第 5 章　分治法

1. 利用分治法求一组数据中最大的两个数。

C/C++代码：

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>// 包含 INT_MIN 的定义

//分治法求最大的两个数
void findMaxTwo(int arr[], int left, int right, int result[]) {
    // 如果只有一个元素
    if (left == right) {
        result[0] = arr[left];
        result[1] = INT_MIN; // 使用 INT_MIN 表示最小整数值
        return;
    }
    // 如果有两个元素
    if (right - left == 1) {
        if (arr[left] > arr[right]) {
            result[0] = arr[left];
            result[1] = arr[right];
        }else {
            result[0] = arr[right];
            result[1] = arr[left];
        }
        return;
    }
    // 分治法: 将数组分成两部分
    int mid = (left + right) /2;
    int leftResult[2], rightResult[2];
    // 递归处理左半部分
    findMaxTwo(arr, left, mid, leftResult);
    // 递归处理右半部分
    findMaxTwo(arr, mid +1, right, rightResult);
    // 合并结果
    if (leftResult[0] > rightResult[0]) {
        result[0] = leftResult[0];
```

```
            result[1] = (leftResult[1] > rightResult[0]) ? leftResult[1] :
rightResult[0];
        }else {
            result[0] = rightResult[0];
            result[1] = (rightResult[1] > leftResult[0]) ? rightResult[1] :
leftResult[0];
        }
    }
    int main() {
        int n;
        scanf("% d", &n);
        if (n <= 0) {
            printf("数组大小必须大于 0! \n");
            return 1;
        }
        int * arr = (int * )malloc(n *  sizeof(int)); // 动态分配数组内存
        if (arr == NULL) {
            printf("内存分配失败! \n");
            return 1;
        }
        for (int i = 0; i < n; i++) {
            scanf("% d", &arr[i]);
        }
        int result[2];
        findMaxTwo(arr, 0, n - 1, result);
        printf("% d % d\n", result[0], result[1]);
        free(arr);
        return 0;
    }
```

Python 代码：

```
import sys

#分治法求最大的两个数
def find_max_two(arr, left, right):
    # 如果只有一个元素
    if left == right:
        return arr[left], -sys.maxsize  # 使用 sys.maxsize 表示最小整数值

    # 如果有两个元素
```

```
    if right - left == 1:
        if arr[left] > arr[right]:
            return arr[left], arr[right]
        else:
            return arr[right], arr[left]

    # 分治法: 将数组分成两部分
    mid = (left + right) //2
    left_max, left_second = find_max_two(arr, left, mid)
    right_max, right_second = find_max_two(arr, mid +1, right)

    # 合并结果
    if left_max > right_max:
        return left_max, max(left_second, right_max)
    else:
        return right_max, max(right_second, left_max)

#主程序
if __name__ == "__main__":
    n =int(input())  # 输入数组大小
    if n <= 0:
        print("数组大小必须大于0!")
    else:
        arr =list(map(int, input().split()))  # 输入数组元素
        max1, max2 = find_max_two(arr, 0, n - 1)
        print(max1, max2)
```

2. 利用分治法求一组数据的和。

C/C++代码：

```
#include <stdio.h>
#include <stdlib.h>

//分治法求和
int findSum(int arr[], int left, int right) {
    // 如果只有一个元素
    if (left == right) {
        return arr[left];
    }
    // 分治法: 将数组分成两部分
    int mid = (left + right) /2;
```

```
    int leftSum = findSum(arr, left, mid); // 左半部分的和
    int rightSum = findSum(arr, mid +1, right); // 右半部分的和
    // 合并结果
    return leftSum + rightSum;
}

int main() {
    int n;
    scanf("% d", &n);    // 输入数组大小
    if (n <= 0) {
        printf("数组大小必须大于 0! \n");
        return 1;
    }
    int * arr = (int * )malloc(n *  sizeof(int)); // 动态分配数组内存
    if (arr == NULL) {
        printf("内存分配失败! \n");
        return 1;
    }
    for (int i = 0; i < n; i++) {
        scanf("% d", &arr[i]);
    }
    int sum = findSum(arr, 0, n - 1);    // 使用分治法求和
    printf("% d\n", sum);                 // 输出结果
    free(arr); // 释放内存
    return 0;
}
```

Python 代码：

```
#分治法求和
def find_sum(arr, left, right):
    # 如果只有一个元素
    if left == right:
        return arr[left]
    # 分治法: 将数组分成两部分
    mid = (left + right) //2
    left_sum = find_sum(arr, left, mid)# 左半部分的和
    right_sum = find_sum(arr, mid +1, right)  # 右半部分的和
    # 合并结果
    return left_sum + right_sum
```

```
def main():
    # 输入数组大小
    n =int(input())
    if n <= 0:
        print("数组大小必须大于 0!")
        return
    # 输入数组元素(一行以空格分隔)
    arr =list(map(int, input().split()))
    # 检查输入的元素个数是否与数组大小一致
    if len(arr) ! = n:
        print("输入的元素个数与数组大小不匹配!")
        return
    # 使用分治法求和
    total_sum = find_sum(arr, 0, n - 1)
    print(total_sum)

if __name__ == "__main__":
    main()
```

3. 对于给定的 n 个元素的数组 $a[0:n-1]$，要求从中找出第 k 小的元素。

C/C++代码：

```
#include <stdio.h>
//快速选择排序
int cul(int arr[], int l, int r, int k){
    if (l == r)
        return arr[l];
    int target = arr[(l + r) /2], left = l - 1, right = r + 1;
    while (left < right){
        do{
            left++;
        }while (arr[left] < target);
        do{
            right--;
        }while (arr[right] > target);
        if (left < right){
            int temp = arr[left];
            arr[left] = arr[right];
            arr[right] = temp;
        }
    }
```

```
        // right == left, right 左边的数都是小于某个基准值的, right 右边的数都是大于某个基准值的
        if (k <= right)
            cul(arr, l, right, k);
        else
            cul(arr, right +1, r, k);
    }

    int main(){
        int n, k;
        scanf("% d % d", &n, &k);
        int arr[n +1];
        for (int i = 1; i <= n; i++)
            scanf("% d", &(arr[i]));
        printf("第% d 小的元素是% d \n", k, cul(arr, 0, n - 1, k));
        return 0;
    }
```

Python 代码：

```
def quick_select(arr, l, r, k):
    if l == r:
        return arr[l]

    # 选择基准值
    target = arr[(l + r) //2]
    left = l -1
    right = r +1

    while left < right:
        while True:
            left +=1
            if arr[left] >= target:
                break
        while True:
            right -=1
            if arr[right] <= target:
                break
        if left < right:
            # 交换元素
            arr[left], arr[right] = arr[right], arr[left]
```

```
    # 根据 k 的位置决定递归范围
    if k <= right:
        return quick_select(arr, l, right, k)
    else:
        return quick_select(arr, right + 1, r, k)

def main():
    # 输入 n 和 k
    n, k =map(int, input().split())
    # 输入数组元素
    arr =list(map(int, input().split()))
    # 确保 k 在有效范围内
    if k < 1 or k > n:
        print("k 的值无效!")
        return
    # 调用快速选择算法
    result = quick_select(arr, 0, n - 1, k - 1)  # Python 的索引从 0 开始

    # 输出结果
    print(f"第{k}小的元素是{result}")

if __name__ == "__main__":
    main()
```