第3章 分治法

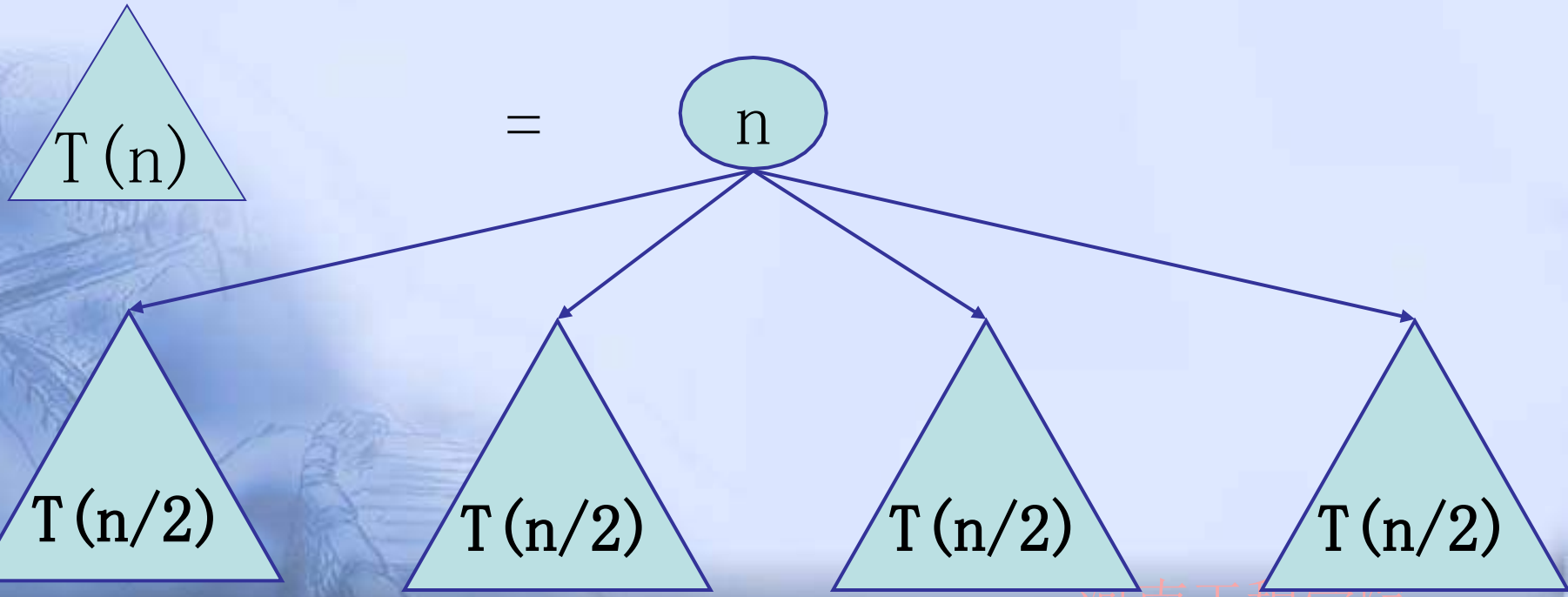
- 分治法的思想将一个难以直接解决的问题分解成容易求解的子问题，以便各个击破、分而治之。

