□

其就是完全二叉树 其可以用连续的数据结构即数组来存储 可分为大顶□和小顶□

- □定义上具有递归性质
- 大顶堆指的是在堆结构中任何一个三元组都满足最大元素在根节点的性质
- 小顶堆指的是在堆结构中任何一个三元组都满足最小元素在根节点的性质
- 堆□实就是优先队列，根据权重比出入队

□的插入和弹出

插入

在数组最后一位插入并执行向上调整

弹出

将□顶元素和最后一位元素交换并执行向下调整 交换后除□顶元素其它部分都符合□的性质

□排序

- 将堆顶元素与堆尾元素进行交换，看成弹出堆顶元素
- 对堆内剩余元素进行调整
- 再将堆顶元素与堆尾元素进行交换，重复上述操作

建□

普通建□法

- 向上调整法 从下往上
- 依次插入节点，从第二个节点开始，比较孩子和父亲，比较完比较下一对

线性建□法

- 采用向下调整法 从上往下
- 从最后一个孩子的父亲节点开始依次向下调整
- 把原数组看成完全二叉树
- 使用向下调整法，从后往前依次扫描每一层节点
- 时间复杂度为O(n)

set（集合）模拟□操作

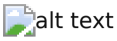
set天生就是小顶□，若要模拟大顶□，对元素先取符号

set基础操作

- 创建set：`set<int> s`；
- 插入：s.insert(3); 类的方法
- set会进行去重处理
  - 若要插入相同元素，可以用pair进行打包
  - 即 `set<pair<int,int>>`
- 遍历集合：使用迭代器遍历（按照值从小到大遍历）
  - for(auto x: s) x是迭代器，s是集合名
  - 若要对pair类型中数据进行读取
    - 使用x.first对第一个字段读取
    - 使用x.second对第二个字段读取
  - s.begin()->first也是可行的

- 访问: s.begin() 返回指向集合第一个元素的迭代器
- s.end()与s.begin()类似
- 删除: s.erase() 删除集合中的元素
- 大小: s.size() 返回集合大小

数据流中的第K大元素

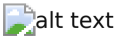


- 第K大元素->建立小顶堆
- 第K小元素->建立大顶堆
- 对k个元素建立堆，若新插入的元素比堆顶元素大，则将插入堆中
- 否则不进行操作，因为这样不会对结果产生影响

```
class KthLargest {
public:
typedef pair<int,int> PII;
int tot,k;
set<PII> s;
KthLargest(int k, vector<int>& nums) {
    this->k = k; //this针: 向类里面变量
    for(auto x: nums)
    {
        add(x);
    }
    return;
}

int add(int val) {
    if(s.size() < k)
    {
        s.insert(PII(val,tot++));
    }
    else{
        if(s.begin()->first < val)
        {
            s.insert(PII(val,tot++));
        }
    }
    if(s.size() > k) s.erase(s.begin());
    return s.begin()->first;
}
};
```

数据流中的中位数



- 用堆的思想来解题
- 双堆法，对顶堆解题

- 将数据分成两段
  - 数据个数为奇数，中位数是前半段的最大值
  - 数据个数为偶数，中位数是前半段的最大值和后半段的最小值的平均值
  - 前半段用大顶堆维护，后半段用小顶堆维护
  - 插入元素和大顶堆的堆顶元素比较
    - 比他大插入到后半段
    - 比他小插入到前半段
- 规定当总个数为奇数，前半段数据个数比后半段数据个数多1
- $n1$是前半段□数据的理论数量 加上1是考虑到总个数可能为奇数

```
class MedianFinder {
public:
    typedef pair<int,int> PII;
    int tot;
    set<PII> s1,s2;
    MedianFinder() {
        tot = 0;
    }

    void addNum(int num) {
        if(s1.size() == 0 || num < -s1.begin()->first)
        s1.insert(PII(-num,tot ++));
        else
        s2.insert(PII(num,tot++));
        int n1 = (s1.size() + s2.size() + 1) / 2;
        if(n1 == s1.size()) return;
        if(s1.size() < n1)
        {
            s1.insert(PII(-s2.begin()->first,tot++));
            s2.erase(s2.begin());
        }
        else
        {
            s2.insert(PII(-s1.begin()->first,tot++));
            s1.erase(s1.begin());
        }
    }

    double findMedian() {
        if((s1.size() + s2.size()) % 2)
        return -s1.begin()->first;
        double a = -s1.begin()->first;
        double b = s2.begin()->first;
        return (a + b) / 2.0;
    }
};
```

