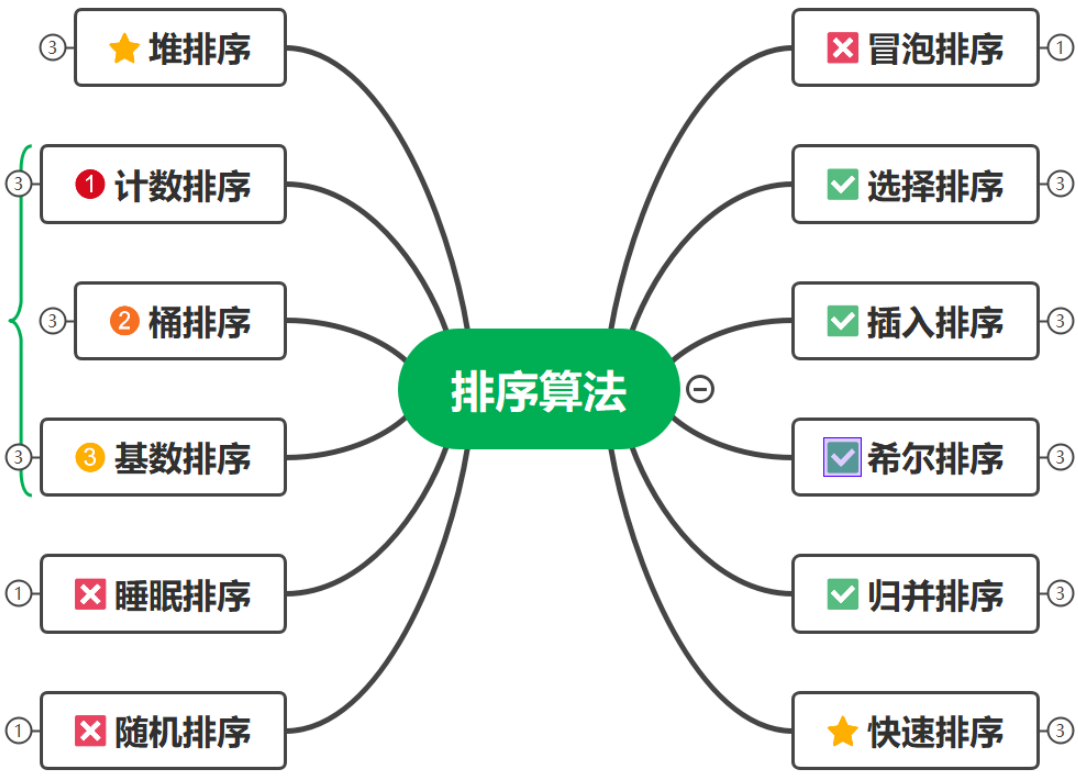
代码小抄：十大排序算法（C++实现）

整体把握



一、冒泡排序

```cpp
/*
 * 冒泡排序
 * 比较交换相邻元素
 * 思路简单，仅学习用
 */
class Bubble
{
public:
    void mysort(vector<int> &nums)
    {
        int n = nums.size();
        for (int i = 0; i < n - 1; i++)
        {
            for (int j = 0; j < n - i - 1; j++)
            {
                if (nums[j] > nums[j + 1])
                {
                    int temp = nums[j];
                    nums[j] = nums[j + 1];
                    nums[j + 1] = temp;
                }
            }
        }
    }
};

int main()
{
    vector<int> nums = {9, 7, 5, 3, 1, 0, 8, 4, 6, 2};
    Bubble bubble;
    bubble.mysort(nums);
    for (int num : nums)
    {
        cout << num << " ";
    }
    return 0;
}
```

二、选择排序

```cpp
/*
 * 选择排序
 * 思路：找最小元素，放到最终位置
 * 特点：时间：O(n²)、非稳定排序
 * 适用：数据量少
 */
class Select
{
public:
    void mysort(vector<int> &nums)
    {
        int n = nums.size();
        for (int i = 0; i < n - 1; i++)
        {
            int minPos = i;
            for (int j = i + 1; j < n; j++)
            {
                if (nums[j] < nums[minPos])
                {
                    minPos = j;
                }
            }
            int temp = nums[i];
            nums[i] = nums[minPos];
            nums[minPos] = temp;
        }
    }
};

int main()
{
    vector<int> nums = {9, 7, 5, 3, 1, 0, 8, 4, 6, 2};
    Select select;
    select.mysort(nums);
    for (int num : nums)
    {
        cout << num << " ";
    }
    return 0;
}
```

