其中(-1,-1)为空数组的下标。

### 线性时间求解最大子数组

Find-Max-Subarray(A)

1. cur\_sum = 0, max\_sum = -
2. cur\_beg = 1
3. for i = 1 to A.length
4. cur\_sum = cur\_sum + A[i]
5. if cur\_sum > max\_sum
6. max\_sum = cur\_sum
7. beg = cur\_beg
8. end = i
9. if cur\_sum <= 0
10. cur\_sum = 0
11. cur\_beg = i+1
12. return (beg, end, max\_sum)

## 矩阵乘法的Strassen

## 用带入法求解递归式

## 用递归树方法求解递归式

## 用主方法求解递归式

## 证明主定理

## 思考题

#### 芯片检测

**Algorithm** 芯片两两配对检测。若检测结果为情况1，则任选一个进入下一轮。其他情况全部丢弃。设情况1共出现k对，其他情况共出现m对。当n是奇数时会剩下一个芯片。若k是奇数，则丢弃该芯片。若k是偶数，则该芯片进入下一轮。

**Claim** 进入下一轮的芯片中好的芯片多于坏的芯片。

**Proof** 分n奇偶讨论。设n=2(k+m)，且情况1中有k1对芯片全好，k2对全坏, k=k1+k2。设坏的芯片数为s，则k+m>s>=m+2k2. 故k1>k2.

现在考虑n=2(k+m)+1。此时k+m>=s>=m+2k2，即k1>=k2。若k奇数，必有k1>k2，此时丢弃未配对的芯片。若k偶数，则应该让该芯片进入下一轮。这是因为，若该芯片是好芯片，则下一轮中好芯片数=k1+1>k2=坏芯片数。若芯片是坏芯片，则k+m>=s>=m+2k2+1，即k1>=k2+1. 注意到k1+k2是偶数，则必有k1>=k2+2. 于是下一轮中好芯片数=k1>k2+1=坏芯片数.

#### Monge阵列

1. 证明：一个数组是Monge阵列当且仅当对所有i=1, 2, ..., m-1和j=1, 2, ..., n-1，有

**Proof** 必要性显然.

充分性. 设. 对任意i，依次用j+1, j+2, …, i-1替换上式的j得到

上面不等式叠加即有

再用i+1, i+2, …, k-1 替换上式的i得到

不等式叠加即有

由此该数组是Monge阵列.

1. 下面数组不是Monge阵列. 改变一个元素使其变成Monge阵列.

**Answer** 改变A[2,3].

1. 令f(i)表示第i行的最左最小元素的列下标. 证明：对任意的Monge阵列，.

**Proof** 对任意, 设f(i)=j1, f(i+1)=j2. 若j1>j2, 则. 于是，与Monge阵列的定义矛盾. 故.

1. 下面是一个计算m\*n的Monge阵列A的每一行最左最小元素的分治算法的描述：提取A的偶数行构造其子矩阵A'。递归地确定A'每行的最左最小元素。然后计算A的奇数行的最左最小元素。

解释如何在O(m+n)时间内计算A的奇数行的最左最小元素（在偶数行的最左最小元素已知的情况下）。

**Answer** . 在已知的条件下计算需次比较，计算需次比较

1. 给出(d)中描述的算法的运行时间的递归式。证明其解为O(m+nlogm)。
2. 概率分析和随机算法

## 雇用问题

描述Random(a,b)过程的一种实现，它只调用现有实现Random（0，1）。作为a和b的函数，你的程序的期望运行时间是多少？

**Algorithm** 用Random(n)表示均匀随机输出0到n-1间的某数. 设.

Random(n)

1. call Random(0,1) k times to produce , is the result of the ith call
2. if , return r
3. else go back to 1

**Claim** 对任意，Random(n)以概率1/n返回r

**Proof** 若，显然成立. 否则，返回r的概率p满足，解得.

**Time Complexity** Random(0,1)的期望调用次数为

假设你希望以各1/2的概率输出0和1。你可以自由使用一个输出0或1的过程BIASED-RANDOM。它以概率p输出1，以概率1-p输出0，其中0<p<1，但是你并不知道p的值。给出一个利用BIASED-RANDOM作为子程序的算法，返回一个无偏向的结果。你的算法的期望运行时间是多少？

**Algorithm** Random()

1. a=BIASED-RANDOM(), b=BIASED-RANDOM()
2. if a>b return 0, a<b return 1
3. if a=b, go back to 1

**Claim** Random()以概率1/2返回0，以概率1/2返回1

**Proof** 返回1的概率p(1)满足，解得

**Time Complexity** BIASED-RANDOM的期望调用次数为

## 指示器随机变量

Answer 正好雇佣一次的概率1/n，正好雇佣n次的概率1/n!

