2019-10-20

基础算法

分治法

一次或多次递归的调用自身以解决紧密相关的若干子问题。

分治法的思想

将原来的问题分解成几个规模较小但类似于原问题的子问题，递归地求解这些子问题，然后再合并这些子问题的解来建立原问题的解。

分治法的步骤

分治法在每层递归时都有三个步骤：

1. **分解**原问题为若干个子问题，这些子问题是原问题的规模较小的实例。
2. **解决**这些子问题，递归地求解各子问题。然而，若子问题的规模足够小，则直接求解。
3. **合并**这些子问题的解成原问题的解。

分治法的复杂性分析

假设T(n)是规模为n的一个问题的运行时间。如果问题规模足够小，假设小于某个常量c，则直接求解需要常量时间，我们将其写作

$$\Theta(1)$$

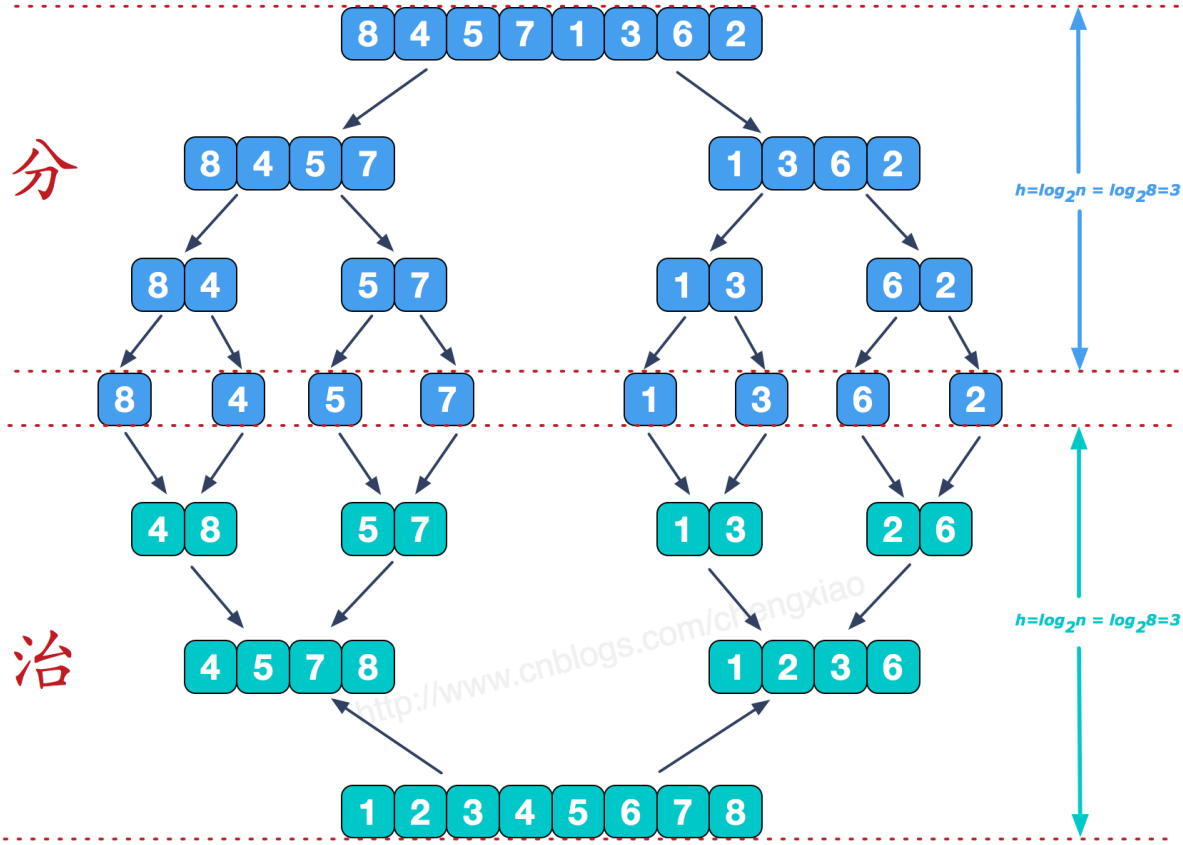
。假设原问题分解成a个子问题，每个子问题的规模是原问题的1/b （对于归并排序，a和b都是2，然而很多分治算法中，a != b)。为了求解一个规模为n/b的子问题，需要T(n/b)的时间，所以需要aT(n/b)的时间来求解a个子问题。如果分解问题成子问题需要时间D(n)，合并子问题的解成原问题的解需要时间C(n)，那么得到递归式：

$$T(n) = \begin{cases} \Theta(1) & \text{，若 } n \leq c \\ aT(\frac{n}{b}) + D(n) + C(n) & \text{，其他} \end{cases}$$

