1、快速排序

```
#include <iostream>
using namespace std;
int partition(vector<int>& arr, int left, int right) {
    ///也可以弄个随机数，与最右交换过来，最后换回去
    int key = arr[right]; ///最右为轴点
    int k = left;
    for(int i = left; i < right; ++i) {
        if(arr[i] < key) {
            swap(arr[i], arr[k++]); //先交换arr[k],k++
        }
    }
    swap(arr[right], arr[k]);
    return k;
}

void quicksort(vector<int>& arr, int left, int right) {
    //if(left < right) {
    //    int i = partition(arr, left, right);
    //    quicksort(arr, left, i-1);
    //    quicksort(arr, i+1, right);
    //}
    //return;
    if(left == right) return;
    int index = partition(arr, left, right);
    if(index > left)  quicksort(arr,left,index-1);
    if(index < right) quicksort(arr,index+1,right);
}
```

2、归并排序

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

void merge(vector<int>& A, vector<int> L, vector<int> R) {
    int l = L.size();
    int r = R.size();
    int i = 0;
    int j = 0;
    int k = 0;
    while(i < l && j < r) {
        if(L[i] < R[j]) {
            A[k++] = L[i++];
        } else {
            A[k++] = R[j++];
        }
    }
    while(i < l) A[k++] = L[i++];
    while(j < r) A[k++] = R[j++];
}
```

```
void mergesort(vector<int>& arr) {
    int n = arr.size();
    if(n < 2) return;
    int mid = n/2;
    int i;
    vector<int> L(mid);
    vector<int> R(n - mid);
    for(i = 0; i < mid; ++i) {
        L[i] = arr[i];
    }
    for(;i < n; ++i) {
        R[i-mid] = arr[i];
    }
    mergesort(L);
    mergesort(R);
    merge(arr, L, R);
}
```

3、冒泡排序

两种方法冒泡排序的最小时间代价θ(n)，最大时间代价，和平均时间代价均为θ(n^2)。

```
#include <iostream>
using namespace std;
void swap(int array[], int i, int j)
{
    int temp = array[i];
    array[i] = array[j];
    array[j] = temp;
}
void BubbleSort1(int array[], int n)
{
    for (int i = 0; i < n-1; i++)
    {
        for (int j = i + 1; j < n-1; j++)
        {
            if (array[i]>array[j])
                swap(array, j, i);//每次i后面的元素比array[i]小就交换。
        }
    }
}
void BubbleSort2(int array[], int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = n - 1; j >= i;j--)
        {
            if (array[j - 1]>array[j])//从后面到i个元素两两比较，把小的不断上顶
                swap(array, j, j - 1);
        }
    }
}
```

合并两个排序链表

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
    if(l1==NULL) {return l2;
    }else if(l2==NULL) {return l1;
    }else if(l1->val < l2->val){
        l1->next = mergeTwoLists(l1->next ,l2);
        return l1;
    }else{
        l2->next = mergeTwoLists(l1, l2->next);
        return l2;
    }

    }
};///迭代

class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {

    ListNode* mergedhead = new ListNode(-1);
    ListNode* prev = mergedhead;

    while(l1 != NULL && l2 != NULL){
       if(l1->val < l2->val){
        prev -> next = l1;
        l1 = l1->next;
       }else{
        prev -> next = l2;
        l2 = l2->next;
       }
       prev = prev -> next;
    }

    prev-> next = l1 == NULL? l2 : l1; ///l1和l2长度不一样
    return mergedhead -> next;

    }
};  //递归
```

求1+2+3+...+n

```
class Solution {
public:
    int sumNums(int n) {
        int ans = n;
        ans && (ans+= sumNums(ans-1));
        return ans;
    }
};

// class test{
// public:
```

```
//     test(){n++;sum+=n;}
//     void reset(){n=0;sum=0;}
//     int getnum(){return sum;}
// private:
//     int n;
//     int sum;
// }
// int sum_solution(int n){
//     test::reset;
//     test *a = test[n];
//     delete []a;
//     a=null;
//     return test::getnum();
// } //构造函数

// typedef int (*fun)(int);
// int (*solution_end)(int n){
//    return 0;
// }
// int (*sum_solution)(int n){
//    fun f[2]={solution_end,sum_solution};
//    return n + f[!!n](n-1);
// }///函数指针

// template<int n>struct sum_solution{
//     enum value{ N = sum_solution<n-1>::N+n};
// };
// template<>struct sum_solution<1>{
//     enum value{ N=1 };
// };
```