3.1 一般算法

- 分治法的求解步骤
- 1 分解
- 2 解决
- 3 合并

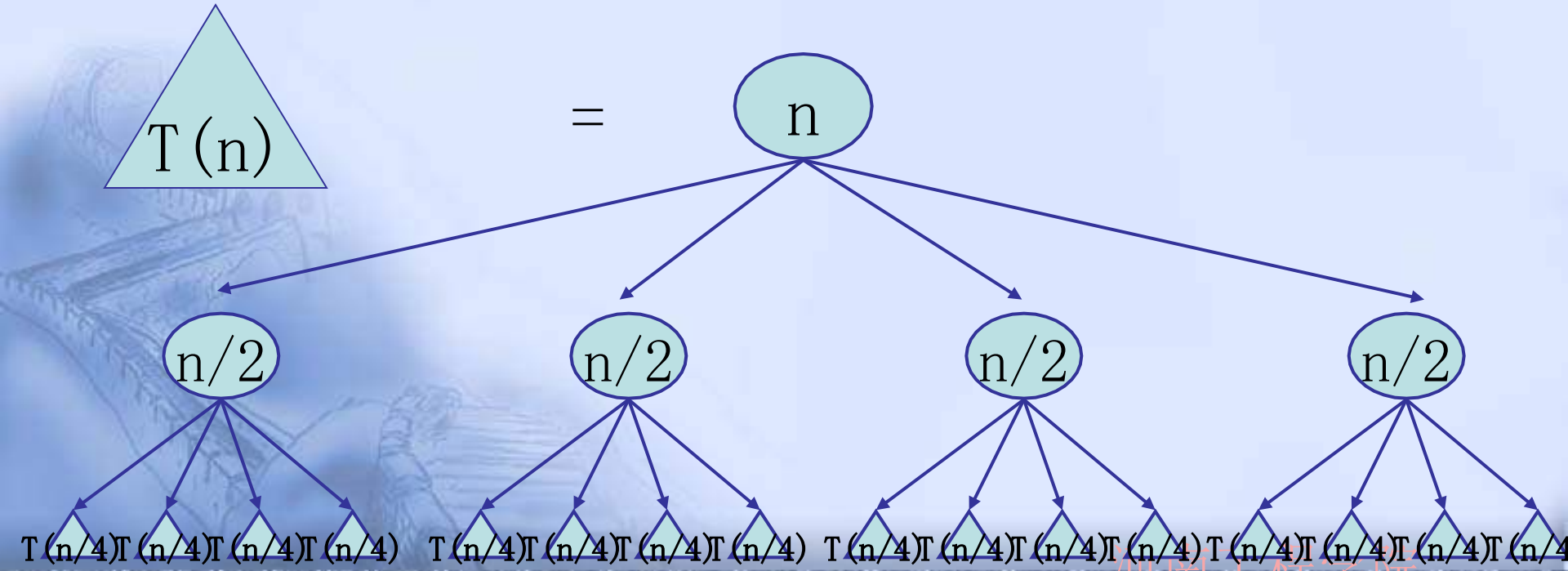
算法总体思想

- 对这k个子问题分别求解。如果子问题的规模仍然不够小，那么再划分为k个子问题，如此递归的进行下去，直到问题规模足够小，很容易求出其解为止。



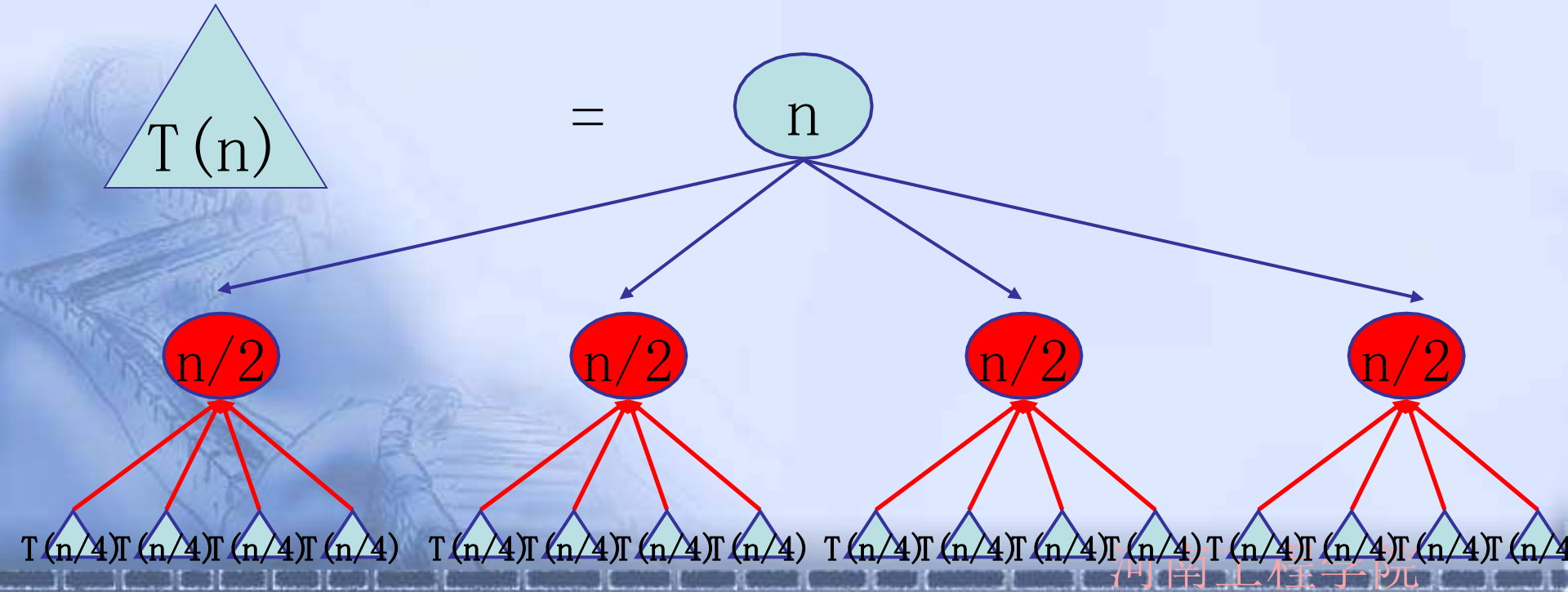
算法总体思想

- 将求出的小规模的问题的解合并为一个更大规模的问题的解，自底向上逐步求出原来问题的解。



算法总体思想

- 将求出的小规模的问题的解合并为一个更大规模的问题的解，自底向上逐步求出原来问题的解。



算法总体思想

- 将求出的小规模的问题的解合并为一个更大规模的问题的解，自底向上逐步求出原来问题的解。

分治法的设计思想是，将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之。

分治法的根本步骤

```
divide-and-conquer(P)
  {
1    if ( n <= n0) //解决小规模的问题
2          then
3              {
4                 解决子问题
5                   return(子问题的解)
6  }
7   for (i=2 to k)//分解问题
8    do   yi←Divided-and-Conquerer(|Pi|)
9   T←MERGE(y1,y2,…,yk);
10  return
```

分治法的适用条件

分治法所能解决的问题一般具有以下几个特征：

该问题的规模缩小到一定的程度就可以容易地解决；

该问题可以分解为假设干个规模较小的相同问题，即该问题具有最优子结构性质

