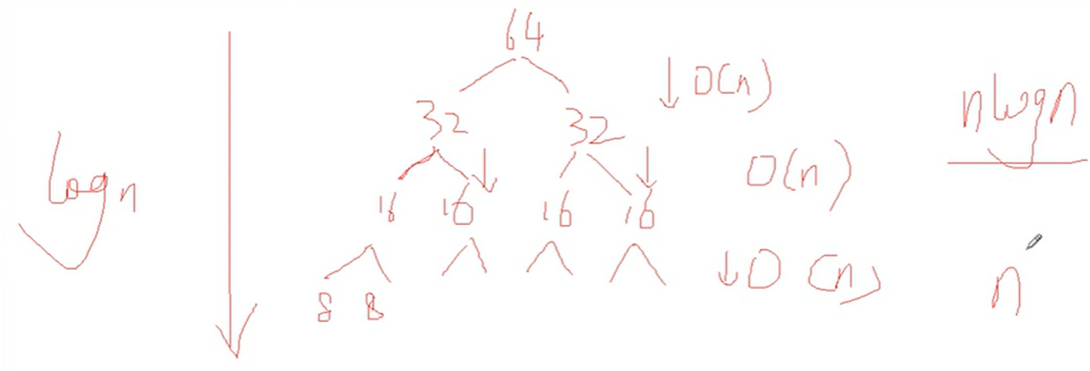
3.3 快速排序

- 快速排序：快
- 思路：
  - 取一个元素p（第一个元素），使元素p归位；
  - 列表被p分成两部分，左边都比p小，右边都比p大；
  - 递归完成排序。

```
def quick_sort(li, left, right):
    if left < right:
        mid = partition(li,left,right)
        quick_sort(li,left,mid-1)
        quick_sort(li,mid+1,right)


def partition(li,left,right):
    tmp = li[left]
    while left < right:
        while left < right and li[right] >= tmp:
            right -= 1
        li[left] = li[right]
        while left < right and li[left] <= tmp:
            left += 1
        li[right] = li[left]
    li[left] = tmp
    return left
```

3.3.1 时间复杂度（不严谨推理）



假设我们有64个数要排序，总共需要log(n)次，每次需要n，所以最好时间复杂度为O(nlogn).

如果是倒序排列，每次只能递归一侧，则最坏时间复杂度为O(n^2)。

总结：

最好情况O(nlogn)

平均情况O(nlogn)

最坏情况O(n^2)

3.3.2 优化（改善倒序情况）

```
def quick_sort(li, left, right):
    if left < right:
        mid = random_partition(li,left,right)
        quick_sort(li,left,mid-1)
        quick_sort(li,mid+1,right)


def partition(li,left,right):
    tmp = li[left]
    while left < right:
        while left < right and li[right] >= tmp:
            right -= 1
        li[left] = li[right]
        while left < right and li[left] <= tmp:
            left += 1
        li[right] = li[left]
    li[left] = tmp
    return left

def random_partition(li,left,right):
    i = random.randint(left,right)
    li[i], li[left] =li[left],li[i]
    return partition(li,left,right)
```

增加了随机性，取得第一个数与剩下的数随机交换位置，使得第一个数有随机性，不再是倒序。其实是将可能倒序的情况时候需要消耗的更多时间，平均到各种情况的可能性里面去了。

最坏时间复杂度不变：O(n^2)。

3.3.3 另一种版本代码

- 方便理解但是占用空间

```
def quick_sort2(li):
    if len(li) < 2:
        return li
    tmp = li[0]
    left = [v for v in li[1:] if v <= tmp]
    right = [v for v in li[1:] if v > tmp]
    left = quick_sort2(left)
    right = quick_sort2(right)
    return left + [tmp] + right
```