排序

快速排序 & 归并排序 & 堆排序 etc.

邵逸帆 *qωq*

23 电信基地班
兰州大学算法与程序设计集训队

2024 年 7 月 15 日

- 排序是将一组数据按照某种顺序重新排列的过程。
- 稳定性: 若两个相等的元素在排序前后的相对位置不变，则称排序算法是稳定的。
- 时间复杂度：简单计算复杂度一般是通过统计“简单操作”的次数来实现的。基于比较的排序算法的时间复杂度的下界是 $O(n \log n)$。
- 空间复杂度：排序算法的空间复杂度是指除了输入数据外，算法运行时所需的额外空间。

八大排序	时间复杂度	空间复杂度	稳定性
冒泡排序	$O(n^2)$	$O(1)$	稳定
选择排序	$O(n^2)$	$O(1)$	不稳定
插入排序	$O(n^2)$	$O(1)$	稳定
希尔排序	$O(n^{\frac{3}{2}})$	$O(1)$	不稳定
归并排序	$O(n\log n)$	$O(n)$	稳定
快速排序	$O(n\log n)$	$O(1)$	不稳定
堆排序	$O(n\log n)$	$O(1)$	不稳定
计数排序	$O(n+k)$	$O(k)$	稳定

Divide and Conquer

分治即“分而治之”，就是把一个复杂的问题分成两个或更多的相同或相似的子问题，直到最后子问题可以简单的直接求解，原问题的解即子问题的解的合并。
分治主要包含三个步骤：

- 分解：将原问题分解成一系列子问题。
- 解决：递归地解决这些子问题。
- 合并：将子问题的解合并成原问题的解。

简单来说就是递归前要做什么 (分解)，递归后要做什么 (合并)。
归并排序和快速排序都是基于分治思想的排序算法。

快速排序

选择一个基准元素 pivot，将数组分成两部分，左边的元素都小于等于 pivot，右边的元素都大于等于 pivot。递归地对左右两部分做快速排序。

划分

- 选择一个基准元素 pivot。
- 用两个指针 $p1, p2$ 分别指向数组的起始位置和结束位置。
- 从 $p1$ 开始向右找到第一个大于等于 pivot 的元素，从 $p2$ 开始向左找到第一个小于等于 pivot 的元素，交换这两个元素。
- 重复上述过程直到 $p1$ 和 $p2$ 相遇。
- 将 pivot 与 $p1$ 指向的元素交换。

Quick Sort

```
void quick_sort(int l, int r) {
  if (l >= r) return;
  int i = l, j = r, x = a[rand() % (r - l + 1) + l];
  while (i <= j) {
    do i++; while (a[i] < x);
    do j--; while (a[j] > x);
    if (i < j) swap(a[i], a[j]);
  }
  quick_sort(l, j), quick_sort(i, r);
}
```

归并排序

- 分解：将数组分成两半。
- 解决：递归地对两半进行归并排序。
- 合并：将两个有序数组合并成一个有序数组。

合并有序数组

- 申请一个临时数组 tmp，大小为 n。
- 用两个指针 $p1, p2$ 分别指向两个有序数组的起始位置。
- 比较 $p1, p2$ 指向的元素，将较小的元素放入 tmp 中。
- 重复上述过程直到两个数组中的元素全部放入 tmp 中。
- 将 tmp 中的元素复制回原数组。

Merge Sort

```
void merge_sort(int l, int r) {
  if (l >= r) return;
  int mid = (l + r) >> 1;
  merge_sort(l, mid), merge_sort(mid + 1, r);
  int i = l, j = mid + 1, k = l;
  while (i <= mid && j <= r) {
    if (a[i] <= a[j]) tmp[k++] = a[i++];
    else tmp[k++] = a[j++];
  }
  while (i <= mid) tmp[k++] = a[i++];
  while (j <= r) tmp[k++] = a[j++];
  for (int i = l; i <= r; i++) a[i] = tmp[i];
}
```

Heap Sort

堆排序

- 建堆：将数组构建成一个大顶堆。
- 排序：将堆顶元素与最后一个元素交换，然后调整堆。

调整堆

- 从根节点开始比较左右子节点的值，将较大的子节点与根节点交换。
- 递归地对交换后的子节点进行调整。

堆排序本质上是一种选择排序。

Algorithm 1 Heap Sort

```
1: BuildHeap()
2: for i = n to 2 do
3:     Swap(a[1], a[i])
4:     AdjustHeap(1, i - 1)
5: end for
```

Algorithm 2 Build Heap

```
1: for i = n/2 to 1 do
2:     AdjustHeap(i, n)
3: end for
```

Algorithm 3 Adjust Heap

```
1:  t = a[i]
2:  for j = 2 × i to n do
3:      if j < n and a[j] < a[j + 1] then
4:          j++
5:      end if
6:      if t ≥ a[j] then
7:          break
8:      end if
9:      a[i] = a[j]
10:     i = j
11: end for
12: a[i] = t
```

THX 4 Listening! :)