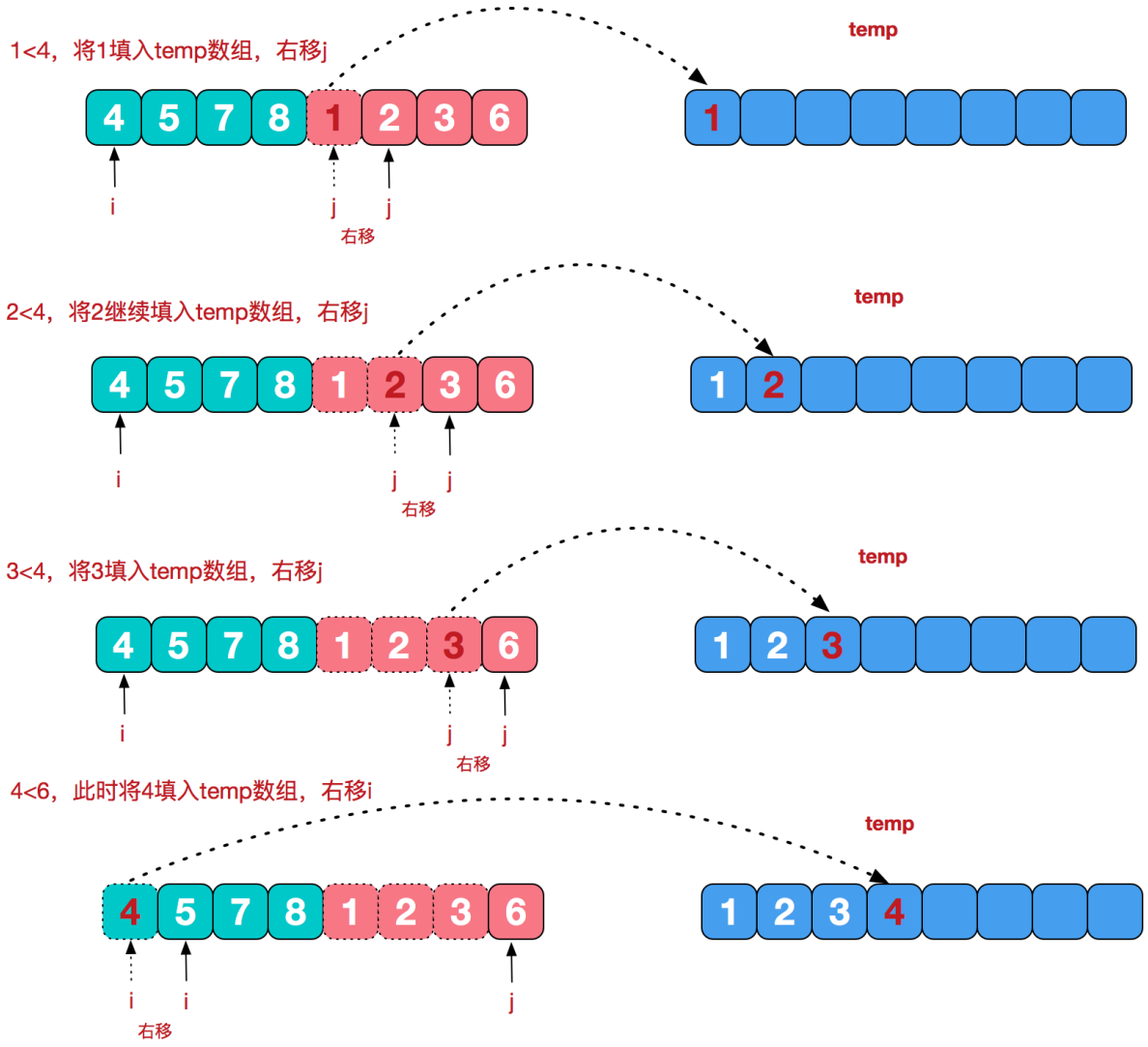
例子分析

归并排序算法：

1. 分解：分解待排序的n个元素的序列成各具n/2个元素的两个子序列。此步骤仅仅计算中间位置，需要常量时间。
2. 解决：使用归并排序递归地排序两个子序列。
3. 合并：合并两个已排序的子序列以产生已排序的答案。这个是归并排序的关键。


