

□□□□

- □□□□□□□□□□
- □□□□□□□□□□□□□□□□

```
#include<bits/stdc++.h>
using namespace std;
int *getranddata(int n)
{
    int *arr = new int [n];
    for(int i = 0;i < n;i++)
    {
        arr[i] = rand()%1000;
    }
    return arr;
}

bool check(int *arr,int n)
{
    for(int i = 1;i < n;i++)
    {
        if(arr[i] < arr[i - 1])
            return false;
    }
    return true;
}

void print(int *arr ,int n)
{
    for(int i = 0;i < n;i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
    return;
}

void selection_sort(int *arr,int n)
{
    for(int i = 0;i < n - 1;i++)
    {
        int ind = i;
        for(int j = i + 1;j < n;j++)
        {
            if(arr[j] < arr[ind]) ind = j;
        }
        swap(arr[i],arr[ind]);
    }
    return;
}

int main()
```

```
{
    int n = 100;
    srand(((unsigned)time(NULL)));
    int *arr = getranddata(n);
    selection_sort(arr,n);
    print(arr,n);
    cout << check(arr,n) << endl;
    delete arr;
    return 0;
}
```

□□□□

- [illegible]

```
#include<bits/stdc++.h>
using namespace std;
int *getranddata(int n)
{
    int *arr = new int [n];
    for(int i = 0;i < n;i++)
    {
        arr[i] = rand()%1000;
    }
    return arr;
}

bool check(int *arr,int n)
{
    for(int i = 1;i < n;i++)
    {
        if(arr[i] < arr[i - 1])
            return false;
    }
    return true;
}

void print(int *arr ,int n)
{
    for(int i = 0;i < n;i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
    return;
}

void insert_sort(int *arr,int n)
{

```

```

    for(int i = 1; i < n; i++)
    {
        int j = i;
        while(j > 0 && arr[j - 1] > arr[j])
        {
            swap(arr[j - 1], arr[j]);
            j -= 1;
        }
    }
    return;
}

int main()
{
    int n = 100;
    srand((unsigned)time(NULL));
    int *arr = getranddata(n);
    insert_sort(arr, n);
    print(arr, n);
    cout << check(arr, n) << endl;
    delete arr;
    return 0;
}

```

時間複雜度

- 時間複雜度(最壞情況) $O(n^2)$
- CPU時間複雜度
- 空間複雜度

優化

- 使用 `while` 循環 `j > 0`
- `j > 0` 時間複雜度 $O(n^2)$
- $O(n)$ 時間複雜度 $O(n^2)$

```

void unsupervise_insert_sort(int *arr, int n) // 優化
{
    int ind = 0;
    for(int i = 1; i < n; i++)
    {
        if(arr[ind] > arr[i]) ind = i;
    }
    while(ind > 0)
    {
        swap(arr[ind], arr[ind - 1]);
        ind -= 1;
    }
    for(int i = 1; i < n; i++)
    {
        int j = i;
        while(arr[j - 1] > arr[j])
        {

```

```

        swap(arr[j - 1],arr[j]);
        j -= 1;
    }
}
return;
}

```

复杂度

- 时间复杂度
- 空间复杂度
- 稳定性
- 是否原地排序

$O(n^2)$ (最坏情况) $n/2, n/4, n/8, n/16, \dots$ **$O(n^{1.5})$** (平均情况)
 Hibbard序列: $1, 3, 7, \dots, 2^{k-1}$

实现

```

void shell_sort(int *arr,int n)
{
    int k = 2,step;
    do
    {
        step = n / k == 0 ? 1:n / k;
        for(int i = 0;i < step;i++) //对step个元素进行插入排序
            unsupervise_insert_sort(arr,n,step);
        k *= 2;
    }while(step != 1);
    return;
}

```

Hibbard

```

void shell_sort_hibbard(int *arr,int n)
{
    int step = 1;
    while(step <= n / 2)    step = step * 2 + 1;
    do
    {
        step /= 2;
        for(int i = 0;i < step;i++)
            unsupervise_insert_sort(arr,n,step);
    }while(step > 1);
    return ;
}

