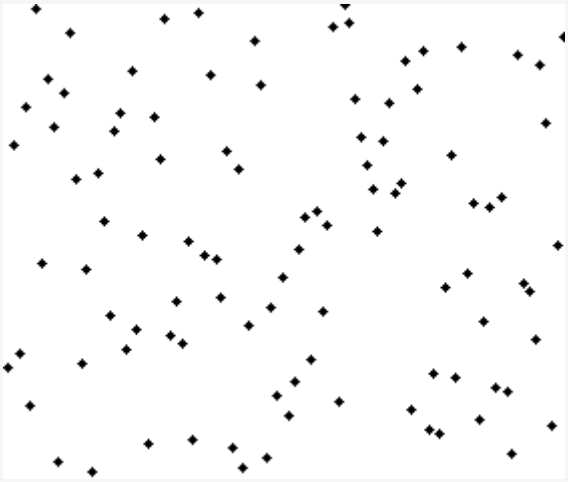
C enum(枚举) C 语言实例 – 约瑟夫生者死者小游戏

C 排序算法

冒泡排序

冒泡排序（英语：Bubble Sort）是一种简单的排序算法。它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序（如从大到小、首字母从A到Z）错误就把他们交换过来。

过程演示：



实例

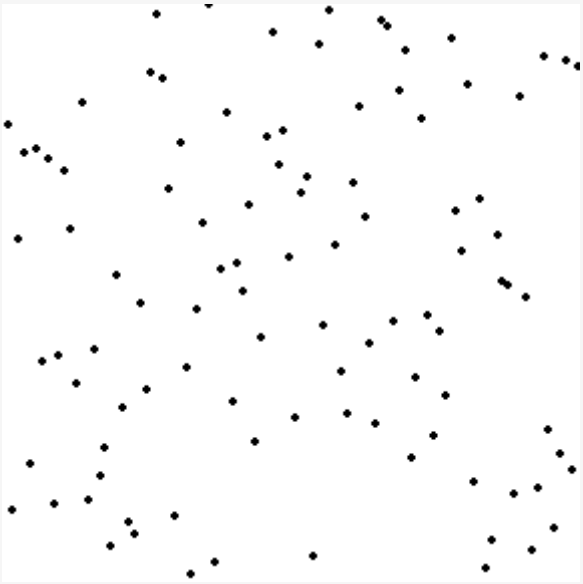
```
#include <stdio.h>
void bubble_sort(int arr[], int len) {
int i, j, temp;
for (i = 0; i < len - 1; i++)
for (j = 0; j < len - 1 - i; j++)
if (arr[j] > arr[j + 1]) {
temp = arr[j];
arr[j] = arr[j + 1];
arr[j + 1] = temp;
}
}
int main() {
int arr[] = { 22, 34, 3, 32, 82, 55, 89, 50, 37, 5, 64, 35, 9, 70 };
int len = (int) sizeof(arr) / sizeof(*arr);
bubble_sort(arr, len);
int i;
for (i = 0; i < len; i++)
printf("%d ", arr[i]);
return 0;
}
```

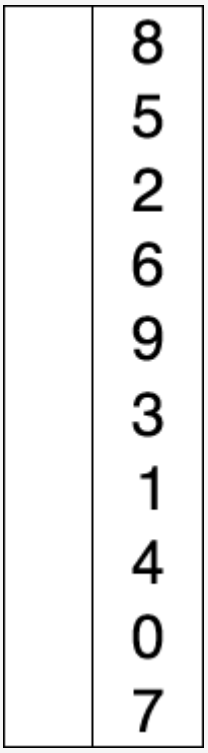
选择排序

选择排序（Selection sort）是一种简单直观的排序算法。它的工作原理如下。首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置，然后，再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。以此类推，直