利用该问题分解出的子问题的解可以合并为该问题的解；

该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题。

这条特征涉及到分治法的效率，如果各子问题是不独立的，那么分治法要做许多不必要的工作，重复地解公共的子问题，此时虽然也可用分治法，但一般用动态规划较好。

3.2 二分检索

给定已按升序排好序的n个元素a[1:n]，现要在这n个元素中找出一特定元素x。

分析:该问题的规模缩小到一定的程度就可以容易地解决;

该问题可以分解为假设干个规模较小的相同问题;

分解出的子问题的解可以合并为原问题的解;

分解出的各个子问题是相互独立的。

分析：很显然此问题分解出的子问题相互独立，即在a[i]的前面或后面查找x是独立的子问题，因此满足分治法的第四个适用条件。

给定已按升序排好序的n个元素a[1:n]，现要在这n个元素中找出一特定元素x。

```
BINSEARCH(A,n,x)
1  low←1
2  high←n
3  while low<=high
4      do
5      {
6        mid←(low+high)/2;
7         if x=A[mid];
8            then return mid;
9         else if x<A[mid]
10             then high←mid-1
11             else low←mid+1
12}
13return 0;
```

算法复杂度分析：

每执行一次算法的while循环， 待搜索数组的大小减少一半。因此，在最坏情况下，while循环被执行了O(logn) 次。循环体内运算需要O(1) 时间，因此整个算法在最坏情况下的计算时间复杂性为O(logn)