三、插入排序

```cpp
/*
 * 插入排序
 * 思路：抓牌一样，插入当前手牌中的适当位置
 * 特点：时间：O(n²)
 * 适用：基本有序
 */
class Insert
{
public:
    void mysort(vector<int> &nums)
    {
        int n = nums.size();
        for (int i = 1; i < n; i++)
        {
            int pos = i - 1;
            int cur = nums[i];
            while (pos >= 0 && cur < nums[pos])
            {
                nums[pos + 1] = nums[pos];
                pos--;
            }
            nums[pos + 1] = cur;
        }
    }
};

int main()
{
    vector<int> nums = {9, 7, 5, 3, 1, 0, 8, 4, 6, 2};
    Insert insert;
    insert.mysort(nums);
    for (int num : nums)
    {
        cout << num << " ";
    }
    return 0;
}
```

四、希尔排序

```cpp
/*
 * 希尔排序
 * 思路：间隔分组+自定义排序（这里给出的是冒泡）
 * 特点：时间：O(nlogn)、非稳定排序
 * 适用：数据量大
 */
class Shell
{
public:
    void mysort(vector<int> &nums)
    {
        int n = nums.size();
        int gap = n / 2;
        while (gap > 0)
        {
            for (int j = gap; j < n; j++)
            {
                int i = j;
                while (i >= gap && nums[i] < nums[i - gap])
                {
                    int temp = nums[i];
                    nums[i] = nums[i - gap];
                    nums[i - gap] = temp;

                    i -= gap;
                }
            }
            gap /= 2;
        }
    }
};

int main()
{
    vector<int> nums = {9, 7, 5, 3, 1, 0, 8, 4, 6, 2};
    Shell shell;
    shell.mysort(nums);
    for (int num : nums)
    {
        cout << num << " ";
    }
    return 0;
}
```

五、归并排序

```cpp
/*
 * 归并排序
 * 思路：递归思想
 * 特点：时间：O(nlogn)、空间：O(n)--非原地
 * 适用：不受数据影响，所需空间与n成正比
 */
class Merge
{
public:
    void mysort(vector<int> &nums, int left, int right)
    {
        if (left < right)
        {
            int mid = left + (right - left) / 2;
            mysort(nums, left, mid);
            mysort(nums, mid + 1, right);
            merge(nums, left, right);
        }
        return;
    }

    void merge(vector<int> &nums, int left, int right)
    {
        vector<int> temp(nums.size());
        int mid = left + (right - left) / 2;
        int p = left;
        int q = mid + 1;
        int k = left;
        while (p <= mid && q <= right)
        {
            if (nums[p] < nums[q])
                temp[k++] = nums[p++];
            else
                temp[k++] = nums[q++];
        }
        while (p <= mid)
            temp[k++] = nums[p++];
        while (q <= right)
            temp[k++] = nums[q++];
        for (int i = left; i <= right; i++)
            nums[i] = temp[i];
    }
};

int main()
{
    vector<int> nums = {9, 7, 5, 3, 1, 0, 8, 4, 6, 2};
    Merge merge;
    merge.mysort(nums, 0, nums.size() - 1);
    for (int num : nums)
        cout << num << " ";
    return 0;
}
```

六、快速排序

```cpp
/*
 * 快速排序
 * 思路：选择中轴元素，比它小的放左，比它大的放右（代码过程很像 小丑在扔三个小球）
 * 特点：时间：O(nlogn)、空间：O(logn)、非稳定
 * 适用：广泛（最快）
 */
class Quick
{
public:
    void mysort(vector<int> &nums, int start, int end)
    {
        if (start >= end)
            return;
        int left = start;
        int right = end;
        int temp = nums[left];
        while (left < right)
        {
            while (left < right && nums[right] > temp)
                right--;
            nums[left] = nums[right];
            while (left < right && nums[left] < temp)
                left++;
            nums[right] = nums[left];
        }
        nums[left] = temp;
        mysort(nums, start, left - 1);
        mysort(nums, left + 1, end);
    }
};

int main()
{
    vector<int> nums = {9, 7, 5, 3, 1, 0, 8, 4, 6, 2};
    Quick quick;
    quick.mysort(nums, 0, nums.size() - 1);
    for (int num : nums)
    {
        cout << num << " ";
    }
    return 0;
}
```

