备忘用~   
这些函数统一接口问 void xxxsort(int \*a, int n),n是长度

void swap(int &x, int &y)

{

x = x^y;

y = x^y;

x = x^y;

}

* 1
* 2
* 3
* 4
* 5
* 6

O(N\*N)的三大算法   
（一）选择排序   
核心思想，每次用后面m个元素中查找最小的放在当前m个元素的第一个位置上。

void selecttsort(int \*a, int n)

{

for (int i = 0; i < n - 1; ++i)

{

int index = i;

int minv = a[i];

for (int j = i+1; j < n; ++j)

{

if (a[j] < minv)

{

minv = a[j];

index = j;

}

}

if (index != i)

swap(a[index], a[i]);

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18

（二）插入排序   
核心思想，类似于打牌，针对每个元素，插入到“适当”的位置上。

void insertsort(int \*a, int n)

{

for (int i = 1; i < n; ++i)

{

int j;

for (j = i - 1; j >= 0; --j)

if (a[j] < a[i])

break;

++j;// 此时j++，是因为上面的j是小于i，因为要把所有>=i的后移，所以j++就代表[j,x]的元素后移

int tmp = a[i];

for (int k = i - 1; k >= j; --k)

a[k + 1] = a[k];

a[j] = tmp;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17

（三）冒泡排序   
核心思想就是不断的从上面到下面两两比较，交换。   
这个是基本算法，可以修改为双向冒泡等

void popsort(int \*a, int n)

{

for (int i = 0; i < n - 1; ++i)

{

for (int j = 0; j <= n - i - 2; ++j)

{

if (a[j] > a[j + 1])

swap(a[j], a[j + 1]);

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12

O（Nlog(N)）的三种方法   
（四）快速排序   
核心思想，找到一个锚点，然后针对锚点双侧迭代执行排序。

int getmidinQS(int \*a, int b, int e)

{

//array in [b,e]

int base = a[b];

while (b < e)

{

while (b < e && a[e] >= base)

--e;

if (b < e)

swap(a[b], a[e]);

while (b < e && a[b] <= base)

++b;

if (b < e)

swap(a[b], a[e]);

}

return b;

}

void quicksort(int \*a, int b, int e)

{

if (b < e)

{

// 1 get min

int mid = getmidinQS(a, b, e);

// 2 quick sort at left side and right side

quicksort(a, b, mid - 1);

quicksort(a, mid+1, e);

}

}

void quicksort(int \*a, int n)

{

return quicksort(a, 0, n - 1);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34

（五）归并排序   
核心思想，先两边分别排序再归并。关键是归并。

// merge data [b,mid][mid+1,e] to [b,e]

void mergetwo(int \*a, int b, int mid, int e)

{

//merge [b,mid] + [mid+1,e] -->[b,e]

int len1 = mid - b + 1;

int len2 = e - mid;

if (!len1 || !len2) return;

int \* p = new int[len1];

int \*q = new int[len2];

memcpy(p, a+b, sizeof(int)\*len1);

memcpy(q, a + b+len1, sizeof(int)\*len2);

int i = 0, j = 0,k=b;

while (i < len1&&j < len2)

{

if (p[i] < q[j])

{

a[k++] = p[i++];

}

else

{

a[k++] = q[j++];

}

}

while (i < len1)

{

a[k++] = p[i++];

}

while (j < len2)

{

a[k++] = q[j++];

}

delete[]p;

delete[]q;

}

void mergesort(int \*a, int b,int e)

{

//1 merge sort two side

if (b < e)

{

int mid = (b + e) / 2;

mergesort(a, b, mid);

mergesort(a, mid+1, e);

//2 merge two side to one whole

mergetwo(a, b, mid, e);

}

}

void mergesort(int \*a, int n)

{

return mergesort(a, 0, n - 1);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52

（六）堆排序   
关键思想：   
1：建立堆。从小到大排序是建立大顶堆，从大到小排序是建立小顶堆。   
建堆的过程，也就是针对堆（堆，这里就是完全二叉树，而数组也是一个完全二叉树的表示媒介）的每一个非叶子节点进行调整的过程。之所以不考虑叶子节点，是以为堆的性质使得我们只需要关注非叶子节点。   
调整的过程分两步：   
1）非叶子节点和两个孩子（至少存在一个），取最大值。   
2）如果当前非叶子节点不拥有最大值，则交换两个值，此时有可能被交换的孩子本身也是有孩子的，因此可以还需要重复调整，正好使用递归重复调用。   
2：堆排序   
堆排序，就是N-1次地，每次拿堆顶元素（次数假设为i，i=0,1,,,N-1）和当前数组第N-1-i个元素交换（交换后，从数组最后一个元素开始，陆陆续续得到了一个排序序列），然后针对堆顶元素进行调整。   
所以，堆排序和建立堆都用到了调整堆的子过程。

void modifyheap(int \*a, int base, int len)