```

优化

- 当cnt达到一定值时break

```

#include<bits/stdc++.h>
using namespace std;
void bubble_sort(int *arr,int n)
{
    int cnt;
    for(int i = n;i > 0;i--)
    {
        cnt = 0;
        for(int j = 1;j < i;j++)
        {
            if(arr[j] >= arr[j - 1]) continue;
            swap(arr[j],arr[j - 1]);
            cnt += 1;
        }
        if(cnt == 0) break;
    }
    return ;
}

int *getranddata(int n)
{
    int *arr = new int [n];
    for(int i = 0;i < n;i++)
        arr[i] = rand() % 1000;
    return arr;
}

int check(int *arr ,int n)
{
    for(int i = 1;i < n;i++)
        if(arr[i] < arr[i - 1]) return 0;
    return 1;
}

void print(int *arr,int n)
{
    for(int i = 0;i < n;i++)
        cout << arr[i] << " ";
    cout << endl;
    return ;
}

int main()
{
    srand((unsigned)time(NULL));
    int n = 100;
    int *arr = getranddata(n);
    bubble_sort(arr,n);
    print(arr,n);
    cout << check(arr,n) << endl;
    delete arr;
}

```

```

    return 0;
}

```

□□□□

- □□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□
 - □□□□□□□□□□□□
 - □□□□□□□□□□□□□□□□□□
 - □□□□□□□□□□□□□□□□□□□□□□□□
 - □□□□□□□□□□□□□□□□□□□□□□□□
 - □□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□(□□)

```

#include<bits/stdc++.h>
using namespace std;

void quick_sort(int *arr ,int l,int r) //□□□□
{
    if(r - l <= 2) //□□□□
    {
        if(r - l <= 1) return;
        if(arr[l] > arr[l + 1]) swap(arr[l],arr[l + 1]);
        return ;
    }
    //□□□□
    int x = l,y = r - 1,z = arr[l];
    while(x < y)
    {
        while(x < y && z <= arr[y]) y--;
        if(x < y) arr[x] = arr[y];
        while(x < y && z >= arr[x]) x++;
        if(x < y) arr[y] = arr[x];
    }
    arr[x] = z;
    quick_sort(arr,l,x);
    quick_sort(arr,x+1,r);
}

int *getranddata(int n)
{
    int *arr = new int [n];
    for(int i = 0;i < n;i++)
        arr[i] = rand() % 1000;
    return arr;
}

int check(int *arr ,int n)
{
    for(int i = 1;i < n;i++)

```



```

    for(int i = l + 1; i < r; i++)
    {
        int j = i;
        while(arr[j - 1] > arr[j])
        {
            swap(arr[j - 1], arr[j]);
            j -= 1;
        }
    }
    return;
}

void print(int *arr, int n)
{
    for(int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
    return ;
}

```

- 冒泡

```

void quick_sort(int *arr, int l, int r) //快速
{
    if(r - l <= 2) //快速
    {
        if(r - l <= 1) return;
        if(arr[l] > arr[l + 1]) swap(arr[l], arr[l + 1]);
        return ;
    }
    //快速
    int x = l, y = r - 1, z = arr[l];
    while(x < y)
    {
        while(x < y && z <= arr[y]) y--;
        if(x < y) arr[x] = arr[y];
        while(x < y && z >= arr[x]) x++;
        if(x < y) arr[y] = arr[x];
    }
    arr[y] = z;
    quick_sort(arr, l, x);
    quick_sort(arr, x + 1, r);
}

```

- 快速排序

```

void quick_sort_v1(int *arr, int l, int r) //快速排序
{
    if(r - l <= 2) //快速
    {

```



```

        if(r - l <= 1) return;
        if(arr[l] > arr[l + 1]) swap(arr[l],arr[l + 1]);
        return ;
    }
    //快速
    int x = l,y = r - 1,z = arr[l];
    while(x <= y)
    {
        while(z < arr[y]) y--;
        while(z > arr[x]) x++;
        if(x <= y)
            swap(arr[x++], arr[y--]);
    }
    quick_sort_v1(arr,l,x);
    quick_sort_v1(arr,x,r);
}

```

- 快速

```

int way(int a,int b,int c)
{
    if(a > b) swap(a,b);
    if(a > c) swap(a,c);
    if(b > c) swap(b,c);
    return b;
}

void quick_sort_v2(int *arr ,int l,int r) //快速
{
    if(r - l <= 2) //快速
    {
        if(r - l <= 1) return;
        if(arr[l] > arr[l + 1]) swap(arr[l],arr[l + 1]);
        return ;
    }
    //快速
    int x = l,y = r - 1;
    int z = way(arr[l],arr[r - 1],arr[(l + r) / 2]);
    while(x < y)
    {
        while( z < arr[y]) y--;
        while( z > arr[x]) x++;
        if(x <= y)
            swap(arr[x++], arr[y--]); //x++y-- 交换x和y
    }
    quick_sort_v2(arr,l,y+1);
    quick_sort_v2(arr,x,r);
}