Answer 考虑恰好雇佣两次的排列。设最大值出现的位置是k+1，则要求最大值前面的k个元素中的最大值排在首位，其余k-1个任意排列，最大值后面的n-k-1个元素也是任意排列，故此时排列方案数为。正好雇佣两次的概率为

Answer 定义指示器随机变量，则。拿到自己帽子的顾客数，则

### Inversion

Answer 对任意定义指示器随机变量，则。逆序对数目，则

## 随机算法

### PERMUTE-WITHOUT-INDENTITY

Answer 尽管不会产生恒等排列，但是有些其他排列也不能产生。以n=3为例，除恒等排列共有5种排列，但该算法只能产生2种排列

### PERMUTE-WITH-ALL

Answer 不会产生均匀随机排列。算法第一次循环产生n个可能排列，对于这每个排列，算法在第二次循环又会产生n个排列，以此类推，n次循环后共产生个排列，但只有n!种不同的排列。如果要求产生均匀随机排列，则个排列中每个不同排列出现次数要相同，从而是n!的倍数，这是显然不成立的。故不能产生均匀随机排列。

### PERMUTE-BY-CYCLE

Answer A[i]在B中的位置。因为offset在[1, n]上均匀分布，故对任意i，dest也是在[1, n]上均匀分布。故A[i]出现在B中任何特定位置的概率是1/n。

该算法只能产生n种不同的排列，显然不是均匀随机排列。

证明：在过程PERMUTE-BY-SORTING的数组P中，所有元素都唯一的概率至少是1-1/n。

Answer

### RANDOM-SAMPLE

**Proof** RANDOM-SAMPLE(m, n)会递归调用RANDOM-SAMPLE(m-1, n-1)，后者又会调用RANDOM-SAMPLE(m-2, n-2), 直到RANDOM-SAMPLE(1, n-m+1)调用RANDOM-SAMPLE(0, n-m)，而后者直接返回空集，递归结束。我们归纳证明，对所有的，RANDOM-SAMPLE(t, n-m+t)返回集合{1，2，…, n-m+t}的均匀随机t组合。

当t=1时，算法第3行的，第4行i=RANDOM(1, n-m+1)。if分支中算法执行第7行，故算法返回的S是集合{1，2，…, n-m+1}均匀随机1组合。

假设情况是t时结论成立，当情况t+1时，根据归纳假设，第3行的S是集合{1，2，…, n-m+t}的均匀随机t组合。为简便起见，令k=n-m+t，并把第8行的返回值记为。考虑{1，2，…, k, k+1}的任意t+1组合T。若，则

若，则

无论哪种情况都有，从而证明了是{1，2，…, n-m+t+1}的均匀随机t+1组合。由此结论在t+1时成立。

## 概率分析和指示器随机变量的进一步使用

Answer 记n=365。假设包括自己有k+1个人，则没有人生日与自己生日相同的概率是。令，则=252.7。设屋里共有k个人，至多两人生日在7月4号的概率满足

Answer 对k=2, 3, …, b+1，投掷次数为k的概率为。

故投掷次数的期望是

Answer 考虑空箱子的数目X。定义指示器随机变量，则且。故。类似地，考虑正好有一个球的箱子数目Y，定义指示器随机变量，则且。故。

## 思考题

#### 5-2 查找无序数组

1. RANDOM-SEARCH (A, x)
2. let checked[1..n] be a new array with each element initialized as false
3. unchecked=n
4. while(unchecked>0)
5. j=RANDOM (1, n)
6. if A[j]==x
7. return j
8. if checked[j]==false
9. checked[j]=true
10. unchecked=unchecked-1
11. 期望查找次数n
12. 期望查找次数n/k
13. 礼券收集者问题，期望次数是
14. 平均运行时间，最坏运行时间n
15. 查找次数为i的概率是，i=1,2,…,n-k+1

