问题 1

对于下面的每一对表达式(A, B), A是否能表示为B的 Θ, Ω, O形式. 请注意, 这些关系中的零个、一个或多个可能成立。列出所有正确的。经常发生一些学生会,把指示写错, 所以请把关系写完整, 例如: $A = O(B), A = \Theta(B)$, 或$A = \Omega(B)$。

1. $A = n^2 - 100n, B = n^2$
2. $A = logn, B = log_{1.2}n$
3. $A = 3^{2n}, B = 2^{4n}$
4. $A = 2^{logn}, B = n$
5. $A = \log \log n, B = 10^{10^{100}}$

1. $A=O(B)$, $A=\Theta(B)$, $A=\Omega(B)$
2. $A=O(B)$, $A=\Theta(B)$, $A=\Omega(B)$
3. $A=O(B)$
4. $A=O(B)$, $A=\Theta(B)$, $A=\Omega(B)$
5. $A=O(B)$

问题 2:

假设有函数 f 和 g 使得 $f(n) = O(g(n))$ 对于下面的每一个陈述, 请判断对错, 如果正确请给出证明, 否则请给出一个反例。

1. $\log f(n) = O(\log(1 + g(n)))$
2. $3^{f(n)} = O(3^{g(n)})$
3. $(f(n))^2 = O((g(n))^2)$

1. $\log f(n) = O(\log(1+g(n)))$ 正确

证明: $\because f(n) = O(g(n))$

$\therefore \exists c>0$, 有 $f(n) \le c \cdot g(n) \Rightarrow \log f(n) \le \log(c \cdot g(n)) = \log c + \log(g(n))$

又$\because \log g(n) \le \log(1+g(n))$

$\therefore \log f(n) \le \log c + \log(1+g(n)) \in O(\log(1+\log(n)))$

即证得: $\log f(n) = O(\log(1+g(n)))$

2. $3^{f(n)} = O(3^{g(n)})$ 错误

反例: $f(n) = 100 \cdot n$

$g(n) = n$

显然 $f(n) = O(g(n))$

但: $3^{f(n)} = 3^{100n} = (3^n)^{100}$, 而 $3^{g(n)} = 3^n$

$\Downarrow$

$3^{f(n)} \gg 3^{g(n)}$

3. $(f(n))^2 = O((g(n))^2)$ 正确

证明: $\because f(n) = O(g(n))$

$\therefore \exists c>0$, 有 $f(n) \le c \cdot g(n)$

两边同时平方得:

$f(n)^2 \le c^2 \cdot g(n)^2 \Rightarrow f(n)^2 = O(g(n)^2)$

即证得: $(f(n))^2 = O((g(n))^2)$

问题 3

根据下列递归公式, 计算下列 $T(n)$ 对应的的渐近上界。要求所求的边界尽可能的紧（tight）, 请写明步骤。

1. $T(1) = 1; T(n) = T(n/4) + 1$ for $n > 1$
2. $T(1) = 1; T(n) = 3T(n/3) + n^2$ for $n > 1$
3. $T(1) = 1; T(n) = T(2n/3) + 1$ for $n > 1$
4. $T(1) = 1; T(n) = 5T(n/4) + n$ for $n > 1$
5. $T(n) = 1$ for $n \le 2; T(n) = T(\sqrt{n}) + 1$ for $n > 2$

1. $T(n)=T(\frac{n}{4})+1$

$=T(\frac{n}{16})+1+1$

$\vdots$

$=T(\frac{n}{4^k})+k$

令 $\frac{n}{4^k}=1$，则 $t=\log_4 n$

$\therefore T(n)=O(1+\log_4 n)$

$=O(\log n)$

2. $T(n)=3T(\frac{n}{3})+n^2$

$a=3,\ b=3,\ f(n)=n^2$

$n^{\log_b a}=n^{\log_3 3}=n$

$f(n)=n^2=\Omega(n^{1+\varepsilon})\quad \varepsilon=1$

$\therefore T(n)=O(f(n))=O(n^2)$

3. $T(n)=T(\frac{2n}{3})+1$

$=T((\frac{2}{3})^2 n)+2$

$\vdots$

$=T((\frac{2}{3})^k n)+k$

令 $(\frac{2}{3})^k n=1$，则 $k=\log_{\frac{3}{2}} n$

$\therefore T(n)=O(\log n)$

4. $T(n)=5T(\frac{n}{4})+n$

$a=5,\ b=4,\ f(n)=n$

$n^{\log_b a}=n^{\log_4 5}\approx n^{1.16}$

$f(n)=n=O(n^{\log_4 5})$

$\therefore T(n)=O(n^{\log_4 5})$

5. $T(n)=T(\sqrt{n})+1$

令 $n_k=\sqrt{n_{k-1}}$

$T(n)=T(\sqrt{n})+1$

$=T(n^{\frac{1}{2^k}})+k$

令 $n^{\frac{1}{2^k}}=2$，则 $k=\log\log n$

$\therefore T(n)=O(\log\log n)$

问题 4：

给定一个包含n个元素的数组 `profits` ，它的第 `i` 个元素 `profits[i]` 表示一支股票第 `i` 天的**收益**（正数表示涨，负数表示跌）。你只能选择 **某一天** 买入这只股票，并选择在 **未来的某一个不同的日子** 卖出该股票。