{

int lc = base \* 2 + 1;// left child index

int rc = lc + 1;//right child index

int maxv = a[base];

int index = base;

if (lc >= len) return;

if (lc < len && maxv < a[lc])

{

index = lc;

maxv = a[lc];

}

if (rc < len && maxv < a[rc])

{

index = rc;

maxv = a[rc];

}

if (index != base)

{

swap(a[base], a[index]);

modifyheap(a, index, len);

}

}

void buildheap(int \*a, int n)

{

for (int i = n / 2 - 1; i >= 0; --i)

{

modifyheap(a, i, n);

}

}

void heapsort(int \*a, int n)

{

buildheap(a, n);

for (int i = 0; i <n-1; ++i)

{

swap(a[0], a[n - i - 1]);

modifyheap(a, 0, n - i - 1);

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42

（七）希尔排序

void shellsort(int \*a, int n)

{

int gap = n / 2;

for (; gap > 0; gap /= 2)

for (int i = 0; i < gap; ++i)

{

// use select sort to sort data at,i,i+gap,i+2gap....

// use select sort internal

for (int j = i + gap; j < n; j += gap)

{

int data = a[j],k;

for (k = j - gap; k >= i; k -= gap)

{

if (a[k] < a[j])

break;

}

if (k < i)

k = i;

else

k += gap;// move data [k,,,j-gap] to the next one

for (int l = j - gap; l >= k; l -= gap)

a[l + gap] = a[l];

a[k] = data;

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28

（八~十）   
（八）计数排序

如下程序所示，其中p数组存放的是不大于数据i的数据的个数。

void countsort(int \*a, int \*b, int n, int k)

{

int \*p = new int[k+1];

memset(p, 0, sizeof(int)\*(k+1));

for (int i = 0; i < n; ++i)

p[a[i]]++;

for (int i = 1; i <= k; ++i)

p[i] += p[i - 1];

for (int i = 0; i < n; ++i)

{

b[p[a[i]] - 1] = a[i];

p[a[i]]--;

}

delete[]p;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17

（九）基数排序   
（十）桶排序   
个人理解桶排序实际上就是建立hash的过程，是hash的一个实现。   
比如hash的长度是10，那么桶个数就是10，然后每个桶里面的数据可以灵活处理（比如按照hash冲突之挂链法形成一条链表或者修改数据结构增加一个表示count的字段等）   
因此对于数据个数是N，桶个数是M来讲，时间复杂度为   
O(N)+M\*N/M(logN/M)   
=O(N)+NlogN-NlogM   
所以当桶个数和数据个数一致的时候，时间复杂度为O(N)   
而此时的所谓桶排序又有了两种类似的算法   
1：直接根据数据范围定义一个数组，比如1000个数据定义个1000长度数组   
2：直接用bitmap   
这两种方法都可以用来做判空或者排序使用。   
所以总的来讲，桶排序就是把数据分段，然后段内排序。   
所形成的数据结构可以是hash表。

因此从时间复杂度上来讲，可以简记为   
N2 选择排序 +插入排序 +冒泡排序   
NlogN 快速排序 +归并排序 +堆排序+   
近似O(N)hash、bitmap、桶排序、计数排序等。

====================================================   
对于查找存在性的题目   
（1）如果数据已经有序，那么使用索引直接查找（hash、bitmap）复杂度是最低的   
否则可以考虑用二分查找，复杂度为logN。   
（2）数据无序时，查找某数据存在性，可以考虑数据排序再查找，但是此时复杂度至少为NlogN   
也可以不考虑排序，直接建立hash，根据hash值来判断，复杂度可能更低。   
（3）判断数值存在性，若数据范围确定的话也可以用bitmap，如果采用2bit表征的话，还可以表示“数据存在一次”“数据存在两次”等。范围不确定的话判断存在性可以使用hash   
总之查找复杂度从低到高为O（1），O（logN）O（N）   
分别为hash、bitmap<二分查找<顺次查找。

进行海量数据处理时，排序会很耗时，   
比如<http://www.bianceng.cn/Programming/sjjg/201412/47166_3.htm>   
对500万个高考成绩排序，若直接使用快排等复杂度为500万log（500万）   
而如果建立hash，分值相同的成绩（包括学生姓名，成绩等）挂在一条链上，则只需要O(N)   
响应的，桶排序也可以达到这样的效果。计数排序等也可以。

比如1亿个QQ号码登录信息，求按照登录次数对QQ号码排序。   
1：先针对QQ号码做hash   
2：依次读文件或者内存数据，相同QQ号码则对登录次数做自加，hash值相同的QQ号码则挂在一条链上   
3：文件读入完毕之后，对于每一个hash值下的QQ号码（即hash值相同的号码）做登录次数排序   
4：所有hash值下的信息做归并排序。