平均运行时间是

最坏运行时间n-k+1

1. 平均运行时间n, 最坏运行时间n
2. k=0 平均运行时间n, 最坏运行时间n

k>=1 平均运行时间(n+1)/(k+1), 最坏运行时间n-k+1

1. 使用DETERMINISTIC-SEARCH
2. 堆排序

## 堆

Answer

Answer

Answer 按子树高度归纳证明

Answer 最大堆的最小元素在叶结点上

Answer 已排序好的数组是最小堆

Answer 不是，6是7父节点但6<7

## 维护堆的性质

Answer 当A[i]比其孩子的值都大时，MAX-HEAPIFY(A, i)无操作

Answer 当i>A.heapsize/2时，MAX-HEAPIFY(A, i)无操作

Answer 当MAX-HEAPIFYF在根结点是0其余结点都是1的堆上从根结点开始运行时，会恰好被调用h次，h是堆的高度。

## 建堆

Answer MXA-HEAPIFY(A, i)运行的前提条件是结点i的左右子树都已是最大堆

Proof 设堆的高度为H，则高度为h的结点数最多为。由于，故

## 堆排序算法

Proof 考虑按降序排列的数组，BUILD-MAX-HEAP过后数组保持不变。

## 优先队列

### 最小优先队列

Answer 关键字设置成使得算法2行执行后A[1..A.heap-size]仍是最大堆，且key>A[A.heap-size]成立。由此HEAP-INCREASE-KEY(A, A.heap-size, key)正确操作。

### HEAP-INCREASE-KEY的正确性

初始化：在增大A[i]之前，对任意非根结点j，最大堆性质成立。A[i]增大至key后，最大堆性质受影响的结点仅可能是i的子节点和i。但对i的子结点而言，其父节点增大，最大堆继续成立。唯一有可能违背的情况发生在：i非根节点且。

保持：若循环前检测到i非根节点且，则算法第5行交换二者的值，交换后i获得其父节点的值，根据循环不变式，i必大于等于其孩子结点，此时PARENT(i)也大于等于其左右孩子结点。PARENT(i)增大，算法第6行令i= PARENT(i)后，i可能大于其父节点，而其余结点满足最大堆性质。

终止：算法终止时或者i=1或者，故所有结点都满足最大堆性质。

Answer line3删除，line4~line6修改为

4 while i>1 and A[PARENT(i)]<key

5 A[i] =A[PARENT(i)

6 i=PARENT (i)

7 A[i]=key

HEAP-DELETE (A, i)

1. while i>1 and A[A.heap-size]>A[PARENT(i)]
2. A[i]=A[PARENT(i)]
3. i=PARENT(i)
4. A[i]=A[A.heap-size]
5. A.heap-size = A.heap-size - 1
6. MAX-HEAPIFY(A, i)

## 思考题

#### 6-1用插入的方法建堆

1. 不一定相同。比如当数组是[1,2,3]时，BUILD-MAX-HEAP返回[3,2,1]，BUILD-MAX-HEAP’返回[3,1,2]
2. 算法循环n-1次，每次运行时间O(logn)，故时间上界是O(nlogn)。当输入数组严格增时，对每个i, MAX-HEAP-INSERT(A, A[i])恰好进行次比较。

#### 6-2 d叉堆

#### 6-3 Young氏矩阵

1. 矩阵其余7个元素填充为。
2. 是矩阵Y的最小元，是矩阵Y的最大元。若，则所有元素都是。若，则所有元素都是有限值。
3. Maintain-Young(Y, i, j)
4. while i<Y.m and j<Y.n
5. if Y[i, j]>Y[i+1, j] and Y[i+1, j]<= Y[i, j+1]
6. exchange Y[i, j] with Y[i+1, j]
7. i = i +1
8. else if Y[i, j]>Y[i, j+1] and Y[i+1, j]>= Y[i, j+1]
9. exchange Y[i, j] with Y[i, j+1]
10. j = j+1
11. else return
12. if i==Y.m
13. while j<Y.n and Y[i, j]<Y[i, j+1]
14. exchange Y[i, j] with Y[i, j+1]
15. j= j + 1
16. else
17. while i<Y.m and Y[i, j]<Y[i+1, j]
18. exchange Y[i, j] with Y[i+1, j]
19. i = i + 1