到所有元素均排序完毕。

过程演示：





实例

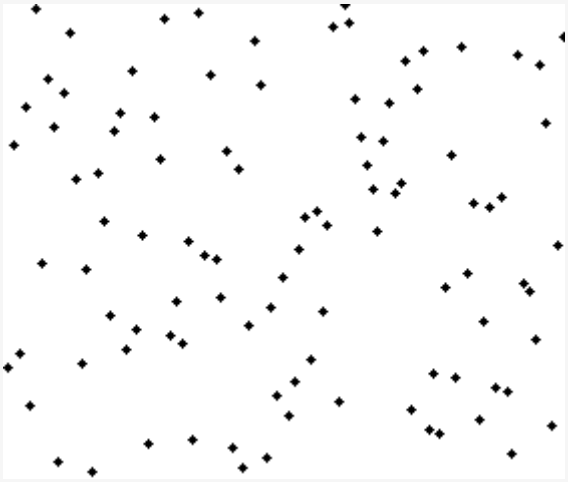
```
void swap(int *a,int *b) //交換兩個變數
{
int temp = *a;
*a = *b;
*b = temp;
}
void selection_sort(int arr[], int len)
{
int i,j;
for (i = 0 ; i < len - 1 ; i++)
{
```

```
int min = i;
for (j = i + 1; j < len; j++) //走訪未排序的元素
if (arr[j] < arr[min]) //找到目前最小值
min = j; //紀錄最小值
swap(&arr[min], &arr[i]); //做交換
}
}
```

插入排序

插入排序（英语：Insertion Sort）是一种简单直观的排序算法。它的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。插入排序在实现上，通常采用in-place排序（即只需用到 {\displaystyle O(1)}{\displaystyle O(1)}的额外空间的排序），因而在从后向前扫描过程中，需要反复把已排序元素逐步向后挪位，为最新元素提供插入空间。

过程演示：



实例

```
void insertion_sort(int arr[], int len){
int i,j,temp;
for (i=1;i<len;i++){
temp = arr[i];
for (j=i;j>0 && arr[j-1]>temp;j--)
arr[j] = arr[j-1];
arr[j] = temp;
}
}
```

