1.快速排序（平均时间复杂度O($n\log_2 n$)）

快速排序是通过多次比较和交换来实现排序，在一趟排序中把将要排序的数据分成两个独立的部分，对这两部分进行排序使得其中一部分所有数据比另一部分都要小，然后继续递归排序这两部分，最终实现所有数据有序。

主要思想：分治

方法：

（1）确定分界点（左边界，中间值，右边界，随机数）

（2）将区间根据分界点分为两段：左侧的数都<=x，右侧的数都>=x（常考）

（3）递归处理左右两端

暴力解法：建立两个数组a和b，扫描整段数字，将小于分界点的放入数组a，大于分界点的放入数组b

优美解法：

在左侧和右侧分别用两个指针i和j，指针i向右移动直至遇到大于等于分界点的数字，指针j向左移动直至遇到小于等于分界点的数字，此时将i和j指针所指的数字进行交换，如此进行下去直到i和j指针指向同一数字。

当输入数据较多时，建议用scanf来读取数据，不要用cin

```
void quick_sort(int q[], int i, int j)
{
    if(i >= j) {
        return;
    }
    int temp = q[i + j >> 1]; // 确定分界点
    int s = i - 1, t = j + 1;
    while(s < t){
        do s++; while(q[s] < temp);
        do t--; while(q[t] > temp);
        if(s < t)
            swap(q[s], q[t]);
    }
    quick_sort(q, i, t);
    quick_sort(q, t + 1, j);
}
```

2.快速选择（时间复杂度O(n)）

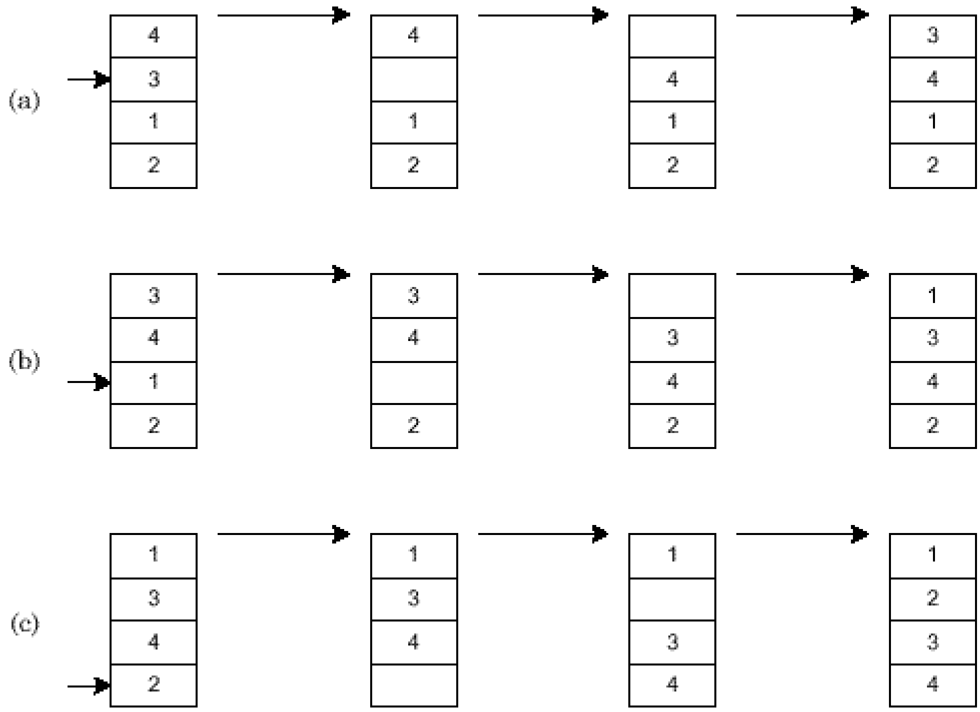
数字 k 小于左侧left值时，对左侧进行快排，否则对右侧进行快排

3.插入排序（平均时间复杂度为O(n^2)，空间复杂度为O(1)）

插入排序中，当待排序数组是有序时，是最优的情况，只需当前数跟前一个数比较一下就可以了，这时一共需要比较 N-1 次，时间复杂度为 O(N)。最坏的情况是待排序数组是逆序的，此时需要比较次数最多，最坏的情况是 O(n^2)。