8 4 5 7 1 3 6 2
分
8 4 5 7
1 3 6 2
8 4
5 7
1 3
6 2
8
4
5
7
1
3
6
2
4 8
5 7
1 3
2 6
治
4 5 7 8
1 2 3 6
1 2 3 4 5 6 7 8
h=log2n = log28=3
h=log2n = log28=3

合并算法：最多需要n-1次比较，所以需要O(n)的时间。



归并排序的递归式，其中常量c代表求解规模为1的问题所需要的时间以及在分解步骤于合并步骤处理每个数组元素所需的时间：

$$T(n) = \begin{cases} c\text{，若 } n = 1 \\ 2T(\frac{n}{2}) + cn\text{，若 } n > 1 \end{cases}$$

我们可以推导出时间复杂度：

```
T(n) = 2*T(n/2) + cn
     = 2*(2*T(n/4) + cn/2) + cn = 4*T(n/4) + 2*cn
     = 4*(2*T(n/8) + cn/4) + 2*cn = 8*T(n/8) + 3*cn
     = 8*(2*T(n/16) + cn/8) + 3*cn = 16*T(n/16) + 4*cn
     ......
     = 2^k * T(n/2^k) + k * cn
     ......
```

当 T(n/2^k)=T(1) 时，也就是 n/2^k=1，我们得到 k=lgn, 将 k 值代入上面的公式，得到 T(n)=cn+cnlgn, 用大 O 标记法来表示的话:

$$O(n \lg n)$$

```
#include <iostream>
using namespace std;

// 合并数组，排好序，然后在拷贝到原来的数组array
void MergeArray(int array[], int start, int end ,int mid, int temp[]) {
    int i = start;
    int j =  mid + 1;
    int k = 0;
    while (i <= mid && j <= end ) {
        if (array[i] < array[j]) {
            temp[k++] = array[i++];
        }else {
            temp[k++] = array[j++];
        }
    }
    while (i <= mid) {
        temp[k++] = array[i++];
    }
    while (j <= end) {
        temp[k++] = array[j++];
    }
    for (int i = 0; i < k; i ++) {
        array[start + i] = temp[i];
    }

}
// 归并排序，将数组前半部分后半部分分成最小单元，然后在合并
void MergeSort(int array[], int start,  int end, int temp[]) {
    if(start < end) {
        int mid = (start + end)/ 2;
        MergeSort(array, start, mid, temp);
        MergeSort(array, mid + 1, end, temp);
        MergeArray(array, start, end, mid, temp);
    }

}
// 在这里创建临时数组，节省内存开销，因为以后的temp都是在递归里使用的。
void MergeSort(int array[], int len) {
    int start = 0;
    int end = len - 1;
    int *temp = new int[len];
    MergeSort(array, start, end, temp);
}
```

```
void PrintArray(int array[], int len) {
    for (int i = 0 ; i < len; ++i) {
        cout << array[i] << " " ;

    }
    cout << endl;
}

int main(int argc, const char * argv[]) {
    int array[] = {3,5,3,6,7,3,7,8,1};

    MergeSort(array, 9);
    PrintArray(array, 9);


    return 0;
}
```

二分查找

二分查找也是分治法的一个应用，它的前提是序列必须是有序的，而且是存储在数组中（为什么）。

```
int BinarySearch_Recursive(int array[], int low, int high, int value)
{
  if (low > high)
    return -1;
  int mid = low + (high - low) / 2;
  if (array[mid] == value)
    return mid;
  else if (array[mid] > value)
    return BinarySearch_Recursive(array, low, mid - 1, value);
  else
    return BinarySearch_Recursive(array, mid + 1, high, value);

}
```