EXRTACT-MIN(Y)

1. min = Y[1, 1]
2. Y[1, 1] = Y[Y.m, Y.n]
3. Y[Y.m, Y.n] =
4. Maintain-Young(Y, 1, 1)
5. return min
6. INSERT(Y, key)
7. i = m, j = n
8. while i>1 and j>1
9. if key <Y[i, j-1] and Y[i, j-1]>=Y[i-1, j]
10. Y[i, j] = Y[i, j-1]
11. j = j-1
12. else if key<Y[i-1, j] and Y[i, j-1]<=Y[i-1, j]
13. Y[i, j] == Y[i-1, j]
14. i = i -1
15. else
16. Y[i, j] = key
17. return
18. if i ==1
19. while j>1 and key<Y[i, j-1]
20. Y[i, j] = Y[i, j-1]
21. j = j-1
22. else
23. while i>1 and key<Y[i-1, j]
24. Y[i, j] = Y[i-1, j]
25. i = i-1
26. Y[i, j] =key
27. Sort(Y)
28. for i=1 to n
29. for j=1 to n
30. INSERT(Y[1:i, 1:j], Y[i, j])
31. let A[1..] be a new array
32. for i=1 to
33. A[i] = EXRTACT-MIN(Y)
34. SEARCH(Y, key)
35. i =1, j= n
36. while i<=m and j>=1
37. if Y[i, j] == key
38. return True
39. else if Y[i, j]<key
40. i = i + 1
41. else j = j – 1
42. return False
43. 快速排序

## 快速排序的描述

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 13 | 19 | 9 | 5 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 |
| 9 | 19 | 13 | 5 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 |
| 9 | 5 | 13 | 19 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 |
| 9 | 5 | 8 | 19 | 12 | 13 | 7 | 4 | 21 | 2 | 6 | 11 |
| 9 | 5 | 8 | 7 | 12 | 13 | 19 | 4 | 21 | 2 | 6 | 11 |
| 9 | 5 | 8 | 7 | 4 | 13 | 19 | 12 | 21 | 2 | 6 | 11 |
| 9 | 5 | 8 | 7 | 4 | 2 | 19 | 12 | 21 | 13 | 6 | 11 |
| 9 | 5 | 8 | 7 | 4 | 2 | 6 | 12 | 21 | 13 | 19 | 11 |
| 9 | 5 | 8 | 7 | 4 | 2 | 6 | 11 | 21 | 13 | 19 | 12 |

**Answer** 当A[p..r]所有元素都相同时，PARTITION返回r。

### PARTITION运行时间

**Proof** line1, line2, lin7各耗时。line3~line6的循环，line4执行n次，line5和line6至多执行n次。故总的执行时间为。

### 降序排序

**Answer** line4改成 if A[j]>=x。

## 快速排序的性能

Proof 若，则

且。取充分小的和充分大的即可使。

Answer 当数组所有元素都相同时，快速排序复杂度为

Proof 设快速排序在按降序排列的数组A[1..n]上运行时间T(n)。第一次调用PARTITION会返回1，同时A[1]与A[n]交换，其余元素不变。接下来算法对子数组A[2,n]进行排序，此时PARTITION返回n，且没有元素交换。于是算法接着对按降序排列的数组A[2,n-1]排序。故。故。

**Answer** 在递归树的第k层，恰好存在k+1种规模的结点：，。由于，规模最大的结点是，规模最小的结点是。规模最小的结点最先达到1，令，得到递归树的最小深度。规模最大的结点最后达到1，令，得到递归树的最大深度。

**Proof** 设PARTITION把大小为n的数组分为q和n-q的两部分，q服从[1, n]上的均匀分布。当时，令，则。故当时，划分比更平衡。此概率为。

## 快速排序的随机化版本

**Answer** We may be interested in the worst-case performance, but in that case, the randomization is irrelevant: it will not improve the worst case. What randomization can do is make the chance of encountering a worst-case scenario small.