1. 设计一个算法来计算你所能获取的最大利润和对应买入和卖出的日期。请分析算法方案，计算其时间复杂度，并且使用python编程实现该算法。
2. * 设计一个时间复杂度为 $O(n)$的算法实现该算法

思路：MaxSubarray_bruteForce(A)

```
        Vmax-A[1]
        for i⟵1 to n:
            V-0
            for j⟵i to n
                V=V+A[j]
                if V >Vmax then V max=v
        return V max
```

时间复杂度：$O(n^2)$

问题 5：

观察下方的分治算法（divide-and-conquer algorithm）的伪代码，回答下面问题

```
DoSomething(A, p, r)
-----
n := r-p+1
if n=2 and A[p]>A[r] then
    swap A[p] and A[r]
else if n >= 3 then
    m = ceil(2n/3)
    DoSomething(A, p, p+m-1)
    DoSomething(A, r-m+1, r)
    DoSomething(A, p, p+m-1)

---
first call: DoSomething(A, 1, n)
```

note：$ceil(2n/3)=\lceil 2n/3\rceil$；:= 表示赋值，等价于 →；A是一个包含n的整数元素的数组，

1. $T(n) = 3T(\frac{2}{3}n) + 1$

$a = 3,\ b = \cancel{\frac{2}{3}}\ \frac{3}{2},\ f(n) = 1$

$\log_b a = \log_{\frac{3}{2}} 3 = \frac{\log 3}{\log(\frac{3}{2})} \approx 2.71$

$\therefore T(n) = O(n^{\log_{\frac{3}{2}} 3}) = O(n^{2.71})$

2. 该算法是**对数组进行排序**，具体逻辑如下，如果当前区间大小为 2，且顺序错误就交换，否则将数组拆为三段，递归排序这些子数组。该算法不是最优算法，快排的时间复杂度是 O(nlogn)，效率更高。

问题 6：

给定一个大小为 `n` 的数组 `nums`，返回其中的多数元素。多数元素是指在数组中出现次数 **大于** ⌊ `n/2` ⌋ 的元素。

你可以假设数组是非空的，并且给定的数组总是存在多数元素。

1. 设计一个算法找到给定数组的多数元素，分析算法设计思路，计算算法时间复杂度，使用python编程实现
2. * 设计时间复杂度为 O(n)、空间复杂度为 O(1) 的算法解决此问题，分析算法设计思路，使用python编程实现

1. majority_element hash(nums)

```
counts← 新建一个空的哈希表
for num in nums
   if num 存在于 counts 中
      counts[num]←counts[num]+ 1
   else
      counts[num]←1
   if counts[num]>数组长度 nums/2
      return num
```

问题 7：

给定一个包含不同整数元素的数组 $A[1..n]$，并且满足条件：$A[1] > A[2]$ 并且 $A[n-1] < A[n]$；规定：如果一个元素比它两边的邻居元素都小，即：$A[x] < A[x-1], A[x] < A[x+1]$，称这个元素A[x]为"局部最小"。通过遍历一次数组，我们可以很容易在 $O(n)$的时间复杂度下找到一个局部最小值，

1. 分析该问题，设计一个算法在$O(logn)$的时间复杂度下找到一个局部最小(返回数值)，要求：分析算法设计思路，并且使用python编程实现
2. * 设计算法找出所有局部最小值，分析算法设计思路，并使用python编程实现

1. 通过二分搜索寻找局部最小，算法伪代码如下：

```
def find_one_local_min(A):
    n = len(A)
    left, right = 1, n - 2   # 因为 A[0] > A[1] 和 A[n-2] < A[n-1]
    while left <= right:
        mid = (left + right) // 2
        if A[mid] < A[mid - 1] and A[mid] < A[mid + 1]:
            return A[mid]
        elif A[mid] > A[mid - 1]:
            right = mid - 1
        else:
            left = mid + 1
    # 检查边界情况（理论上不会走到这里，因为题目保证存在局部最小）
    if left == 0 and A[0] < A[1]:
        return A[0]
```

```
        if right == n - 1 and A[n - 1] < A[n - 2]:
            return A[n - 1]
        return None
```