```

- 快速排序的递归实现

```
void quick_sort_v3(int *arr ,int l,int r) //快速排序的递归实现
{
    while(l < r)
    {
        if(r - l <= 2) //递归
        {
            if(r - l <= 1) return;
            if(arr[l] > arr[l + 1]) swap(arr[l],arr[l + 1]);
            return ;
        }
        //递归
        int x = l,y = r - 1;
        int z = way(arr[l],arr[r - 1],arr[(l + r) / 2]);
        while(x <= y)
        {
            while(z < arr[y]) y--;
            while(z > arr[x]) x++;
            if(x <= y)//递归
                swap(arr[x++], arr[y--]);
        }
        quick_sort_v3(arr,l,x);
        l = x;
    }
    return ;
}
```

- 快速排序的非递归实现

```
void __quick_sort_v4(int *arr ,int l,int r) //快速排序的非递归实现
{
    while(r - l > 16)
    {
        //递归
        int x = l,y = r - 1,z = arr[l];
        while(x <= y)
        {
            while(z < arr[y]) y--;
            while(z > arr[x]) x++;
            if(x <= y) //递归
                swap(arr[x++], arr[y--]);
        }
        __quick_sort_v4(arr,l,x);
        l = x;
    }
    return ;
}
```

```
void quick_sort_v4(int *arr ,int l,int r) //快速
{
    __quick_sort_v4(arr,l,r);
    unsupervise_insert_sort(arr,l,r);
}
```

- 快速排序的时间复杂度为 $O(n \log n)$

归并排序

- 归并排序的时间复杂度为 $O(n \log n)$
- 归并排序的空间复杂度为 $O(n)$
- 归并排序是稳定的
- 归并排序的时间复杂度为 $O(n \log n)$
- 归并排序的时间复杂度为 $O(n \log n)$

```
void merge_sort(int *arr,int l,int r)
{
    if(r - l <= 1) return ;
    int mid = (l + r) / 2;
    merge_sort(arr,l,mid);
    merge_sort(arr,mid,r);
    int p1 = l , p2 = mid , k = 0;
    while(p1 < mid || p2 < r)
    {
        if(p2 == r || (p1 < mid && arr[p1] <= arr[p2]))
            temp[k++] = arr[p1++];
        else
            temp[k++] = arr[p2++];
    }
    for(int i = l;i < r;i++)
        arr[i] = temp[i - l];
    return ;
}
```

快速排序

快速排序

- 快速排序的时间复杂度为 $O(n \log n)$
- 快速排序的空间复杂度为 $O(n)$
- 快速排序是稳定的
- 快速排序的时间复杂度为 $O(n \log n)$

快速排序的时间复杂度为 $O(n \log n)$

- 快速排序的时间复杂度为 $O(n \log n)$
- 快速排序的时间复杂度为 $O(n \log n)$

```

#include<bits/stdc++.h>
using namespace std;

void print(int *arr,int l,int r)
{
    for(int i = l;i < r;i++)
    {
        if(i) cout << " ";
        cout << arr[i];
    }
    cout << endl;
    return;
}

int *getnewdata(int l ,int r)
{
    srand((unsigned)time(NULL));
    int *arr = new int[r - l];
    for(int i = l;i < r;i++)
    {
        arr[i] = rand() % 100;
    }
    return arr;
}

#define K 10
void radix_sort(int *arr,int l,int r)
{
    int *cnt = new int[K];
    int *temp = new int[r - l];
    memset(cnt,0,sizeof(int) * K);
    for(int i = l;i < r;i++) cnt[arr[i] % K] += 1;
    for(int i = 1;i < K;i++) cnt[i] += cnt[i - 1]; //[]
    for(int i = r - 1;i >= l;i--) temp[--cnt[arr[i] % K]] = arr[i];
    memcpy(arr + l,temp,sizeof(int) * (r - l));
    memset(cnt,0,sizeof(int) * K);
    for(int i = l;i < r;i++) cnt[arr[i] / K] += 1;
    for(int i = 1;i < K;i++) cnt[i] += cnt[i - 1]; //[]
    for(int i = r - 1;i >= l;i--) temp[--cnt[arr[i] / K]] = arr[i];
    memcpy(arr + l,temp,sizeof(int) * (r - l));
    delete cnt,temp;
    return ;
}

bool check(int *arr,int l,int r)
{
    for(int i = l + 1;i < r;i++)
    {
        if(arr[i] < arr[i - 1]) return false;
    }
}

```

```

        return true;
    }
#define n 10000
int main()
{
    int *arr = getnewdata(0,n);
    radix_sort(arr,0,n);
    //print(arr,0,n);
    cout << check(arr,0,n);
    delete arr;
    return 0;
}