**Answer** 最坏情况是n-1次，最好情况是次。

## 快速排序分析

Proof 假设，则。对给定的，只需取满足就有。故。

**Proof** 设是最好情况下快速排序在规模为n的数组上运行时间，则有递归式：

猜测对任意都有，当时显然成立。当带入上式有

对给定的，只须取充分小的c满足对任意都有即可有。由此证明了。

**Proof** 考虑函数。。故在单调递减，在上单调递增。最大值为。

**Proof** 首先说明PARTITION至少会被调用次，否则被选为主元的元素数目k至多为。设这k个主元排序为，考虑排在之前，与之间，之后这k+1个位置的元素数目。必在某个位置存在两个或两个以上的元素。这是因为，若在所有位置至多存在1个元素，则元素数目之和最大为，矛盾。设在某个位置存在两个及以上元素，这些元素中没有主元，故而它们之间没有排序，也就是算法并未正确排序。故PARTITION至少会被调用次。因此，类似于引理7.1，我们可以得到快速排序运行时间为。其中X满足

当时有。故快速排序的期望运行时间。

**Answer** 不妨设。设PARTITION把大小为n的数组分为q和n-q的两部分，。对于给定的q, 这种划分的概率是。当时，划分比更平衡。记，则当时，。获得比更平衡划分的概率为

原来的划分对应的概率约为。注意到，故三数取中划分得到平衡划分的概率更大。

## 思考题

#### 7-1 Hoare划分

1. 操作过程如下

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 13 | 19 | 9 | 5 | 12 | 8 | 7 | 4 | 11 | 2 | 6 | 21 |
| 13 | 19 | 9 | 5 | 12 | 8 | 7 | 4 | 11 | 2 | 6 | 21 |
| 6 | 19 | 9 | 5 | 12 | 8 | 7 | 4 | 11 | 2 | 13 | 21 |
| 6 | 2 | 9 | 5 | 12 | 8 | 7 | 4 | 11 | 19 | 13 | 21 |

1. line4~line13的循环首次迭代时，line8~line10执行后，。line5~line7执行后，j要么等于，要么等于某个，。如果，循环结束，此过程中和显然都没有越界。如果，则断言当循环结束时有且，从而和都没有越界。首次迭代会交换与的值，交换后，。由于每次迭代至少加1，至少减1，故此后的迭代过程中必有且。假设循环中可以增加到大于或者j可以减小到小于，那么必有某次迭代，line8~line10执行后，或者某次迭代，line5~line7执行后。无论哪种情况，在line11执行时一定有，这会导致循环结束。故决不可能增加到大于且j决不可能减小到小于。
2. 上面已经证明循环结束时。下面说明。初始时若，则首次迭代后就有，此后每次迭代至少减1。若，则首次迭代后且。于是，line12执行，而后进入第二次迭代，至少减1，故循环结束后。
3. **循环不变式**：每次迭代在line10执行后都有对数组下标，若，则，若，则。

**初始化**：首次迭代后显然成立。

**保持**：在某次迭代line10执行后，。若循环不变式成立且，则line12交换与，交换后，。下次迭代时line10执行后增加到，减小到，则对任意有，任意有。故对数组下标，若，则，若，则，即循环不变式继续成立。

**终止**：终止时且。按照循环不变式，若，则，若，则。由此中的每个元素都小于等于中每个元素。

1. QUICKSORT(A, p, r)
2. if p < r
3. j=HOARE-PARTITION(A, p, r)
4. QUICKSORT(A, p, j)
5. QUICKSORT(A, j+1, r)

#### 7-2 针对相同元素值的快速排序

1. 算法动态维护下标。：小于主元的分区。：等于主元的分区。：大于主元的分区。：尚未划分的部分。

PARTITION’(A, p, r)