最简单的排序方法

对于少量元素的排序，插入排序是一种有效的算法

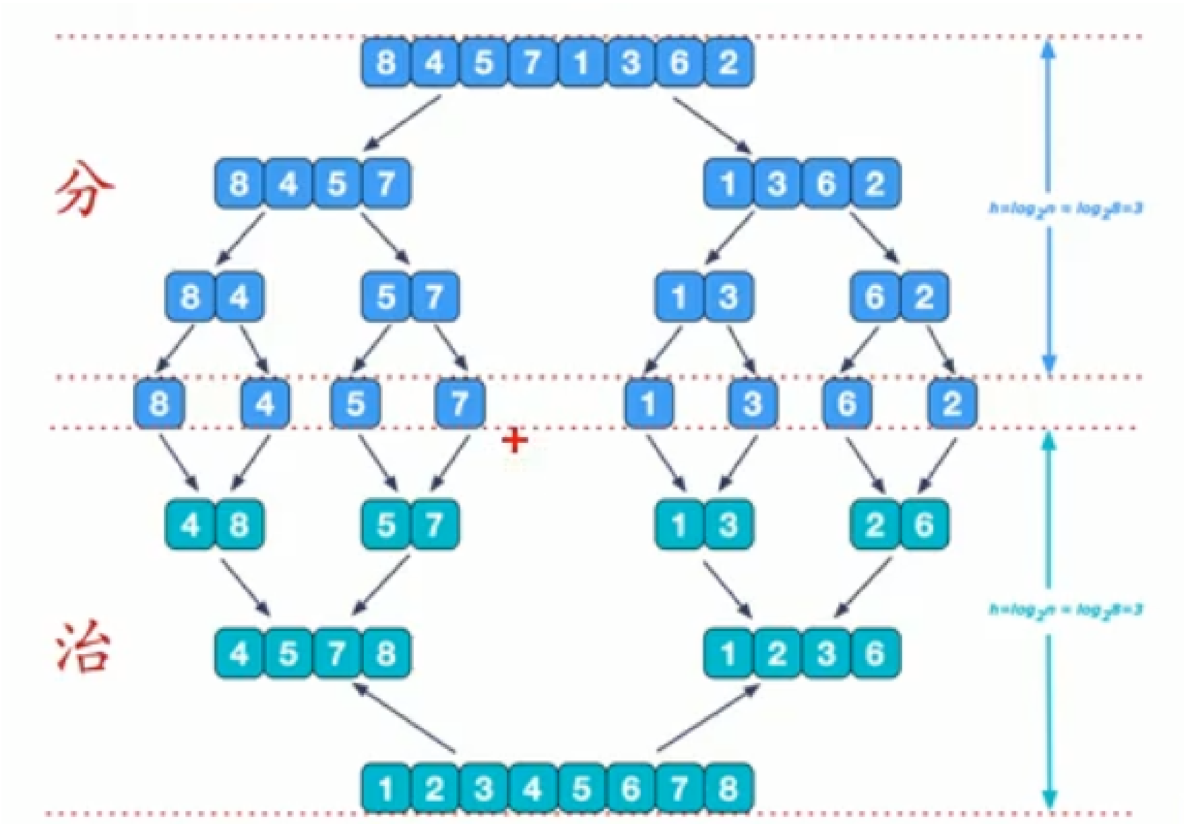


4.归并排序——分治（时间复杂度O($n\log_2 n$)）

（1）找分界点：取中间值

（2）递归排序left right

（3）归并——合二为一 时间复杂度为$n\log_2 n$



排序的稳定性：一个序列中若有两个元素的值相同，排序之后，这两个元素的相对位置不变，那么就称这个排序算法是稳定的。

```
void merge_sort(int q[], int l , int r)
{
    if(l >= r) // 区间中的元素个数为1或者0
```

```
        return;
    int mid = l + r >> 1; // 寻找分界点
    merge_sort(q, l, mid);
    merge_sort(q, mid + 1, r);
    int k = 0, i = l, j = mid + 1;
    while(i <= mid && j <= r) // 进行归并
    {
        if(q[i] <= q[j])
            tmp[k++] = q[i++];
        else
            tmp[k++] = q[j++];
    }
    while(i <= mid) // 若有剩下的元素
    {
        tmp[k++] = q[i++];
    }
    while(j <= r)
    {
        tmp[k++] = q[j++];
    }
    for(i = l, j = 0; i <= r; i++, j++) // 将tmp数组中的值赋给q数组
    {
        q[i] = tmp[j];
    }
}
```

5.二分法

二分的本质并不是单调性，本质是边界

（1）若要找的性质为绿色区的，判断mid是否满足性质

找分界点 mid=(left + right + 1) / 2

a. mid满足性质 则分界点的范围为[mid, right]，此时令left = mid即可

b. mid不满足性质 则分界点的范围为[left, mid - 1]，此时令right = mid - 1即可

若要找的性质为红色区的，判断mid是否满足性质

找分界点 mid=(left + right) / 2

a. mid满足性质 则分界点的范围为[left, mid]

b. mid不满足性质 则分界点的范围为[mid + 1, right]

