1. 在计算机科学与社会生活中，经常涉及到要求前K个元素即top-k的问题，请给出不同策解决这一问题，并对比分析。

- 直接排序法：简单粗暴，将所给数组排序，可使用快速排序得到递增数组，再输出前n个数即可。

  给出伪代码如下：

```
void topK(int* arr,int n,int k){
    int* arr_topK=quickSort(arr,n);
    fot(int i=0;i<k;i++){
        cout<<arr[i]<<" ";
    }
    cout<<endl;
}
```

该方法时间复杂度为O(nlog n)，主要是排序较浪费时间。

- 局部冒泡法：由冒泡排序可知，冒泡排序每一轮都能找到一个最大值，因此冒泡排序k轮就可以找到最大的k个数。

  给出伪代码如下：

```
void part_bubbleSort(int* arr,int n,int k){
    for(int i=0;i<k;i++){
        for(int j=0;j<n-i-1;j++){
            sawp(arr[j],arr[j+1]);
        }
    }
    for(int i=n;i>n-k;i--){
        cout<<arr[i]<<" ";
    }
    cout<<endl;
}
```

- 维护最小堆法：我们可以维护一个大小为k的小顶堆，由小顶堆的定义可知，任意一个顶点的值都不大于其子节点。那么k大小的小顶堆中元素最小值是根结点上元素，其余元素都比该元素大。这时数组中剩余n-k个元素与根结点元素比较，若大于根结点元素，只需弹出堆顶元素，加入该元素并维护小顶堆；若小于根结点元素则比较下一组。

  代码如下：

```
//top-K问题，求数组前k大的元素
//c++ stl 中priority_queue为我们提供了优先队列的实现，其底层是堆排序
#include<iostream>
#include<queue>
#include<algorithm>
#define MaxSize 20
using namespace std;
int main(){
    //将数组arr中前k大的元素放入堆中
    int arr[MaxSize];
    srand((unsigned)time(NULL));
    cout<<"数组元素为："<<endl;
```

```
    for (int i = 0;i<MaxSize;i++){
        arr[i] = rand()%100;
        cout<<arr[i]<<" ";
    }
    cout<<endl;

    cout<<"为方便测试，将数组排序后结果输出："<<endl;
    sort(arr,arr+MaxSize);
    for (int i = 0;i<MaxSize;i++){
        cout<<arr[i]<<" ";
    }
    cout<<endl;

    int k = 5;
    //定义最小堆，堆顶元素为最小元素
    priority_queue<int,vector<int>,greater<int>> q;
    //将前k个元素放入堆中
    for (int i = 0;i<k;i++){
        q.push(arr[i]);
    }
    //遍历数组，将大于堆顶元素的元素放入堆中
    for (int i = k;i<MaxSize;i++){
        if (arr[i]>q.top()){
            q.pop();
            q.push(arr[i]);
        }
    }
    //输出堆中元素
    cout<<"前"<<k<<"大的元素为："<<endl;
    while (!q.empty()){
        cout<<q.top()<<" ";
        q.pop();
    }
    cout << endl;
    return 0;
}
```

测试结果如下：

```
问题    输出    调试控制台    终端

/home/liqi/Documents/code/C_Cpp/Algorithm/build/algorithm
[liqi@myArch build]$ /home/liqi/Documents/code/C_Cpp/Algorithm/build/algorithm
数组元素为：
63 0 56 62 35 92 78 55 67 7 99 63 98 48 34 71 14 90 43 90
为方便测试，将数组排序后结果输出：
0 7 14 34 35 43 48 55 56 62 63 63 67 71 78 90 90 92 98 99
前5大的元素为：
90 90 92 98 99
[liqi@myArch build]$
```

2. 请用递归方式实现堆排序，并进行性能分析。

我们可以构建最大堆(或最小堆)，构建出堆最大堆，则堆根结点元素是其中堆最大值，将其排除再构建最大堆便可以将数组排序。该算法每次维护最大堆时间复杂度为O(log n)，共维护n次，故O(n*log n)

代码如下：

```
//请用递归方式实现堆排序
#include <iostream>
#define MaxSize 5
using namespace std;
//维护最大堆
void heapify(int arr[], int n, int i) {
    //i为当前节点
    int largest = i;
    int l =2*i+1;//左子结点
    int r =2*i+2;//右子结点
    if (l<n&&arr[l]>arr[largest])//当前节点与左子结点比较
        largest = l;
    if (r<n&&arr[r]>arr[largest])
        largest = r;
    if (largest != i) {//需要交换
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);//递归，维护下一层
    }
}

//递归堆排序
//过程是先建立最大堆，然后将堆顶元素与最后一个元素交换，然后维护最大堆
void heapSort(int* arr, int n) {
    for (int i=n/2-1;i>=0;i--){//n个元素，最后一个元素的父节点下标为(n-1)/2
        heapify(arr,n,i);
    }
    for (int i=n-1;i>0;i--) {//每次构成大顶堆，能找到一个最大值
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}

int main(){
    int arr[MaxSize];
    srand((unsigned)time(NULL));
    cout<<"数组元素为："<<endl;
    for (int i = 0;i<MaxSize;i++){
        arr[i] = rand()%100;
        cout<<arr[i]<<" ";
    }
    cout<<endl;
    //堆排序
    heapSort(arr,MaxSize);

    cout<<"排序后结果输出："<<endl;
    for (int i = 0;i<MaxSize;i++){
        cout<<arr[i]<<" ";
    }
    cout<<endl;
    return 0;
}
```

测试结果如下：


问题
输出
调试控制台
终端
/home/liqi/Documents/code/C_Cpp/Algorithm/build/algorithm
[liqi@myArch build]$ /home/liqi/Documents/code/C_Cpp/Algorithm/build/algorithm
数组元素为：
48 13 88 86 36
排序后结果输出：
13 36 48 86 88
[liqi@myArch build]$