1. x = A[r]
2. i = j = p
3. for k = p to r
4. key = A[k]
5. if key < x
6. A[k] = A[j]
7. A[j] = A[i]
8. A[i] = key
9. i = i +1，j = j +1
10. else if key == x
11. A[k] = A[j]
12. A[j] = key
13. j = j + 1
14. return (i, j-1)
15. QUICKSORT’(A, p, r)
16. if p < r
17. (q, t) = RANDMIZED-PARTITION’(A, p, r)
18. QUICKSORT’(A, p, q-1)
19. QUICKSORT’(A, t+1, r)
20. 假设原始数组互不相同的元素有个，从小到大依次为，并设的数目为，这个值为的元素记为。设，同样定义为所有值在与之间的元素集合。于是与进行比较当且仅当中被选为主元的第一个元素是或，此概率为。设，与进行比较当且仅当中被选为主元的第一个元素是或，此概率为。故比较次数的期望为

#### 7-3 另一种快速排序的分析方法

1. 定义随机变量R为主元的顺序统计量，则。于是
2. Proof

最后一步是因为可移入。

1. Proof 一方面，

另一方面

1. 假设存在正常数c使对任意有，那么

只须取就有。

假设存在正常数c使对任意有，那么

只要充分小就有对就有任意使得，从而。

由此证明了

#### 7-4 快速排序的栈深度

1. TAIL-RECURSIVE-QUICKSORT(A, p, r)与QUICKSORT(A, p, r)对数组A的操作完全相同。QUICKSORT(A, p, r)对A[p..r]先做PARTITION并返回q, 然后以参数(A, p, q-1)和参数(A, q+1, r)调用自身。TAIL-RECURSIVE-QUICKSORT(A, p, r)也是对A[p..r]先做PARTITION并返回q, 然后以参数(A, p, q-1)调用自身。接着设置p=q+1然后再次循环，这就相当于以参数(A, q+1, r)调用自身。
2. 考虑已按序排列的数组，每次PARTITION(A, p, r)都返回r。于是递归调用序列就是

TAIL-RECURSIVE-QUICKSORT (A, 1, n)

TAIL-RECURSIVE-QUICKSORT (A, 1, n-1)

…

TAIL-RECURSIVE-QUICKSORT (A, 1, 1)

直到调用至TAIL-RECURSIVE-QUICKSORT (A, 1, 1)才开始返回。此时栈深度。

1. 把数组划分成两个子数组后，应当先对短的子数组进行递归调用，该子数组长度小于划分前的一半，如此至多经过次递归调用必然调用至TAIL-RECURSIVE-QUICKSORT (A, p, p)，然后返回至上一层调用点。此过程最大栈深度为。注意栈深度可以是，因为在最坏情况下，每次划分都等分数组，则最大栈深度恰为。

TAIL-RECURSIVE-QUICKSORT (A, p, r)

1. while p < r
2. q = PARTITION(A, p, r)
3. if q – p < r – q
4. TAIL-RECURSIVE-QUICKSORT (A, p, q)
5. p = q + 1
6. else
7. TAIL-RECURSIVE-QUICKSORT (A, q+1, r)
8. r = q – 1

期望运行时间显然不变，因为修改前后的算法对数组A的操作完全相同，只是在一些子数组间的操作顺序上有差异。

#### 7-5 三数取中划分

1. 与练习7.4-6相同的分析
2. 。直接划分的概率是。

极限情况下中位数被选中的概率增加了50%。

1. 直接划分得到好划分的概率是1/3。三数取中划分相应的概率是

三数取中划分得到好的划分的概率增加了

1. 三数取中法在最好情况下每次都把数组等分，这意味着其复杂度存在下限，从而其期望运行时间为。三数取中比直接划分得到平衡划分的概率更大，而直接划分的期望运行时间，这意味着三数取中法期望运行时间为。由此说明三数取中法期望运行时间为，与直接划分的快速排序相同，只是其蕴含的常数项因子更小。

#### 7-6 对区间的模糊排序

1. 使用思考题7-2的划分，即是将数组划分为小于主元，等于主元和大于主元的三部分。设主元是区间，任给区间，定义区间的偏序关系：当且仅当。于是
2. 若，则进入小于主元的部分
3. 若，则进入大于主元的部分
4. 若且（即区间有重叠），则进入等于主元的部分

