作业十

T1

> 设有整型数组 x，试编写算法：将负数集中在数组 x 的一端，正数集中在数组 x 的另一端。要求算法时间复杂性为 $O(n)$

双指针即可，借鉴快排中分治一层的逻辑。

```
void clarify() {
    int n = 10;
    int x[10] = {-1, 4, 2, -3, -9, -10, 3, 5, -6, -7};

    int l = 0, r = n - 1;
    while (l < r) {
        while (x[l] < 0) l++;
        while (x[r] > 0) r--;
        if (l < r) swap(x[l], x[r]);
    }

    for (int i = 0; i < n; i++) {
        cout << x[i] << " \n"[i == n - 1];
    }
}
```

T2

> 请给出快速排序的非递归的算法描述

思路：

- 我们先从递归的角度进行思考，在 $divide$ 时，就是一个简单的双指针进行一次 $partition$，$conquer$ 时就是对前面分出的两个部分进行继续 $partition$ 的过程，其中传递的参数为数组边界。由于在递归操作过程中，采用的类似于“前序遍历”的顺序，因此对于递归的子结构之间是独立相关的，正因为是独立相关的，所以子结构排好序之后，不会影响全局的排序情况且全局排好序！
- 因此栈模拟的思路就有了：我们只需要将每次 $partition$ 的左右边界存入栈即可

```
    void nonRecursiveQuickSort() {
        int n = 10;
```

```cpp
        int x[10] = {-1, 4, 2, -3, -9, -10, 3, 5, -6, -7};

        // partition function
        auto partition = [&](int l, int r) {
            int i = l - 1, j = r + 1, m = x[(l + r) >> 1];
            while (i < j) {
                while (x[++i] < m);
                while (x[--j] > m);
                if (i < j) swap(x[i], x[j]);
            }
            return j;
        };
        stack<pair<int, int>> stk;
        stk.push({0, n - 1});

        while (stk.size()) {
            auto [l, r] = stk.top();
            stk.pop();
            if (l >= r) continue;
            int j = partition(l, r);
            stk.push({l, j});
            stk.push({j + 1, r});
        }

        for (int i = 0; i < n; i++) {
            cout << x[i] << " \n"[i == n - 1];
        }
    }
```

T3

> 已知 $(k_1, k_2, ..., k_n)$ 是堆，试编写算法将 $(k_1, k_2, ..., k_n, k_{n+1})$ 调整为堆

最直接的方法就是无视原始的堆序列，直接对 $n+1$ 个数进行建堆的操作，但是这样的时间复杂度为 $O(n \log n)$ 且没有利用上给的初始堆的条件，我们考虑优化。

优化：为了利用上初始 n 个数为堆的条件，我们考虑插入元素时如何交换元素即可。我们知道，为了使得插入第 $n+1$ 个数之后，序列依然满足堆序列，那么元素比较顺序就一定是固定的，即从堆顶（我们记为 top ）到完全二叉树的最后一个结点的后一个结点（我们记为 $last$ ）。显然完全二叉树中从 top 到 $last$ 的路径是唯一的，且路径上的值是降序的（大顶堆时），因此我们只需要顺序比较 top 到 $last$ 的元素值与新插入的元素的值大小即可，在找到插入位置之后，直接进行下移操作即可。

时间复杂度 $O(\log n)$

```cpp
void insertNum2Heap() {
    int num = 10;
    vector<int> heap = createHeap(num);
    // resize
    int n = heap.size();
    heap.resize(n + 1);

    /* 1. find pos to be inserted */
    int pos = n / 2 - 1;
    while (pos >= 0) {
        if (heap[pos] < num) {
            if (pos % 2) pos = (pos - 1) / 2;
            else pos = (pos - 2) / 2;
        } else {
            break;
        }
    }

    /* 2. move path down */
    int i = n / 2 - 1, last = n;
    while (i >= pos) {
        if (i == pos) {
            heap[last] = num;
            break;
        }
        heap[last] = heap[i];
        last = i;
        if (i % 2) {
            i = (i - 1) / 2;
        } else {
            i = (i - 2) / 2;
        }
    }
}
```