（2）写一个check函数

6.高精度运算

（1）如何存储大整数

将其存储到数组中，不用int变量

将数字的个位存到数组的第0项，这样在相加进位的时候，数组可以自动增加一项

（2）加法

```
vector<int> add(vector<int> &A, vector<int> &B)
{
    vector<int> C;
    int j = 0; // 此处注意j的妙用
    for(int i = 0; i < A.size() || i < B.size(); i++)
    {
        if(i < A.size())
            j = j + A[i];
        if(i < B.size())
            j = j + B[i];
        C.push_back(j % 10);
        j = j / 10;
    }
    if(j)
        C.push_back(j);
    return C;
}
```

（3）减法

```
vector<int> sub(vector<int> &A, vector<int> &B)
{
    vector<int> C;
    int t = 0; // t为借位
    for(int i = 0; i < A.size(); i++)
    {
        t = t + A[i];
        if(i < B.size())
            t = t - B[i];
        C.push_back((t + 10) % 10);
        if(t < 0)
            t = -1;
        else
            t = 0;
    }
    while(C.size() > 1 && C.back() == 0)
        C.pop_back();   //去掉前导0
    return C;
}
```

（4）乘法

```
vector<int> mul(vector<int> &A, int B)
{
    vector<int> C;
    int temp = 0; // 进位
    for(int i = 0; i < A.size(); i++)
```

```
    {
        temp = A[i] * B + temp;
        C.push_back(temp % 10);
        temp = temp / 10;
    }
    while(temp)
    {
        C.push_back(temp % 10);
        temp = temp / 10;
    }
    while(C.size() > 1 && C.back() == 0)
        C.pop_back();  //去掉前导0
    return C;
}
```

（5）除法

除法的本质：A[i]除以B，将结果放入C[i]，余数乘以10加上下一位继续运算

```
vector<int> divide(vector<int> &A, int B)
{
    vector<int> C;
    int t = 0;
    for(int i = A.size() - 1; i >= 0; i--)
    {
        t = t * 10 + A[i];
        if(t < B) {
            C.push_back(0);
        } else {
            C.push_back(t / B);
            t = t % B;
        }
    }
    yushu = t;
    while(C.size() > 1 && C[0] == 0)
    {
        C.erase(C.begin());
    } // 去掉0
    return C;
}
```

7.前缀和

数组中$S_i = a_1 + a_2 + a_3 + \cdots + a_i$，定义 $S_0 = 0$.

用途：计算数组中从 l 到 r 的和即S_r - S_{l-1}

8.差分

数组a_1，a_2，$\cdots$，a_3构造数组b_1，b_2，$\cdots$，b_n使得$a_i = b_1 + \cdots + b_i$

$O_{(n)}$的复杂度，从 b 数组得到 a 数组

应用

数组a中[l, r]加上c，相当于让b_l 加上c，b_{r+1}减去c，即在O_n的时间内给数组的一段数据加上值

9.双指针算法

（1）指向两个序列

（2）指向一个序列

核心思想

```
for(int i = 0; i < n; i++)
{
    for(int j = 0; j < n; j++)
    {
        // O(n2)
    }
}
```

作用：优化了计算效率，将上面的朴素算法优化到O(n)

模板

```
for(i = 0, j = 0; i < n; i++)
{
    while(j < i && check(i, j))
        j++;
    // 每道题目的具体逻辑
}
```

10.位运算

n 的二进制表示中第 k 位是几

（1）先把第 k 位移到最后一位 n >> k

（2）看个位是几，x & 1（x与1进行与运算）

综上，方法为n >> k & 1

lowbit（x)：树状数组基本操作，返回 x 最后一位1，若 x 的二进制表示为1010，则lowbit(x)返回10

实现方式：x & $-x$（补码）= x & （~x + 1）

11.离散化

整数的离散化

将 a 数组1,3，100,2000，10^5映射为0，1,2,3,4

存在问题：（1）数组 a 中可能存在重复元素，去重

（2）如何算出 a_i 离散化后的值，二分

去重代码

```
vector<int> alls; // 存储所有待离散化的值
sort(alls.begin(), alls.end()); // 将所有值排序
alls.erase(unique(alls.begin(), alls.end()), alls.end()); // 去掉重复元素
```