3.3 找最大值和最小值

- 1 将数据等分为两组，目的是分别选取其中的最大和最小值
- 2 递归分解直到每组元素的个数<=2， 可简单找到最大和最小值
- 3 回溯时合并子问题的解

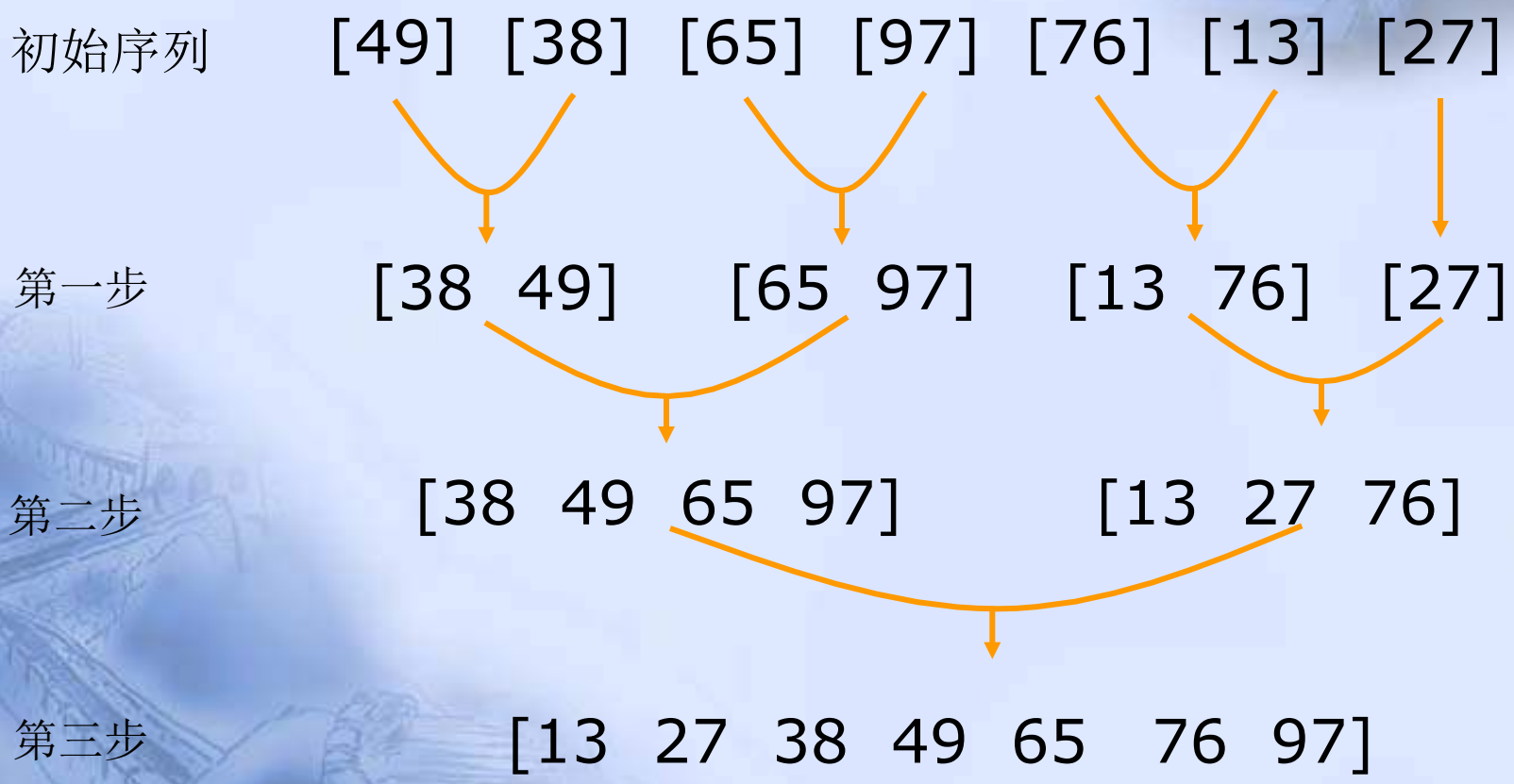
```
float A[n]
maxmin(i,j,fmax,fmin)
1  if i=j
2  then
3  {        fmax←A[i]
5        fmin←A[i]   }
7   else if i=j-1
8         then if  A[i]<A[j]
9                    then
11                        { fmax←A[j]
12                          fmin←Api]}
14                   else
15
21                   { mid←(i+j)/2
22                   maxmin(i, mid, lmax, lmin)
23                 maxmin(mid+1, j, rmax,rmin)
24
```