七、堆排序

```
/*
 * 堆排序
 * 思路：升序用大顶堆，每次调整后把最大的移出，再调整...
 * 特点：时间：O(nlogn)、非稳定
 * 适用：数据结构学习
 */
class Heap
{
public:
    void mysort(vector<int> &nums)
    {
        int n = nums.size();
        // （1）构造初始堆
        // 从第一个非叶子节点（倒数第二行最后一个）开始调整
        // 左右孩子节点中较大的交换到父节点中
        // 注意这里i是自底往上的!
        for (int i = n / 2 - 1; i >= 0; i--)
        {
            headAdjust(nums, n, i);
        }
        // （2）排序
        // 第一步.交换list[0]（最大）和list[i]
        // 第二步.此时list[0]在堆底，固定住，已排好
        //   （for循环的i代表len，i--即每次把最后一个排好的忽略掉）
        // 第三步.把无缘无故提上来的幸运儿list[i]再adjust回它应该在的位置
        //   （像石头沉入水底）
        // 下一个循环
        for (int i = n - 1; i > 0; i--)
        {
            //交换
            int temp = nums[i];
            nums[i] = nums[0];
            nums[0] = temp;
            //调整
            headAdjust(nums, i, 0);
        }
    }

    // 辅助函数：调整堆
    // 参数说明：nums代表整个二叉树、len是nums的长度、 i代表三个中的根节点
    void headAdjust(vector<int> &nums, int len, int i)
    {
        int index = 2 * i + 1;

        // 这步while的意义在于把较小的沉下去，把较大的提上来
        while (index < len)
        {
            // （1）index指向左右孩子较大的那个
            if (index + 1 < len)
            {
                if (nums[index + 1] > nums[index]) // 说明还有右孩子
                {
                    index = index + 1;
                }
```

```
            }
            // （2）比较交换大孩子和根节点
            if (nums[index] > nums[i])
            {
                //交换
                int temp = nums[i];
                nums[i] = nums[index];
                nums[index] = temp;
                //更新
                i = index;
                index = 2 * i + 1;
            }
            else
            {
                break;
            }
        }
    }
};

int main()
{
    vector<int> nums = {9, 7, 5, 3, 1, 0, 8, 4, 6, 2};
    Heap heap;
    heap.mysort(nums);
    for (int num : nums)
    {
        cout << num << " ";
    }
    return 0;
}
```

八、计数排序

```cpp
/*
 * 计数排序
 * 思路：借助足够大的辅助数组，把数字排在一个相对位置不会错的地方，最后并拢
 * 特点：时间：O(n+k)、空间：O(n+k)--非原地
 * 适用：max和min的差值不大
 */
class Count
{
public:
    void mysort(vector<int> &nums, int min, int max)
    {
        vector<int> temp(max - min + 1);

        for (int num : nums)
        {
            temp[num - min]++;
        }

        int index = 0;
        for (int i = 0; i < temp.size(); i++)
        {
            int cnt = temp[i];
            while (cnt != 0)
            {
                nums[index] = i + min;
                index++;
                cnt--;
            }
        }
    }
};

int main()
{
    vector<int> nums = {9, 7, 5, 3, 1, 0, 8, 4, 6, 2};
    Count count;
    count.mysort(nums, 0, 9);
    for (int num : nums)
    {
        cout << num << " ";
    }
    return 0;
}
```

