第 2 章 算法基础

2.1 插入排序

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

$T(n) = \Theta(n^2)$

Ex2.1-2 重写过程 INSERTION-SORT, 使之按非升序排序.

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j] A[1 .. j − 1].
    i = j − 1
    while i > 0 and A[i] < key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

Ex2.1-3 线性查找问题.

```
LINEAR-FIND(A, v)
for j = 1 to A.length
    if A[j] == v
        return j
return NIL
```

Ex2.1-4 二进制数加法

BINNARY-NUMBER-ADDITION(A, B, n)

Let $C[1\,..\,n+1]$ be a new array.
$r = 0$
for $i = n$ **downto** 1
  $C[i] = (A[i] + B[i] + r)\%2$
  $r = (A[i] + B[i] + r)/2$
$C[1] = r$
return C

2.2 分析算法

Ex2.2-2: 选择排序 首先找到 A 中最小元素并与 A[1] 进行交换, 然后找出次最小元素并与 A[2] 进行交换...

SELECTION-SORT(A)

for $i = 1$ **to** $A.length - 1$
  $min = i$
  for $j = i + 1$ **to** $A.length$
    if $A[j] < A[min]$
      $min = j$
  Exchange $A[min]$ and $A[i]$

2.3 设计算法

MERGE(A, p, q, r)

$n_1 = q - p + 1$
$n_2 = r - q$
let $L[1\,..\,n_1+1]$ and $R[1\,..\,n_2+1]$ be bew arrays
for $i = 1$ **to** n_1
    $L[i] = A[p+i-1]$
for $j = 1$ **to** n_2
    $R[j] = A[q+j]$
$L[n_1+1] = \infty$
$R[n_2+1] = \infty$
$i = 1$
$j = 1$
for $k = p$ **to** r
    if $L[i] \leq R[j]$
        $A[k] = L[i]$
        $i = i + 1$
    else
        $A[k] = R[j]$
        $j = j + 1$

MERGE-SORT(A, p, r)

if $p < r$
    $q = \lfloor (p+r)/2 \rfloor$
    MERGE-SORT(A, p, q)
    MERGE-SORT$(A, q+1, r)$
    MERGE(A, p, q, r)

利用递归树, 可以得知 MERGE-SORT 的算法复杂度为 $\Theta(nlgn)$

Ex2.3-2 重写过程 MERGE-SORT, 使之不使用哨兵, 而是一旦数组 L 或 R 的所有元素均被复制回 A 就立刻停止, 然后把另一个数组的剩余部分复制回 A.

Merge-Sort-Without-Guard(A, p, q, r)

```
n1 = q - p + 1
n2 = r - q
Let L[1..n1] and R[1..n2] be new arrays.
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
i = 1
j = 1
k = p
while i ≤ n1 and j ≤ n2
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else
        A[k] = R[j]
        j = j + 1
    k = k + 1
while i ≤ n1
    A[k] = L[i]
    i = i + 1
while j ≤ n2
    A[k] = R[j]
    j = j + 1
```

Ex2.3-4 我们可以把插入排序表示为如下的一个递归过程。为了排序 $A[1\,..\,n]$, 我们递归地排序 $A[1\,..\,n-1]$, 然后把 $A[n]$ 插入已排序的数组 $A[i\,..\,n-1]$。为插入排序的这个版本的最坏情况运行时间写一个递归式。

Insertion(A, p)

```
key = A[p]
i = p - 1
while i > 0 and A[i] < key
    A[i + 1] = A[i]
    i = i - 1
A[i + 1] = key
```

```
INSERTION-SORT-RECURSIVE(A, p)
if p > 1
    INSERTION-SORT-RECURSIVE(A, p − 1)
    INSERTION(A, p)
```

其递归式如下：

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(n-1) + \Theta(n) & \text{if } n = 2 \end{cases}$$

Ex2.3-5 假设 A 已排好序，为二分查找写出迭代式或递归的伪代码。证明二分查找的最坏情况运行时间为 $\Theta(lgn)$

```
BINARY-SEARCH-RECURSIVE(A, key, low, high)
if low ≤ high
    mid = ⌊(low + high)/2⌋
    if key < A[mid]
        return BINARY-SEARCH-RECURSIVE(A, key, low, mid − 1)
    elseif key > A[mid]
        return BINARY-SEARCH-RECURSIVE(A, key, mid + 1, high)
    else
        return mid
else
    return NIL
```

该算法的递归式为 $T(n) = T(n/2) + \Theta(1)$，由递归树可知，其算法复杂度为：$\Theta(lg(n))$

Ex2.3-6 注意到过程 INSERTION-SORT 的第 5-7 行的 while 循环采用一种线性查找来 (反向) 扫描已排好序的子数组 $A[1\,..\,j-1]$。我们可以使用二分查找来把插入排序的最坏情况总运行时间改进到 $\Theta(nlgn)$ 吗？

先将上面的 BINARY-SEARCH-RECURSIVE 算法修改为如下形式:

```
BINARY-SEARCH-RECURSIVE(A, key, low, high)
if low ≤ high
    mid = ⌊(low + high)/2⌋
    if key < A[mid]
        return BINARY-SEARCH-RECURSIVE(A, key, low, mid − 1)
    elseif key > A[mid]
        return BINARY-SEARCH-RECURSIVE(A, key, mid + 1, high)
    else
        return mid
else
    return low
```

```
INSERTION-SORT-USING-BINARY-SEARCH(A)
for j = 2 to A.length
    key = A[j]
    i = j − 1
    index = BINARY-SEARCH-RECURSIVE(A, key, 1, i)
    if A[index] < key
        index = index + 1
    while i ≥ index
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

可见，在最坏情况下，就算使用了 BINARY-SEARCH-RECURSIVE 算法，也不能将其运行时间降低到 $\Theta(nlgn)$。