注意，区间的关系是具有传递性的，但是重叠关系是没有传递性的。比如令，则与有重叠，与有重叠，但。也就是说，上述划分等于主元的区间是不能任意排列的。什么情况下可以任意排列呢？那就是当且仅当所有区间都重叠时。如何保证等于主元的区间都重叠呢？上述划分的第3种情况应改为

假设当前主元区间是。若且（即区间有重叠），则进入等于主元的部分且将作为新的主元。

注意动态的缩小主元区间对小于主元的区间和大于主元的区间相对于初始主元区间的排列正确性没有影响。即假设当前主元区间，与主元进行比较的区间。若是个正确的排列，则也是个正确的排列。

如此，所有等于主元的区间之交一定非空，从而其任意排列都是符合要求的排列，这时算法只须递归处理小于主元和大于主元的区间。

1. 该算法与思考题7-2的排序完全类似，故期望运行时间为。随着重叠区间的增多，需递归排序的子区间组长度会减小，运行时间会变短。特别地，当所有区间都重叠时，划分得到的两个子区间组长度都是0，算法就此结束，运行时间为。
2. 线性时间排序

## 排序算法的下界

Answer 最小深度是n-1。考虑插入排序，当初始数组已按序排列时，决策过程为1:2，2:3，…，n-1:n，相应的叶结点为<1, 2, …, n>，该结点的深度恰为n-1。

Answer 一方面，

另一方面，存在正常数c，，只要，就有

故

Proof 假设某比较排序算法能对n!种长度为n的输入的种实现线性时间排序，考虑其决策树中由这个叶结点及其所有祖先结点构成的子树，则该子树的高度为。但是一棵有个叶结点的二叉树，无论什么形态，其高度必然满足。于是

所有情况下都有，这与子树高度矛盾，故不可能实现线性时间排序。

Proof 待排序序列共有种可能的排列。故任何能对该序列正确排序的决策树至少含有个叶结点。该决策树高度满足

即，从而所需比较次数的下界是。

## 计数排序

Answer

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|  | 6 | 0 | 2 | 0 | 1 | 3 | 4 | 6 | 1 | 3 | 2 |

初始化数组C:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|  | 2 | 2 | 2 | 2 | 1 | 0 | 2 |

数组C累和：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|  | 2 | 4 | 6 | 8 | 9 | 9 | 11 |

排序：

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|  | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 6 | 6 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|  | 0 | 2 | 4 | 6 | 8 | 9 | 9 |

Proof 设数组A中有k个值为某给定值v的元素，其下标为。设line8执行后，则稳定排序要求的位置为，。而line10将j从A.length递减至1恰好满足这一点。当时，算法先是，而后。当时就有，而后。依次类推，最后。

Proof j按照任意次序遍历1到A.length都能正确排序，但当数组存在相等元素时，只有当j以递减次序遍历时才是稳定的。如果j按递增次序遍历，则相等元素排序后的次序与它们初始次序正好相反。

Answer 考虑line7-8执行后的数组C。落在区间的元素个数为，其中。

## 基数排序

Answer

COW

DOG

SEA

RUG

ROW

MOB

BOX

TAB

BAR

EAR

TAR

DIG

BIG

TEA

NOW

FOX

SEA

TEA

MOB

TAB

DOG

RUG

DIG

BIG

BAR

EAR

TAR

ROW

COW

NOW

BOX

FOX

TAB

BAR

EAR

TAR

SEA

TEA

DIG

BIG

MOB

DOG

ROW

COW

NOW

BOX

FOX

RUG

BAR

BIG

BOX

COW

DIG

DOG

EAR

FOX

MOB

NOW

ROW

RUG

SEA

TAB

TAR

TEA

Answer 插入排序和归并排序是稳定的。为使任何排序算法都稳定，可将键值连同下标一块排序。当键值相同时比较下标。额外的空间开销：下标范围1~n，每个下标可用位存储，累计需要存储空间。渐近运行时间不变，因为只是每次比较增加了常量时间。

Proof 按照数的位数d归纳证明。d=1时显然成立。假设基数排序在d-1位数上正确运行。当基数排序在d位数上运行时，依次按照第1位、第2位直到第d-1位使用底层算法排序，这就相当于按照每个数的低d-1位进行基数排序。然后按照最高位使用底层排序算法排序。考虑两个元素和，其最高位记为和。

