排序算法 C语言版

1. 冒泡排序

```c
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;
            }
}
```

2. 选择排序

```c
void selection_sort(int a[], int len)
{
    int i,j,temp;

    for (i = 0 ; i < len - 1 ; i++)
    {
        int min = i;                  // 记录最小值，第一个元素默认最小
        for (j = i + 1; j < len; j++)     // 访问未排序的元素
        {
            if (a[j] < a[min])    // 找到目前最小值
            {
                min = j;    // 记录最小值
            }
        }
        if(min != i)
        {
            temp=a[min];  // 交换两个变量
            a[min]=a[i];
            a[i]=temp;
        }
        /* swap(&a[min], &a[i]);  */   // 使用自定义函数交换
    }
}

/*
void swap(int *a,int *b) // 交换两个变量
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
*/
```

3. 插入排序

```c
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;
    }
}
```

4. 希尔排序

```c
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;
        }
}
```

5. 归并排序

迭代法

```c
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

```c
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);
}
```

3. 快速排序

迭代法

```c
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等负值时一起段错误（Segment Fault）
    // r[]模拟列表,p为数量,r[p++]为push,r[--p]为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]; // 选取中间元素为基准点
        int left = range.start, right = range.end;
        do
        {
            while (arr[left] < mid) ++left;   // 检测基准点左边的元素是否符合条件
            while (arr[right] > mid) --right; //检测基准点右边的元素是否符合条件

            if (left <= right)
            {
                swap(&arr[left],&arr[right]);
                left++;right--;               // 移动指针继续
            }
        } 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

```c
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)
```

```c
        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);
}
```