合并k个升序链表

alt text

- □□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□
- □□□□□□□□
- □□□□□□□□□□ □□□□□□□□□□□□□□□□

```
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        typedef pair<int,int> PII;
        set<PII> s;
        int n = lists.size();
        for(int i = 0;i < n;i++)
        {
            if(lists[i] == nullptr) continue;
            s.insert(PII(lists[i]->val,i));
        }
        ListNode new_head,*p = &new_head , *q;
        new_head.next = nullptr;
        while(s.size())
        {
            PII a = *s.begin();
            s.erase(s.begin());
            q = lists[a.second];
            lists[a.second] = lists[a.second]->next;
            p->next = q;
            q->next = nullptr;
            p = q;
            if(lists[a.second])
            {
            s.insert(PII(lists[a.second]->val,a.second));
            }
        }
        return new_head.next;
    }
};
```

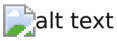
□□

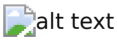
alt text

- □□□□□1
- □□□□□□□(□□□□)□□□□□□□□□□□□□□□□□□□
- □□n□□□ans□□□□□□□□□□ans □□□□□□ alt text
- □□□□□□□□□□□□□□□□(2□3□5)□□□

```
class Solution {
public:
    int nthUglyNumber(int n) {
        set<long long> s;
        s.insert(1);
        long long ans = 0;
        while(n--)
        {
            ans = *s.begin();
            s.erase(s.begin());
            if(ans % 5 == 0)
            {
                s.insert(ans * 5);
            }
            else if(ans % 3 == 0)
            {
                s.insert(ans * 3);
                s.insert(ans * 5);
            }
            else
            {
                s.insert(ans * 2);
                s.insert(ans * 3);
                s.insert(ans * 5);
            }
        }
        return ans;
    }
};
```

超市卖货



- 按照过期日期从小到大进行排序，用数组存储
- 创建一个集合模拟堆，用于存储要出售的商品
- 将新插入集合的商品的过期日期与最晚过期日期的商品进行比较
  - 比□大则直接插入
  - 与□相等则在已排序的集合中找一个利润最小的商品与□进行利润比较，利润大的留在集合中
  - 由于已经排好序，因此新插入的商品不可能比集合中最晚过期的商品晚 
- 我们用到C++里面的构造函数，用于初始化结构体里面的变量，即Data构造函数，他在调用结构体时会自动执行，并对成员变量进行赋值

```
#include<bits/stdc++.h>
using namespace std;
struct Data
{
    int p,d;
    Data(int p,int d):p(p),d(d){}
```

```
        bool operator<(const Data &obj)const
        {
            if(d != obj.d) return d < obj.d;
            return p > obj.p;
        }
    };

    typedef pair<int,int> PII;

    int main()
    {
        int n;
        cin >> n;
        vector<Data> arr;
        set<PII> s;
        for(int i = 0,p,d;i < n;i++)
        {
            cin >> p >> d;
            arr.push_back(Data(p,d));
        }
        sort(arr.begin(),arr.end());
        for(int i = 0;i < n;i++)
        {
            if(arr[i].d > s.size())
                s.insert(PII(arr[i].p,i));
            else
            {
                if(arr[i].p > s.begin()->first)
                {
                    s.erase(s.begin());
                    s.insert(PII(arr[i].p,i));
                }
            }
        }
        int ans = 0;
        for(auto x: s)
            ans += x.first;
        cout << ans;
        return 0;
    }
```

序列**M**小和

 alt text 这道题就是要注意第**i**行的前**M**小和就在第**i-1**行和第**i**行**m**个数字组成的**m * m**种情况中  alt text

```
    #include<bits/stdc++.h>
    using namespace std;
    typedef pair<int,int> PII;
    int main()
    {
```

```
    int n,m,t = 0;
    cin >> n >> m;
    set<PII> s;
    s.insert(PII(0, t++));
    for(int i = 0;i < n;i++)
    {
        vector<int> temp;
        for(auto x : s)
        {
            temp.push_back(x.first);
        }
        s.clear();
        for(int j = 0 ,a;j < m;j++)
        {
            cin >> a;
            for(auto x: temp)
            {
                if(s.size() < m || s.begin()->first < x - a)//□□□□□□□
                    s.insert(PII(x - a,t++));
                if(s.size() > m) s.erase(s.begin());
            }
        }
    }
    int flag = 0;
    for(auto iter = s.rbegin();iter != s.rend();iter++)//□□□□□□
    {
        if(flag) cout << " " ;
        cout << -iter->first ;
        flag = 1;
    }
    return 0;
}
```