T4

> 给定有序序列 `A[m]` 和有序序列 `B[n]`，试编写算法将它们归并为一个有序序列 `C[m+n]`

双指针 $combine$ 即可。时间复杂度 $O(n+m)$

```cpp
void mergeTwoOrderedSeq() {
    vector<int> a = {1, 3, 5, 7, 9};
    vector<int> b = {2, 4, 6, 8, 10};
    vector<int> res(a.size() + b.size());
    int i = 0, j = 0, k = 0;
    while (i < a.size() && j < b.size()) {
        if (a[i] < b[j]) res[k++] = a[i++];
        else res[k++] = b[j++];
    }

    while (i < a.size()) res[k++] = a[i++];
    while (j < b.size()) res[k++] = b[j++];

    for (int x: res) {
        cout << x << " ";
    }
}
```

T5

> 一个序列中的逆序对是这样的两个元素，对于序列 A 而言，$i>j$ 且 $A[i]<A[j]$，于是 $A[i]$ 和 $A[j]$ 就形成一个逆序对。设计有一个有效的算法统计一个给定的数组 A 中的逆序对的个数，并给出该算法的时间复杂度。

首先很容易想到一个 $O(n^2)$ 的做法，就是直接双重循环统计当前元素后面比自己小的元素的个数即可。但是可以利用归并排序中分支后归并的过程进行优化。

优化：

- 利用归并排序 $combine$ 过程中比大小的情况来统计逆序数
- 在采用分治法进行归并排序的过程中我们知道，需要统计的逆序数分为三部分，分别为
  - 左半部分的逆序数
  - 右半部分的逆序数
  - 左右之间的逆序数

- 我们知道上述三种情况是相互独立的，不会互相影响，因此我们分治法计算逆序数是可行的。对于当前局面，左右两部分都是有序的，因此在比较的过程中，如果遇到左边部分的某个元素比右边部分的某个元素大，那么左边部分剩余的元素都会比右边部分当前元素大，于是 `res += mid-i+1` 就得出来了，那么对于当前局面统计逆序数的时间开销就是 $O(n)$ 的。由于总的局面数是 $O(\log n)$ 级别的，因此：
- 时间复杂度 $O(n \log n)$

```
void countReverseOrder() {
    vector<int> a = {1, 3, 5, 2, 4, 6};
    int cnt = 0;

    // merge sort
    function<void(int, int)> mergeSort = [&](int l, int r) {
        if (l >= r) return;

        // divide
        int mid = (l + r) >> 1;

        // conquer
        mergeSort(l, mid), mergeSort(mid + 1, r);

        // combine
        int t[a.size()], i = l, j = mid + 1, k = 0;
        while (i <= mid && j <= r) {
            if (a[i] <= a[j]) t[k++] = a[i++];
            else {
                t[k++] = a[j++];
                cnt += mid - i + 1;     // count
            }
        }
        while (i <= mid) t[k++] = a[i++];
        while (j <= r) t[k++] = a[j++];

        for (i = l, k = 0; i <= r; i++) a[i] = t[k++];
    };

    mergeSort(0, a.size() - 1);

    cout << cnt << "\n";
}
```

实验十

实验代码：https://github.com/Explorer-Dong/DataStructure/blob/main/Code/chapter10_Sort/Experiment_10.cpp

上机实验题 10

实验题　编写程序，实现插入排序、快速排序、堆排序、归并排序。

基本要求：

（1）利用随机函数产生 1 000 个随机整数，作为待排序序列。

（2）要求程序模块化结构，一个函数实现一种排序算法。

（3）记录各种算法的排序过程中数据的移动次数，并进行比较。

在 10^3 量级的排序下，各排序算法的结果如下：

算法	移动次数
插入排序	257563
快速排序	2539
堆排序	16486
归并排序	8724