算法复习提纲————NO.1

算法复杂度

- Insertion Sort $O(n^2)$
- Merge Sort $O(n\log(n))$
- HeapSort $O(n\log(n))$
- QuickSort $O(n\log(n))$ `Average`
- Sorting in linear time

`Insertion Sort——插入排序（菜鸟级）`

算法总体思想：首先将需要插入的元素插入序列的末尾，然后从原始序列的最后一个元素开始，每个元素后移一个，直到找到新元素所在的位置，停止，并且将新元素插入到该位置上

```
for i = 2 to length(A)
    do key = A[j]
    #把A[j]插入到已经拍好序的A[1...j-1]序列中
    i = j
    while i>0 and A[i]>key
        do A[i+1] = A[i]
           i = i - 1
    A[i+1] = key #找到插入位置
```

`Merge Sort——归并排序（递归）`

基本思想：分治法，将数组分成若干个子数组分别进行排序，再将排好序的子数组进行合并

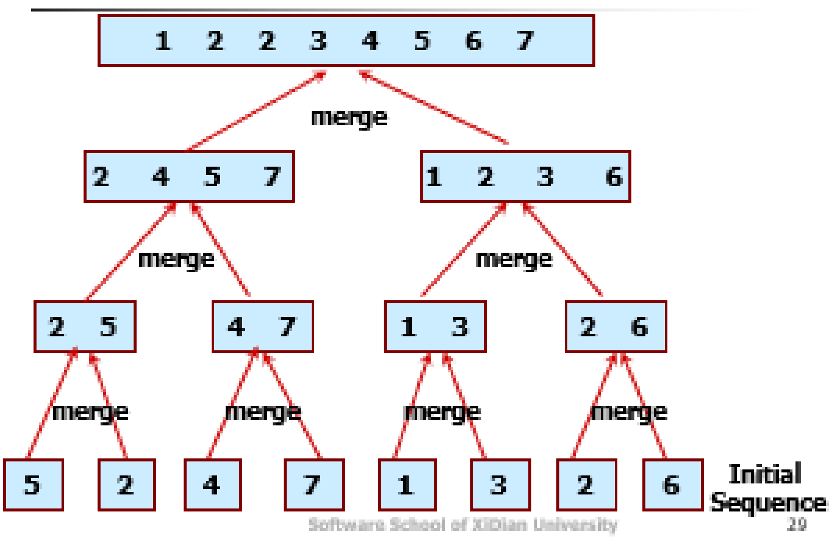
`INPUT` a sequence of n numbers stored in array A

`OUTPUT` an ordered sequence of n numbers

```
MERGE_SORT(A, p, r)
    if p < r
```

```
        then q= [(p+r)/2] #取下届
        MERGE_SORT(A,p,q)
        MERGE_SORT(A,q+1,r)
        MERGE(A,p,q,r)

MERGE(A,p,q,r)
    n1 = q - p + 1
    n2 = r - q
    create arrays L[1...n1+1] and R[1...n2+1]
    for i = 1 to n1
        do L[i] = A[p+1-1]
    for j = 1 to n2
        do R[j] = A[q+j]
    L[n1+1] = MAX #代表无穷
    R[n2+1] = MAX
    i = 1
    j = 1
    for k = p to r
        do if L[i] <= R[j]
            then A[k] = L[i]
                 i = i + 1
            else
                A[k] = R[j]
                j = j + 1
```



`HeapSort——堆排序`

堆排序的要求：

> 1.根节点是A[1]
> 2.完全二叉树

`MAX-HEAPIFY(O(lgn))`

```
MAX_HEAPIFY(A,i)
    l = LEFT(i)
    r = RIGHT(i)
    if l<=heap-size[A] and A[l]>A[i]
        then largest = l
        else largest = i
    if r<=heap-size[A] and A[r]>A[largest]
        then largest = r
    if largest!=i
        then EXCHANGE A[i] WITH A[largest]
             MAX-HEAPIFY(A, largest)
```

`BUILD-MAX-HEAP(Θ(n)）`

```
BUILD-MAX-HEAP(A)
    heap-size[A] = length(A)
    for i=length(A)/2 downto 1
        do MAX-HEAPIFY(A,i)
```

`HeapSort`

```
HEAPSORT(A)
    BUILD-MAX-HEAP(A)
    for i = length(A) downto 2
        do EXCHANGE A[1] WITH A[i]
           heap-size(A) = heap-size(A) - 1
```

```
            MAX-HEAPIFY(A,1)
```

Quick Sort——传说中的神器

```
QUICKSORT(A, p, r)
    if p < r
        then q = PARTITION(A, p, r)
        QUICKSORT(A, p, q-1)
        QUICKSORT(A, q+1, r)
        #初始调用 QUICKSORT(A, 1, n)
```

```
PARTITION(A, p, r)
    x = A[r]
    i = p - 1
    for j=p to r-1
        do if A[j] <= x
            then i = i + 1
                 EXCHANGE A[i] WITH A[j]
    EXCHANGE A[i+1] WITH A[r]
    return i+1
```

有一个随机版本，区别在于，在选取r的位置时，不是确定位置，而是设置成为随机数

```
Random-PARTITION(A,p,r)
    i = RANDOM(p,r)
    EXCHANGE A[I] WITH A[R]
    return PARTITION(A, p, r)
```

Counting Sort——计数排序

根据已知的数组，计算出每个数有多少个比这个数小的数，建立映射

```
COUNTING-SORT(A, B, k)
    for i=1 to k
```

```
        do C[i] = 0
    for j=1 to length(A)
        do C[A[j]] = C[A[j]] + 1
    for i=2 to k
        C[i] = C[i] + C[i-1]
    for j = length(A) downto 1
        do B[C[A[j]]] = A[j]
           C[A[j]] = C[A[j]] - 1
```

`Radix Sort——基数排序`

传说中的一个数字一个数字的排序

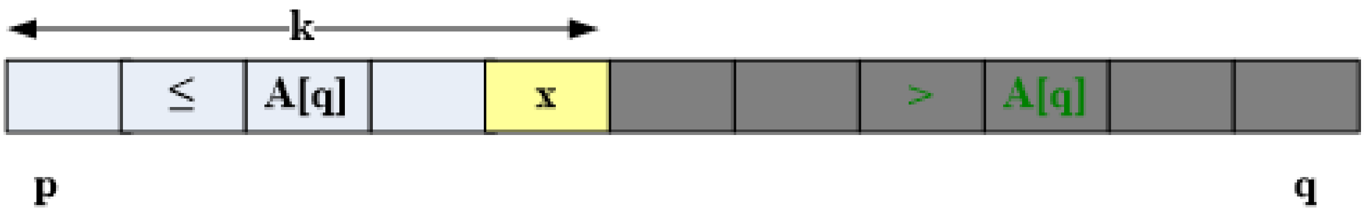
`Bucket Sort——桶排序（酒桶的桶）`

链表机制~首先分成大范围，在小范围使用插入排序

中位数

`RANDOMIZED-SELECT`

算法要求：找到第i小的数
实现：同QuickSort，只不过根据根据要找的位置和当前元素的位置觉得递归的下一层



if `i<k`, 左递归
if `i>k`, 右递归
if `i=k`, Exactly！！

```
RANDOMIZED-SELECT(A, p, r, i)
    if p = r
        then return A[p]
    q = RANDOMIZED-PARTITION(A, p, r)
```

```
    k = q - p + 1
    if i = k
        then return A[q]
    elseif i<k
        then return RANDOMIZED-SELECT(A, p, q-1, i)
    else
        return RANDOMIZED-SELECT(A, q+1, r, i- k)
```