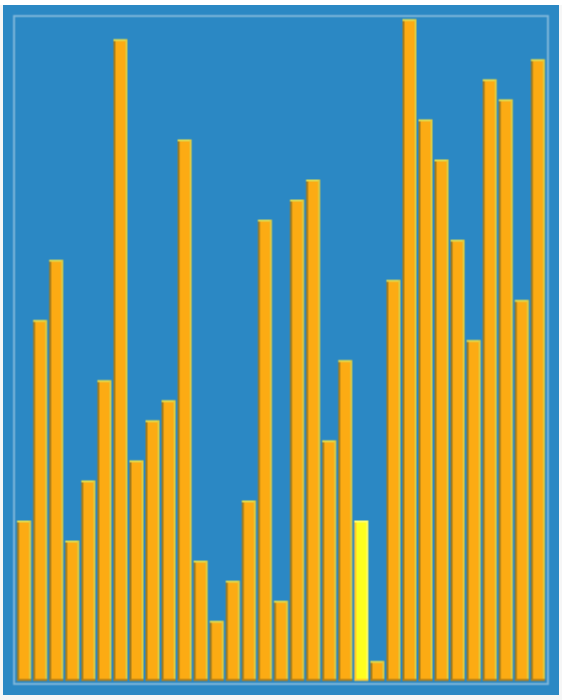
希尔排序

希尔排序，也称递减增量排序算法，是插入排序的一种更高效的改进版本。希尔排序是非稳定排序算法。

希尔排序是基于插入排序的以下两点性质而提出改进方法的：

- 插入排序在对几乎已经排好序的数据操作时，效率高，即可以达到线性排序的效率
- 但插入排序一般来说是低效的，因为插入排序每次只能将数据移动一位

过程演示：



实例

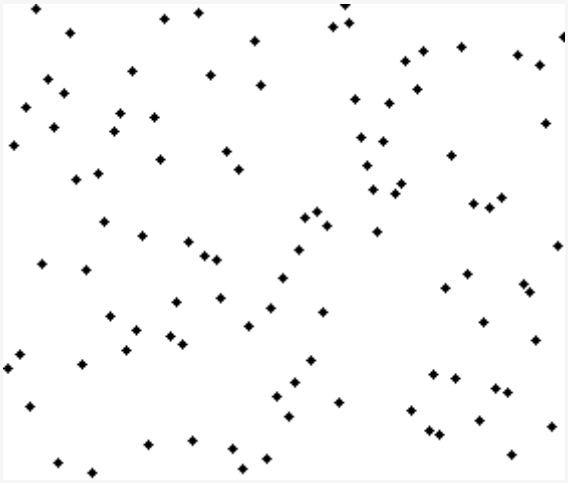
```
void shell_sort(int arr[], int len) {
int gap, i, j;
int temp;
for (gap = len >> 1; gap > 0; gap = gap >>= 1)
for (i = gap; i < len; i++) {
temp = arr[i];
for (j = i - gap; j >= 0 && arr[j] > temp; j -= gap)
arr[j + gap] = arr[j];
arr[j + gap] = temp;
}
}
```

归并排序

把数据分为两段，从两段中逐个选最小的元素移入新数据段的末尾。

可从上到下或从下到上进行。

过程演示：



6 5 3 1 8 7 2 4

迭代法

```
int min(int x, int y) {
return x < y ? x : y;
}
void merge_sort(int arr[], int len) {
int* a = arr;
int* b = (int*) malloc(len * sizeof(int));
int seg, start;
for (seg = 1; seg < len; seg += seg) {
for (start = 0; start < len; start += seg + seg) {
int low = start, mid = min(start + seg, len), high = min(start + seg + seg, len);
int k = low;
int start1 = low, end1 = mid;
int start2 = mid, end2 = high;
while (start1 < end1 && start2 < end2)
b[k++] = a[start1] < a[start2] ? a[start1++] : a[start2++];
while (start1 < end1)
b[k++] = a[start1++];
while (start2 < end2)
b[k++] = a[start2++];
}
int* temp = a;
a = b;
b = temp;
}
if (a != arr) {
int i;
for (i = 0; i < len; i++)
b[i] = a[i];
b = a;
}
free(b);
}
```

递归法

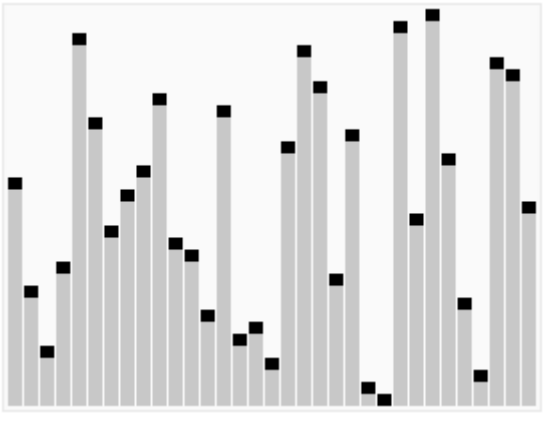
```
void merge_sort_recursive(int arr[], int reg[], int start, int end) {
if (start >= end)
return;
int len = end - start, mid = (len >> 1) + start;
int start1 = start, end1 = mid;
int start2 = mid + 1, end2 = end;
```

```
merge_sort_recursive(arr, reg, start1, end1);
merge_sort_recursive(arr, reg, start2, end2);
int k = start;
while (start1 <= end1 && start2 <= end2)
reg[k++] = arr[start1] < arr[start2] ? arr[start1++] : arr[start2++];
while (start1 <= end1)
reg[k++] = arr[start1++];
while (start2 <= end2)
reg[k++] = arr[start2++];
for (k = start; k <= end; k++)
arr[k] = reg[k];
}
void merge_sort(int arr[], const int len) {
int reg[len];
merge_sort_recursive(arr, reg, 0, len - 1);
}
```