若，则底层排序算法将排在前面。这是正确的，因为。

若，则底层排序算法将排在前面。这是正确的，因为。

若，按照归纳假设，和已按低d-1位排好次序，这就是它们的正确相对次序。同时底层排序算法是稳定的，不会改变和的相对次序，故排序正确。

如果底层排序算法不是稳定的，那么当时可能会改变和的相对次序，而原先和已处在正确的相对次序上。

Answer 将每个整数写n进制形式，该形式下每个整数恰有3位，每一位的取值范围0到n-1。使用基数排序，调用3轮，每轮耗时，整个算法耗时。

Answer

## 桶排序

|  |  |
| --- | --- |
| 0 |  |
| 1 | 0.13 0.16 |
| 2 | 0.20 |
| 3 | 0.39 |
| 4 | 0.42 |
| 5 | 0.53 |
| 6 | 0.64 |
| 7 | 0.71 0.79 |
| 8 | 0.89 |
| 9 |  |

Answer 极端情况下所有元素都进入同一个桶，此时插入排序耗时。对每个桶里的元素使用复杂度的排序算法可以保证平均情况为线性时间代价且最坏情况下代价为。

Answer

Answer 把单位圆划分为n个等面积的同心圆环，由内到外这些圆环的外圆半径为。第i个圆环的外圆面积，，故。定义n个桶，存放落在第i个圆环的点，即到原点的距离d在内的点。注意到

即将点置于桶中。如此在均匀分布的假设下，每个点以相等的概率落在各个桶中。故期望运行时间为。

Answer这是桶排序的一般形式，同样需要设计n个桶，使得随机变量等概率的落入各个桶。设且，。定义n个桶，存放落在区间内的点。这时随机变量落在桶的概率为。给定随机变量的一个取值，假设可以在时间内确定所属的区间，不妨设为，则将置于桶中，由此实现了桶排序。

## 思考题

#### 8-1 比较排序的概率下界

1. 一方面，没有两个不同的排列可以到达同一叶结点，故可达叶结点数至少为。另一方面，A是确定性算法，故某给定排列只会到达某固定叶结点，故可达叶结点数至多为。故可达叶结点数恰有个可达叶结点，与个排列一一对应，每个叶结点到达的概率为，其余叶结点是不可达的。
2. 考虑每个叶结点对等式两边的贡献。每个叶结点只对D(LT)和D(RT)之一有贡献，且其贡献比对D(T)的贡献小1。所有叶结点对D(LT)和D(RT)的贡献之和则比D(T)小。
3. 一方面，任取某个有个叶结点的决策树，其左右子树的叶结点数各为和，。假设其左右子树都是给定叶结点数下外部路径长度最小的决策树，则。按照的定义，对任意，有即，从而。

另一方面，假设决策树T的外部路径长度达到了最小值，其左右子树的叶结点数各为和，则必然有。如果或，即全部k个叶结点在单个子树上，比如RT上，则有，矛盾。于是，从而。

所以。

1. 考虑函数，则。故在上递减，在上递增，在处有最小值。

下面归纳证明，对任意，存在正常数c，使得。

*，*，命题成立。

当时，假设对任意对任意成立，则

故只要取就有。归纳完成。

故。

1. 的可达叶结点数为且不妨假设只有可达叶结点。。平均每个叶结点的路径长度即排序的平均运行时间为

即排序的平均运行时间。

1. 我们可以对随机排序算法的决策树做修改得到一个确定算法的排序树。修改方法是，在每个随机化结点，选择其若干个子树中平均比较次数最少的子树，其余子树删除。

能正确排序，因为随机算法中沿着随机结点的任意子节点都能正确排序。

的平均比较次数不超过，因为我们总是舍弃了平均比较次数较高的子树。具体的说，每次修改随机化结点时总是保持整个树的平均比较次数不增。修改前后，总是相同的输入排列到达该处结点，但是修改后处理这些排序的平均比较次数不增，而同时树的其余部分保持不变。

注意到的平均运行时间为，故任意随机算法的平均算法也是。