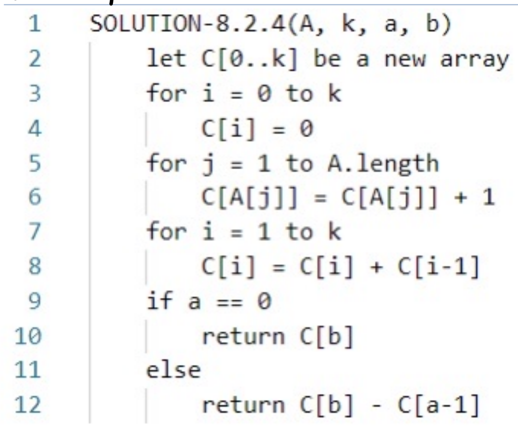
8.2.4

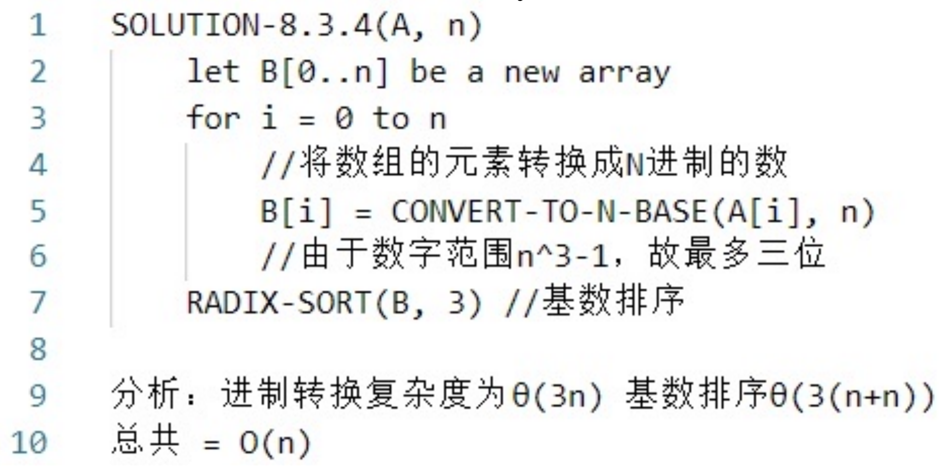
郑源泽 19307130077



```
SOLUTION-8.2.4(A, k, a, b)
    let C[0..k] be a new array
    for i = 0 to k
        C[i] = 0
    for j = 1 to A.length
        C[A[j]] = C[A[j]] + 1
    for i = 1 to k
        C[i] = C[i] + C[i-1]
    if a == 0
        return C[b]
    else
        return C[b] - C[a-1]
```

8.3.4

Show how to sort n integers in the range 0 to n^3-1 in $O(n)$ time

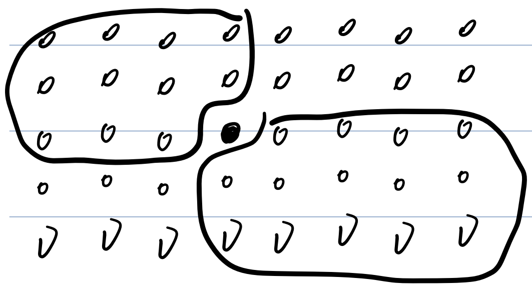


```
SOLUTION-8.3.4(A, n)
    let B[0..n] be a new array
    for i = 0 to n
        //将数组的元素转换成N进制的数
        B[i] = CONVERT-TO-N-BASE(A[i], n)
        //由于数字范围n^3-1，故最多三位
    RADIX-SORT(B, 3) //基数排序

分析：进制转换复杂度为θ(3n) 基数排序θ(3(n+n))
总共 = O(n)
```

9.3.2

analyze SELECT to show that if $n \geq 140$, then at least $\lceil n/4 \rceil$ elements are greater than the median-of-medians x and at least $\lceil n/4 \rceil$ elements are less than x



由

$$\frac{3n-60}{10} \geq \lceil \frac{n}{4} \rceil$$

$$\frac{3n-60}{10} \geq \frac{n}{4}+1$$

$$n \geq 140$$

9.1 前i个最大元素

a. 利用快排，从尾取i个元素

```
let B[0..i] be a new array
QUICK-SORT(A)
for j = n to n-i+1
    B[n-j] = A[j]
```

$\boxed{O(n\lg n) + O(i) = \Theta(n\lg n)}$

b.