3.4 归并分类

3.4.1 根本方法

根本思想：将待排序元素分成大小大致相同的2个子集合，分别对2个子集合进行排序，最终将排好序的子集合合并成为所要求的排好序的集合。

*


初始序列
[49] [38] [65] [97] [76] [13] [27]
第一步
[38 49] [65 97] [13 76] [27]
第二步
[38 49 65 97] [13 27 76]
第三步
[13 27 38 49 65 76 97]

```
MERGESORT(A, low, high)
1  if low<high
2    then
3   {
4      mid←(low+high)/2
5      MERGESORT(A, low, mid)
6      MERGESORT(A, mid+1, high);
7      MERGE(A, low, mid, high)
8  }
```

- MERGE(A, low, mid, high)
- 1 h←low
- 2 i←low
- 3 j←mid+1
- 4 While h<=mid and j<=high
- 5 do {
- 7 if A[h]<=A[j]
- 8 then {
- 10 B[i]←A[h] ; h←h+1}
- 13 else {
- 14 B[i]←A[j]; j←j+1}

- 20 if h>mid
- 21 then {
- 23 for k←j to high
- 24 do
- 25 { B[i]←A[k] i←i+1}}
- 30 else {
- 32 for k←h to mid
- 33 do
- 34 { B[i]←A[k]; i←i+1}}
- 39 for k←low to high
- 40 do A[k]←B[k]

合并排序

- 最坏时间复杂度：O(nlogn)
- 平均时间复杂度：O(nlogn)
- 辅助空间：O(n)

3.4.2 改进的归并分类

- MERGESORT(A, low, high, p)
- 1 if high-low+1<16
- 2　then INSERTIONSORT(A, LINK, low,high, p)
- 3 else {
- 5　　mid←(low+high)/2
- 6　　MERGESORT1(low, mid, q)
- 7　　MERGESORT1(mid+1, high, r);
- 8　　MERGE1(q,r p);
- 9 }

- MERGE1(A, q, r,p)
- 1 i←q; j←r; k←0;
- 4 while i<>0 and j<>0
- 5 do {
- 7 if A[i]<=A[j]
- 8 then
- 9 { LINK[k]←i; k←i; i←LINK[i] }
- 14 else
- 15 {LINK[k]←j; k←j; j←LINK[j]}}
- 21 if (i=0) then LINK[k]←j;
- 23 else LINK[[k]←i;
- 25 p←LINK[0]

3.5 快速分类

- 3.5.1 快速分类算法
- 根本思想：选取A的某个元素， 如t=A[i]，然后将其他元素重新排序， 使A[1,…,n]中在t以前出现的元素都小于或等于t， 而所有在t后面出现的元素都大于或等于t

- Partition(A,m,p)
- 1 v←[m]
- 2 while true
- 3 do {
- 5 i←m
- 6 while A[i]<=v do i←i+1;
- 7 while A[p]>=v do p←p-1;
- 8 if i<p then swap(A[i], A[p]
- 9 else break }
- 12 A[m]←A[p]
- 13 A[p]←v

- QUICKSORT(A,p,q)
- 1 if p<q
- 2 then {
- 4 j←q+1;
- 5 Partion (p,j);
- 6 QUICKSORT(p, j-1);
- 7 QUICKSORT(j+1, q);

- 3.5.2 快速分类的分析