九、桶排序

```
/*
 * 桶排序
 * 思路：先粗略分类分桶，再各桶排序
 * 特点：时间：O(n+k)、空间：O(n+k)--非原地
 * 适用：均匀分布的数据
 */
class Bucket
{
public:
    void mysort(vector<int> &nums)
    {
        //（1）初始化桶
        int n = nums.size();
        vector<list<int>> buckets(n);

        //（2）数据放入桶并完成排序
        for (int num : nums)
        {
            int index = getBucketIndex(num);
            insertSort(buckets[index], num);
        }

        //（3）从桶取数据，放入nums
        int index = 0;
        for (list<int> bucket : buckets)
        {
            for (int num : bucket)
            {
                nums[index] = num;
                index++;
            }
        }
    }

    //辅助函数一：获得桶的序号
    int getBucketIndex(int num)
    {
        return num / 3;
    }

    //辅助函数二：把数据插入对应桶(这里用的插入排序)
    void insertSort(list<int> &bucket, int num)
    {
        int n = bucket.size();
        bool flag = true;
        for (auto it = bucket.begin(); it != bucket.end(); it++)
        {
            if (num <= *it)
            {
                bucket.insert(it, num);
                flag = false;
                break;
            }
        }
```

```
        if (flag)
            bucket.push_back(num);
    }
};

int main()
{
    vector<int> nums = {9, 7, 5, 3, 1, 0, 8, 4, 6, 2};
    Bucket b;
    b.mysort(nums);
    for (int num : nums)
    {
        cout << num << " ";
    }
    return 0;
}
```

> 注：`桶排序`的`C++实现`我磕磕绊绊，其中遇到了以下困难，希望以后注意：
>
> 1.`vector<list<int>> buckets(n);`的数据结构照顾到频繁插入
>
> 2.`list`结构无法根据索引直接获取值，所以需要`迭代器`遍历
>
> 3.获取桶序号的函数`return num / 3;`，一定要确保大小顺序。比如一开始我用的`num % 3`导致大小错开了。
>
> 4.`insertSort`函数里的`flag`判断很精妙，尾插必不可少

十、基数排序

```
/*
 * 基数排序
 * 思路：桶排序的一种。  按数位分桶：从低位开始 -> 分桶、收集 -> 下一位...
 * 特点：时间：O(kn)、空间：O(n+k)--非原地
 * 适用：max和min的差值不大
 */
class Radix
{
public:
    void mysort(vector<int> &nums, int d)
    {
        int p = 1;
        int n = nums.size();
        vector<vector<int>> buckets(10, vector<int>(n));
        vector<int> order(10);

        while (p < d)
        {
            //（1）进行一轮分桶
            for (int num : nums)
            {
                int index = num / p % 10;            //获取桶号
                buckets[index][order[index]] = num; // num放入index号桶的第
order[index]位置
                order[index]++;                      //位置++
            }
            //（2）进行一轮排序
            int k = 0;
            for (int i = 0; i < 10; i++)
            {
                if (order[i] == 0)      continue;
                for (int j = 0; j < order[i]; j++)
                {
                    nums[k] = buckets[i][j];
                    k++;
                }
                order[i] = 0; //各桶计数器清零
            }
            p *= 10;
        }
    }
};

int main()
{
    vector<int> nums = {999, 765, 780, 215, 13, 66, 230, 450, 699, 21};
    Radix radix;
    radix.mysort(nums, 1000);
    for (int num : nums)
        cout << num << " ";
    return 0;
}
```

EX1、睡眠排序（Java）

```java
/*
 * 睡眠排序
 * 思路：娱乐算法：睡醒了起来报数
 * 注意：java实现，因为c++没有多线程
 */
public class Sleep {
    public static void sleepSort(int[] array) {
        for (int i : array) {
            new Thread(() -> {
                try {
                    Thread.sleep(i);
                } catch (Exception e) {
                    e.printStackTrace();
                }
                System.out.println(i);
            }).start();
        }
    }

    // 测试代码
    public static void main(String[] args) {
        int[] array = { 10, 30, 50, 60, 100, 40, 150, 200, 70 };
        sleepSort(array);
    }
}
```

EX2、随机排序（Java）

```java
/*
 * 随机排序
 * 思路：碰运气瞎排，再看蒙对没有
 */
public class Random {
    public static void randSortX(int[] array) {
        List<Integer> list = new ArrayList<>();
        for (Integer integer : array) {
            list.add(integer);
        }
        int pre = 0;
        int index = 0;
        while (true) {
            pre = 0;
            for (index = 1; index < list.size(); index++) {
                if (list.get(index) > list.get(pre)) {
                    pre++;
                } else {
                    break;
                }
            }
            if (pre + 1 == list.size()) {
                break;
            }
            Collections.shuffle(list);
        }
        System.out.println(list.toString());
    }

    // 测试代码
    public static void main(String[] args) {
        int[] array = { 10, 30, 50, 60, 100, 40, 150, 200, 70 };
        randSortX(array);
    }
}
```