快速排序

在区间中随机挑选一个元素作基准，将小于基准的元素放在基准之前，大于基准的元素放在基准之后，再分别对小数区与大数区进行排序。

过程演示：



迭代法

```
typedef struct _Range {
int start, end;
} Range;
Range new_Range(int s, int e) {
Range r;
r.start = s;
r.end = e;
return r;
}
void swap(int *x, int *y) {
int t = *x;
*x = *y;
*y = t;
}
void quick_sort(int arr[], const int len) {
if (len <= 0)
```

```
return; // 避免len等於負值時引發段錯誤（Segment Fault）
// r[]模擬列表,p為數量,r[p++]為push,r[--p]為pop且取得元素
Range r[len];
int p = 0;
r[p++] = new_Range(0, len - 1);
while (p) {
Range range = r[--p];
if (range.start >= range.end)
continue;
int mid = arr[(range.start + range.end) / 2]; // 選取中間點為基準點
int left = range.start, right = range.end;
do
{
while (arr[left] < mid) ++left; // 檢測基準點左側是否符合要求
while (arr[right] > mid) --right; //檢測基準點右側是否符合要求
if (left <= right)
{
swap(&arr[left],&arr[right]);
left++;right--; // 移動指針以繼續
}
} while (left <= right);
if (range.start < right) r[p++] = new_Range(range.start, right);
if (range.end > left) r[p++] = new_Range(left, range.end);
}
}
```

递归法

```
void swap(int *x, int *y) {
int t = *x;
*x = *y;
*y = t;
}
void quick_sort_recursive(int arr[], int start, int end) {
if (start >= end)
return;
int mid = arr[end];
int left = start, right = end - 1;
while (left < right) {
while (arr[left] < mid && left < right)
left++;
while (arr[right] >= mid && left < right)
right--;
swap(&arr[left], &arr[right]);
}
if (arr[left] >= arr[end])
swap(&arr[left], &arr[end]);
else
left++;
if (left)
quick_sort_recursive(arr, start, left - 1);
quick_sort_recursive(arr, left + 1, end);
}
void quick_sort(int arr[], int len) {
```

```
quick_sort_recursive(arr, 0, len - 1);
}
```

C enum(枚举) C 语言实例 – 约瑟夫生者死者小游戏

2 篇笔记

写笔记

1. 希尔排序缩小递增量必须是要互质的。
2. 快速排序可以不用交换中间值。

以下代码仅供参考：

```
void Array_Map_Sort_Quickly_Extrem(int* Array, int start, int end)
{
    int i=start;
    int j=end;
    int Pivot = Array[end];
    if(start<end)
    {
        while(i<j)
        {
            while(i<j &&Array[i]<=Pivot) i++;//Note: i choose the end as parameter
            Array[j]=Array[i];
            while(i<j &&Array[j]>=Pivot) j--;
            Array[i]=Array[j];
        }
        Array[i]= Pivot;
    }
    else
        return;
    Array_Map_Sort_Quickly_Extrem(Array,start,i-1);
    Array_Map_Sort_Quickly_Extrem(Array,i+1,end);
}
```

唐瓷 3个月前 (12-20)

有种排序叫做猴子排序（Bogo Monkey）：

- 1、检查是否排好
- 2、打乱
- 3、检查是否排好
- 4、打乱
- 5、……

如果数据稍多的话，几乎是不可能排序好的。

排序代码：

```
#include <time.h>
#include <stdlib.h>
#include <stdbool.h>

void swap(int* x, int* y){
  //交换
  int temporary = *x;
  *x = *y;
  *y = temporary;
}
void randomize(int arr[], int length){
  //打乱数组
  for(int i = 0; i < length; i++){
    srand(time(NULL)+i);//引入i增加随机性
    if(rand()%2) swap(&arr[i],&arr[i+1]);
  }
  //printf("!");//记录打乱次数
}
bool isSorted(int arr[], int length){
  for(int i = 0; i < length; i++) if(arr[i]>=arr[i+1]) return false;
  return true;
}
void bogoSort(int array[], int length){
  while(!isSorted(array,length)) randomize(array,length);
}
```

Demo：

```
#include <stdio.h>
int main(){
  int numbers[] = {20,9,233,0,-23,7,1,666,4,345,63,45,2,45};
  bogoSort(numbers,14);//也可以改成更小
  for(int i = 0; i < 14; i++) printf("%d,",numbers[i]);
}
```

学神之女 1个月前 (02-09)