```

sort

- 对vector1中的元素按照vector2中的元素(从小到大)
- 对vector
- 对vector中的元素按照3个元素greater<vector<T>>()
- vector.end()返回的是vector的结束位置

对vector

1. 对vector

```

bool cmp(int a,int b)
{
    return a > b; //对vector中的元素按照true
}
sort(arr,arr + 10;cmp);

```

2. 对vector

```

bool cmp(struct a,struct b)
{
    if(a.x != b.x) return a.x > b.x;
    return a.y < b.y;
}

```

- 对vector×vector中的元素按照y

对vector

 alt text

- 对vector中的元素按照
- 对vector中的元素按照
- 对vector中的元素按照(从小到大)
- 对vector中的元素按照target p1 p2
- 对vector arr


```

        else
        {
            p->next = h2;
            h2 = p;
        }
        p = q;
    }
    h1 = sortList(h1);
    h2 = sortList(h2);
    p = h1;
    while(p->next) p = p->next;
    p->next = h2;
    return h1;
}
};

```

- 递归

```

class Solution {
public:
    int getlength(ListNode *head)
    {
        int n = 0;
        while(head) n+=1 , head = head->next;
        return n;
    }
    ListNode *merge_sort(ListNode *head , int n)
    {
        if(n <= 1) return head;
        int l = n / 2, r = n - l;
        ListNode *p = head,*p1 = head,*p2,new_head;
        for(int i = 1;i < l;i++) p = p->next;
        p2 = p->next;
        p->next = NULL;
        p1 = merge_sort(p1,l);
        p2 = merge_sort(p2,r);
        p = &new_head,new_head.next = NULL;
        while(p1 || p2)
        {
            if(p2 == NULL || (p1 && p1->val < p2->val))
            {
                p->next = p1;
                p = p1;
                p1 = p1->next;
            }
            else
            {
                p->next = p2;
                p = p2;
                p2 = p2->next;
            }
        }
    }
};

```

```

    }
    return new_head.next;
}
ListNode* sortList(ListNode* head) {
    int n = getlength(head);
    return merge_sort(head, n);
}
};

```

□□□□□□□□

- □□sort

```

class Solution {
public:
    int t = 65536;
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        if(n == 0) return;
        for(int i = 0; i < n; i++) nums1[m + i] = nums2[i];
        sort(nums1.begin(), nums1.end());
        return;
    }
};

```

- □□□

```

class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        vector<int> arr;
        arr.resize(nums1.size());
        int p1 = 0, p2 = 0, k = 0;
        while(p1 != m || p2 != n)
        {
            if(p2 == n || (p1 != m && nums1[p1] < nums2[p2]))
            {
                arr[k] = nums1[p1];
                p1++, k++;
            }
            else
            {
                arr[k] = nums2[p2];
                p2++, k++;
            }
        }
        for(int i = 0; i < (m + n); i++)
        {
            nums1[i] = arr[i];
        }
        return ;
    }
};

```



```
    }
};
```

- 时间复杂度
- 空间复杂度 O(1)

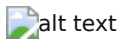
```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int p1 = m - 1, p2 = n - 1, k = m + n - 1;
        while(p1 >= 0 || p2 >= 0)
        {
            if(p2 == -1 || (p1 != -1 && nums1[p1] > nums2[p2]))
            {
                nums1[k] = nums1[p1];
                p1--; k--;
            }
            else
            {
                nums1[k] = nums2[p2];
                p2--; k--;
            }
        }
        return ;
    }
};
```

合并两个有序链表

- 时间复杂度 O(m+n)

```
class Solution { public: ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) { ListNode *p, new_head; p = &new_head; while(list1 || list2) { if(list2 == NULL || (list1 != NULL && list1->val < list2->val)) { p->next = list1; list1 = list1->next; p = p->next; } else { p->next = list2; list2 = list2->next; p = p->next; } } return new_head.next; } };
```

寻找两个正序数组的中位数



```
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        int n = nums1.size(), m = nums2.size();
        vector<int> temp(n + m);
        int p1 = 0, p2 = 0, k = 0;
        while(p1 != n || p2 != m)
        {
            if(p2 == m || (p1 != n && nums1[p1] < nums2[p2]))
```

```

        {
            temp[k] = nums1[p1];
            p1++, k++;
        }
        else
        {
            temp[k] = nums2[p2];
            p2++, k++;
        }
    }
    int t = temp.size() / 2;
    if(temp.size() % 2)
        return temp[t];
    return (temp[t] + temp[t - 1]) / 2.0;
}
};

