面试问题整理

数据结构与算法

递归

- 大问题化为小问题,可以极大的减少代码量；
- 代码更简洁清晰，可读性更好
- 递归调用函数,佔用大量空间,递归太深容易造成堆栈的溢出；

1. 冒泡排序
2. 选择排序：选择排序与冒泡排序有点像，只不过选择排序每次都是在确定了最小数的下标之后再进行交换，大大减少了交换的次数
3. 插入排序：将一个记录插入到已排序的有序表中，从而得到一个新的，记录数增 1 的有序表
4. 快速排序：通过一趟排序将序列分成左右两部分，其中左半部分的的值均比右半部分的值小，然后再分别对左右部分的记录进行排序，直到整个序列有序。

- 在堆排中，每一个操作都是不利于程序的局部性原理的，每次元素间的比较、数的调整等，都不是相邻或者尽可能附近的元素间的比较(堆调整每次都从堆底拿元素到堆顶然后向下进行调整），那么这就需要不断地在磁盘和内存间换入换出数据。反观快排，利用分而治之的方法，元素间的比较都在某个段内，局部性相当好

```
def partition(arr,low,high):
    i = ( low-1 )         # index of smaller element
    pivot = arr[high]     # pivot
    for j in range(low , high):
        # If current element is smaller than or equal to pivot
        if   arr[j] <= pivot:
            # increment index of smaller element
            i = i+1
            arr[i],arr[j] = arr[j],arr[i]
    arr[i+1],arr[high] = arr[high],arr[i+1]
    return i+1

def quickSort(arr,low,high):
    if low < high:
        # pi is partitioning index, arr[p] is nowat right place
        pi = partition(arr,low,high)
        # Separately sort elements before partition and after partition
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)
```

backtracking

```
result = []
def backtrack(路径, 选择列表):
```

```
    if 满足结束条件:
        result.add(路径)
        return

    for 选择 in 选择列表:
        做选择
        backtrack(路径, 选择列表)
        撤销选择
```

sliding window

```
int left = 0, right = 0;
while (right < s.size()) {
    window.add(s[right]); 哈希表充当计数器
    right++; 擴大搜索window
    while (window 中的字符串已符合 needs 的要求了) {
        window.remove(s[left]);
        left++;左指針，減少搜索window
    }
}
```

树

- 树是图的子集
- 树有一个根节点，图没有
- 树可以递归遍历，图要看情况
- 树有层次划分，图没有
- 树的非根节点必定有一个父节点，图不一定

完全二叉树

- 若设二叉树的高度为 h，除第底层外，其它各层 (1 ～ h-1) 的结点数都达到最大个.这就是完全二叉树。

平衡二叉树

- 左子树和右子树的高度差不超过 1 时，这个树是平衡二叉树。