二分求离散化后的值代码

```
int find(int x)
{
    int l = 0, r = alls.size() - 1;
    while(l < r)
    {
        int mid = l + r >> 1;
        if(alls[mid] >= x)
            r = mid;
        else
            l = mid - 1;
    }
    return r + 1; // 加一的话就是映射到1,2,3
}
```

当范围较小时，可以直接用前缀和来做

整个值域跨度很大，但用的数很少

12.区间合并

（1）按照区间左端点排序

（2）更新区间

关键点

```
void merge(vector<PII> &segs)
{
    vector<PII> res;
    sort(segs.begin(), segs.end());
    int begin = -2e9, end = -2e9;
    for(auto seg : segs)
    {
        if(end < seg.first)
        {
            if(begin != -2e9) // 防止输入的区间为空
                res.push_back({begin, end});
            begin = seg.first;
            end = seg.second; // 进行区间更新
        }
        else
            end = max(seg.second, end); // 进行区间更新
    }
    if(begin != -2e9)
    {
        res.push_back({begin, end});
    }
    segs = res;
}
```

13.链表

（1）数组模拟单链表（用的最多是邻接表）

邻接表最主要应用：存储图和树

（2）数组模拟双链表（用来优化某些问题）

14.单调栈

给定一个长度为N的整数数列，输出每个数***左边第一个比它小的数***，如果不存在则输出 -1。

解法：

构造一个栈，存放数列。当 $i < j$ 且 $a_i > a_j$ 时，就将 a_i 从数列中移出。

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

const int N = 1e5 + 10;
int stack[N], tt;

int main()
{
    int n;
    cin >> n;
    for(int i = 0; i < n; i++)
    {
        int x;
        scanf("%d", &x);
        while(tt && stack[tt] >= x) // 如果栈顶元素大于当前待入栈元素，则出栈
        {
            tt--;
        }
        if(tt)
            cout << stack[tt] << ' ';
        else
            cout << -1 << ' ';
        stack[++tt] = x; // 将当前元素入栈
    }
    return 0;
}
```

15.滑动窗口

给定一个大小为 n≤ 10^6 的数组。有一个大小为 k 的滑动窗口，它从数组的最左边移动到最右边。你只能在窗口中看到 k 个数字。每次滑动窗口向右移动一个位置。

以下是一个例子：

该数组为 [1, 3, -1, -3, 5, 3, 6, 7]，k 为 3

确定滑动窗口位于每个位置时，窗口中的最大值和最小值。

解法：

用一个队列来模拟窗口，每次移动把一个新的数插到队尾，然后把队头删掉。查询最小最大值暴力解决。

（1）用普通队列该怎么做

（2）将队列中的没有用的元素删掉->具有了单调性

（3）可以用O(1)时间从队头/队尾取出最值

应用场景：

求窗口内极值，找出离它最近/最大的元素

16.KMP算法

暴力做法

```
for(int i = 1; i <= n ; i++)
{
    bool flag = true;
    for(int j = 1; j <= m; j++)
    {
        if(s[i + j - 1] != p[j])
        {
            flag = false;
            break;
        }
    }
}
```

对于模版串P，去寻找这一字符串中，重复的值从哪里开始。比如说，从 p_1 到 p_i 寻找从哪里开始的接下来 j 个字符，与 p_1 到 p_j 的字符串一样。即 $p[1,j] = p[i-j+1,i]$ （i 为 p 数组终点）

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

const int N = 1e5 + 10;
const int M = 1e6 + 10;

char p[N], s[M]; // p短，s长
int ne[M];

int main()
{
    int n, m;
    cin >> n >> p + 1 >> m >> s + 1;
    // 求next数组过程
    for(int i = 2, j = 0; i <= m; i++)
    {
        while(j && p[i] != p[j+1])
        {
            j = ne[j];
        }
        if(p[i] == p[j+1])
        {
            j++;
        }
        ne[i] = j;
    }

    for(int i = 1, j = 0; i <= m; i++)
    {
        while(j && s[i] != p[j + 1]) // 当前缀不能匹配或者j已经移动到了开头时
```

```
        {
            j = ne[j];
        }
        if(s[i] == p[j + 1]) // 当两者已经匹配时，移动到下一位
        {
            j++;
        }
        if(j == n) // 匹配成功
        {
            printf("%d ", i - n);
            j = ne[j];
        }
    }
    return 0;
}
```