```

□□□□□□

- □□

```

class Solution {
public:
    bool containsNearbyDuplicate(vector<int>& nums, int k) {
        int p1 = 0, p2 = 1, n = nums.size();
        while (p1 < n) {
            while (p2 < n && (p2 - p1) <= k) {
                if (nums[p1] == nums[p2])
                    return true;
                p2++;
            }
            p1++;
            p2 = p1 + 1;
        }
        return false;
    }
};

```

* □□

```

class Solution {
public:
    bool containsNearbyDuplicate(vector<int>& nums, int k) {
        int n = nums.size();
        vector<int> ind(n);
        for(int i = 0; i < n; i++) ind[i] = i;
        sort(ind.begin(), ind.end(), [&](int i, int j) -> bool
        {
            return nums[i] < nums[j];
        });
        for(int i = 0; i < n - 1; i++)


```



```

    {
        if(nums[ind[i]] != nums[ind[i + 1]]) continue;
        if(abs(ind[i] - ind[i + 1]) <= k) return true;
    }
    return false;
};

```

□□□□□

 alt text

- `aaaa(aaaaaaaaaa)b(aaaaaaaaa),c(aaaaaaaaaa)`
- `aaabaaaaaaaaa`
- `aaabaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`  alt text aaaaaaaaaa
- `p1p2aaaaaaaaa tempaa p2aaaaaaaaa p1aaa aaaa p2aaaaaaaaa`
- `aa tempaaaaaa arraaaaaaaaaa`  alt text

```
#include<bits/stdc++.h>
using namespace std;
int arr[500005],temp[500005];

long long merge_sort(int *arr,int l,int r)
{
    if(r - l <= 1) return 0;
    int mid = (r + l) / 2;
    long long a = merge_sort(arr, l, mid);
    long long b = merge_sort(arr, mid, r);
    long long c = 0;
    int p1 = l, p2 = mid, k = 0;
    while(p1 != mid || p2 != r)
    {
        if(p2 == r || (p1 != mid && arr[p1] <= arr[p2]))
            temp[k++] = arr[p1++];
        else
        {
            temp[k++] = arr[p2++];
            c += (mid - p1);
        }
    }
    for(int i = l;i < r;i++) arr[i] = temp[i - l]; //oooooooooooooooo
    return a + b + c;
}

void solve(int n)
{
    for(int i = 0;i < n;i++) cin >> arr[i];
    cout << merge_sort(arr,0,n) << endl;
    return;
}
```


```


}

int main()
{
    int n;
    while(1)
    {
        cin >> n;
        if(n == 0) break;
        solve(n);
    }
    return 0;
}

```

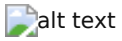
11

 alt text

- □□□□□□y□□□□□□□□□□x□□□□□□□□
- y□□□□□□□□□\$ \$ \sum_{i = 1}^{\{n\}} \{\text{vert } y_i - Y \text{ vert}\} \$ \$
-  alt text
- x□□□□□□□□□\$ \$ \sum_{i = 1}^{\{n\}} \{\text{vert } (x_i- i) - X \text{ vert}\} \$ \$
 - □□□x□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin >> n;
    vector<int> x(n),y(n);
    for(int i = 0;i < n; i++) cin >> x[i] >> y[i];
    int X,Y,costx = 0,costy = 0;
    sort(x.begin(),x.end());
    for(int i = 0; i < n; i++) x[i] = x[i] - i;
    sort(x.begin(),x.end());
    sort(y.begin(),y.end());
    X = x[n / 2];
    Y = y[n / 2];
    for(int i = 0;i < n; i++) costy += abs(y[i] - Y);
    for(int i = 0;i < n; i++) costx += abs(x[i] - X);
    cout << costx + costy << endl;
    return 0;
}
```

□□□□



- `0000100000000000`
- `000C_i 0 C_{i+1} 00000002,0000000000`
- `0000000000000000000000000000200000000010000`

$$\begin{aligned} & \mathbb{P}\{C_{i+1} \leq 1 \mid C_i \leq 1\} \leq \mathbb{P}\{C_{i+1} \leq 1 \mid C_i \leq 1\} \\ & \leq \mathbb{P}\{C_{i+1} \leq 1 \mid C_i \leq 1\} \leq \mathbb{P}\{C_{i+1} \leq 1 \mid C_i \leq 1\} \\ & \leq \mathbb{P}\{C_{i+1} \leq 1 \mid C_i \leq 1\} \leq \mathbb{P}\{C_{i+1} \leq 1 \mid C_i \leq 1\} \end{aligned}$$

0020000000  alt text