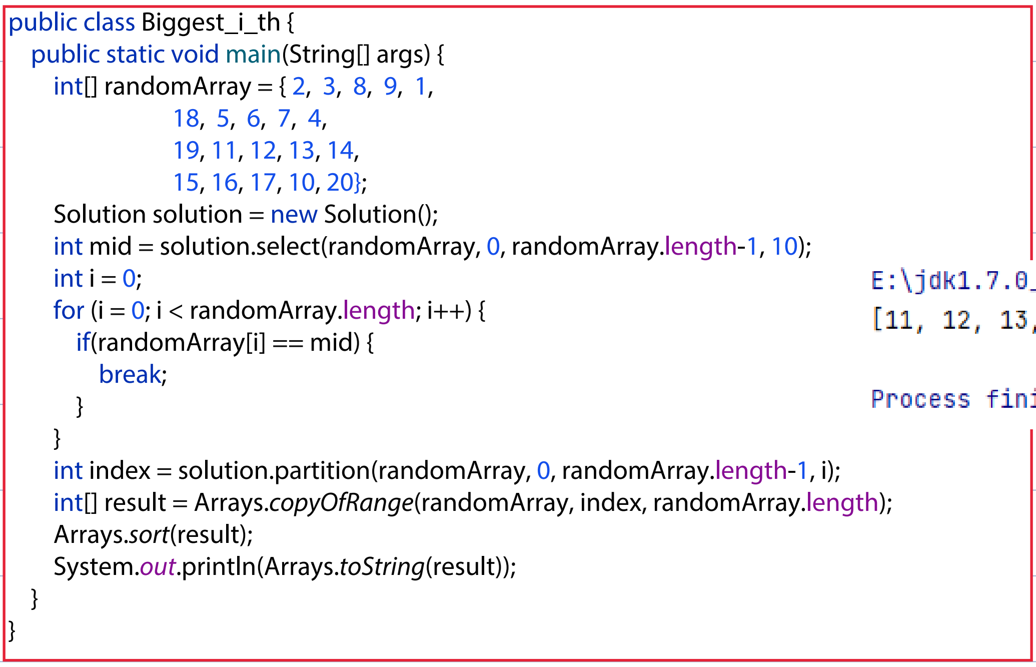
```
BUILD-MAX-HEAP(A)    //O(n)
let B[0..i] be a new array
for j=0 to i-1
    B[j] = HEAP-EXTRAT-MAX(A)    //O(lg n)
```

$\boxed{O(n) + O(i\lg n) = \Theta(n)}$

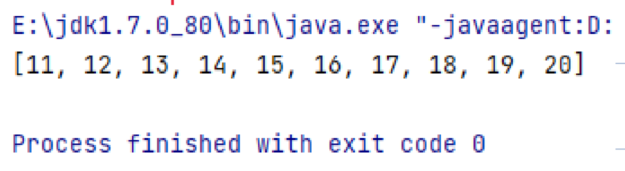
c. 找到第i大的分界 $O(n)$

比较排序 $O(i\lg i)$ $>$ $\boxed{O(n) + O(i\lg i) = \Theta(n)}$

(i为常数级情况下)



```java
public class Biggest_i_th {
    public static void main(String[] args) {
        int[] randomArray = { 2, 3, 8, 9, 1,
                    18, 5, 6, 7, 4,
                    19, 11, 12, 13, 14,
                    15, 16, 17, 10, 20};
        Solution solution = new Solution();
        int mid = solution.select(randomArray, 0, randomArray.length-1, 10);
        int i = 0;
        for (i = 0; i < randomArray.length; i++) {
            if(randomArray[i] == mid) {
                break;
            }
        }
        int index = solution.partition(randomArray, 0, randomArray.length-1, i);
        int[] result = Arrays.copyOfRange(randomArray, index, randomArray.length);
        Arrays.sort(result);
        System.out.println(Arrays.toString(result));
    }
}
```



```
E:\jdk1.7.0_80\bin\java.exe "-javaagent:D:
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

Process finished with exit code 0
```

```
class Solution{
   public int select(int [] arr, int start, int end, int i){
      int groupSize = 5;
      int size = end-start+1;
      int[] mids = size%groupSize == 0 ? new int[size/groupSize] : new int[size/groupSize+1];
      for (int j = 0; j < mids.length; j++) {
         mids[j] = arr[find_mid_number(arr, start+j*groupSize, Math.min(start + (j + 1) * groupSize - 1, end))];
      }
      int mid = find_mid_number(mids, 0, mids.length-1);
      int mid_index = 0;
      for(int k = 0; k < arr.length; k++){
         if(arr[k] == mids[mid]){
            mid_index = k;
            break;
         }
      }
      int index = partition(arr, start, end, mid_index);
      if (i == end-index+1) {
         return arr[index];
      }
      else if (i < end-index+1) {
         return select(arr, index+1, end, i);
      }
      else {
         return select(arr, start, index-1, i-(end-index+1));
      }
   }
   public int partition(int [] arr, int start, int end, int pivot){
      swap(arr, pivot, end);
      int x = arr[end];
      int i = start - 1;
      for (int j = start; j < end; j++) {
         if (arr[j] <= x) {
            i++;
            swap(arr, i, j);
         }
      }
      swap(arr, i+1, end);
      return i+1;
   }
   public int find_mid_number(int [] arr, int start, int end){
      Arrays.sort(arr, start, end+1);
      return (start+end)/2;
   }
   public void swap(int [] arr, int i, int j){
      int temp = arr[i];
      arr[i] = arr[j];
      arr[j] = temp;
   }
}
```