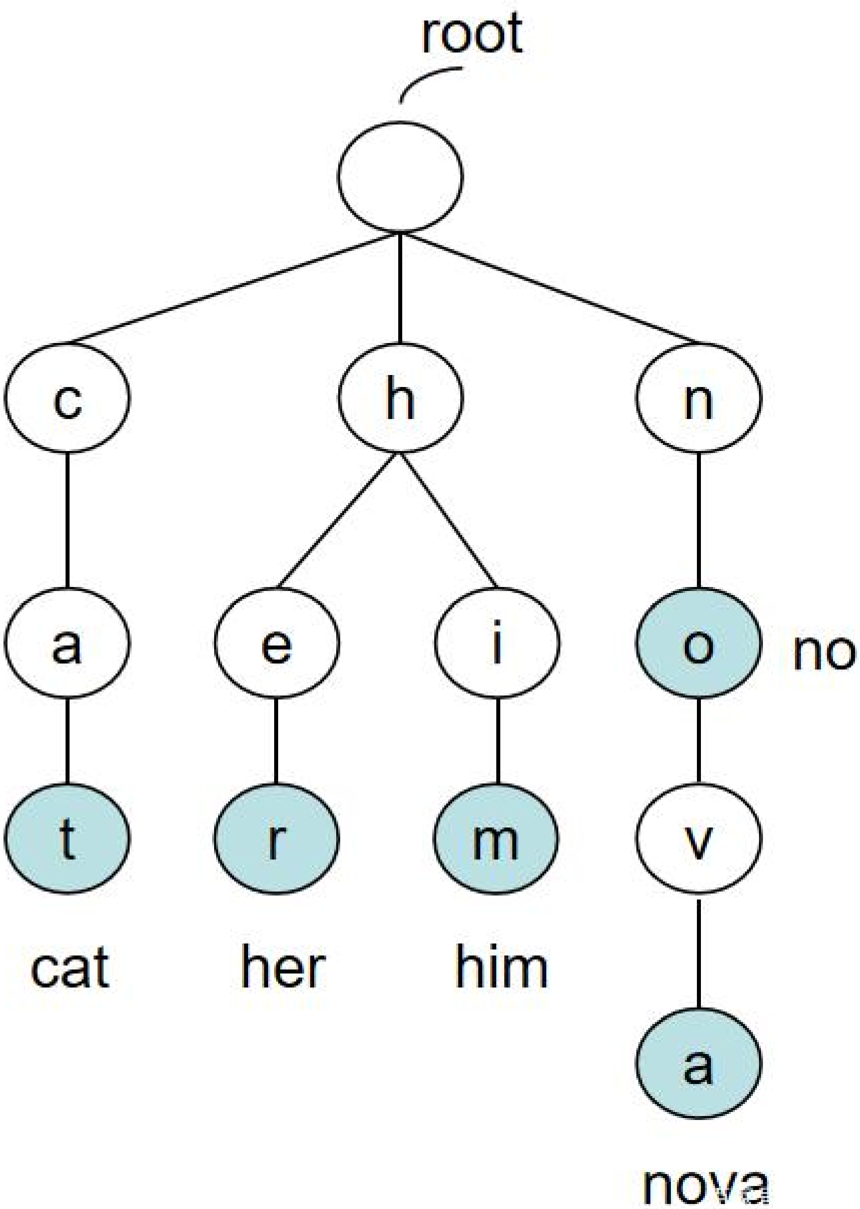
17.Trie树（字典树）

高效的存储和查找字符串集合的数据结构

利用字符串之间的公共前缀，将重复的前缀合并在一起

多叉树结构，每个节点保存一个字符

下图表示了字符串： him 、 her 、 cat 、 no 、 nova 构成的 Trie 树。



在实现过程中，会在叶节点中设置一个标志，用来表示该节点是否是一个字符串的结尾，本例中用青色填充进行标记。

删除一个字符串

（1）待删除的字符串末尾为叶子节点，且与其它字符串无公共前缀。将节点逐一删除即可，例如删除 cat。

（2）待删除字符串末尾不是叶节点。将字符串标志位置为 false 即可，例如删除 no 。

（3）待删除字符串末尾为叶节点，并且中间有其它单词。逐一删除节点，直到待删除节点是另一个字符串的结尾为止，例如删除 nova。

（4）待删除字符串某一节点还有其它子节点。逐一删除节点，如果待删除节点还有其它子节点，则停止删除，例如删除 him。

18.拓扑排序

用DAG图（有向无环图）表示一个工程。顶点表示活动，有向边$< V_i, V_j >$表示活动 V_i 必须先于活动 V_j进行。

拓扑排序的实现:

①从AOV网中选择一个没有前驱(入度为0)的顶点并输出。

②从网中删除该顶点和所有以它为起点的有向边。

③重复①和②直到当前的AOV网为空或当前网中不存在无前驱的顶点为止。

每个AOV网都有一个或者多个拓扑排序序列。

存在回路的图不存在拓扑排序序列。