问题 8：

给定包含n个不同数字的一组数，寻找一种基于比较的算法在这组数中找到k个最小的数字，并按顺序输出它们。

1. 将n个数先进行排序，然后按顺序输出最小的k个数。要求：选择合适的排序算法实现上述操作，计算算法时间复杂度，并使用python编程实现。
2. 建立一个包含这n个数的堆（heap），并且调用 k 次Extract-min 按顺序输出最小的k个数。使用往空堆中不断插入元素的方法建立堆，分析这种方法建堆的时间复杂度，并使用python编程实现
3. * 假设数组中包含的数据总数目超过了计算机的存储能力，请设计一个算法，找到这堆数据的前k小的数值，计算时间复杂度，并使用python实现该算法，假设计算机一定能存储k个数据。

1.冒泡排序，时间复杂度是 $O(n^2)$，代码见 ipynb

2.堆排始终从最后一个非叶子节点开始向上构建，树高度 h=logn，对于根节点，需要调整 h 次，第一层节点需要调整 h-1 次，…对于最后一层节点需要调整 1 次。因此，

$$T(n) = 2^{h-1}(h-1) + 2^{h-2}(h-2) + \cdots + 2\times 1$$

$$2T(n) = 2^{h}(h-1) + 2^{h-1}(h-2) + \cdots + 2^{2}\times 1$$

$$\begin{aligned} T(n) &= 2T(n) - T(n) \\ &= 2^{n}(h-1) + 2^{h-1}(h-2) + \cdots + 2^{2}\times 1 - 2^{h-1}(h-1) - 2^{h-2}(h-2) \cdots - 2\times 1 \\ &= (h-1)2^{h} - 2^{h-1} - 2^{h-2} - \cdots - 2\times 1 \\ &= (h-1)2^{h} - \frac{2(1-2^{h-1})}{1-2} \\ &= (h-1)2^{h} + 2 - 2h \\ &= n(\log h - 1) + 2 - n \\ &= n\log n - 2n + 2 \end{aligned}$$

$\therefore T(n) = O(n)$，即建堆的时间复杂度为 $O(n)$

问题 9：

选择问题:给定一个包含n个未排序值的数组A和一个$k \le n$的整数，返回A中最小的第k项。

在课堂上，学了一个简单的O(n)随机算法来解决选择问题。事实上还有一种更复杂的最坏情况下时间复杂度为$O(n)$ 的选择算法。假设使用一个黑盒过程来实现这个O(n)选择算法：给定一个数组A、 $p < r$ 和 k，$BB(A, p, r, k)$ 可以在$O(r - p + 1)$时间内找到并报告$A[p..r]$中第k小的项的下标。假设你可以在线性时间内处理Partition过程。

1. 请分析如何修改 Quicksork 算法可以使其最差情况下的运行时间为 $O(nlogn)$，使用伪代码实现，并分析为何修改后的版本最差情况的运行时间为$O(nlogn)$

note：伪代码中，你可以直接调用用 `BB(A, p, r, k)` 这个函数用于表示在最坏情况下时间复杂度为$O(n)$的选择算法；

2. 找到一个更好的算法报告数组A中的前k小的项，使用伪代码表示你的算法，并分析你算法的时间复杂度。

举例：A=[13, 3, 7, 9, 11, 1, 15, 2, 8, 10, 12, 16, 14, 5]，当k=4时，应该报告1, 2, 3, 4

note： 最直观的方法就是先将数组A排序，然后从左向右报告其前k项，这样操作的时间复杂度为$O(nlogn)$. 调用用 `BB(A, p, r, k)` 设计一个算法使其报告无序数组A的前k项，满足时间复杂度好于$\Theta(nlogn)$，并且当$k = \sqrt{n}$时，你设计的算法时间复杂度应该为$\Theta(n)$.

3. 给定一个大小为n的数组，找到一个 时间复杂度为$O(nlogk)$ 的算法，该算法将A中的元素重新排序，使它们被划分为k个部分，每个部分的元素小于或等于下一部分的元素。假设n和k都是2的幂。使用伪代码表示你的算法，并分析时间复杂度。

e.g.:

数组：[1, 3, 5, 7, 9, 11, 13, 15, 2, 4, 6, 8, 10, 12, 16, 14]，k=4,

对应重新排序的数组为：[1, 3, 2, 4] [7, 6, 5, 8] [12, 11, 10, 9] [13, 14, 16, 15]

1\. 通过 BB(A,p,r,k)选择中位数作为基准进行排序，伪代码如下:

```
ModifiedQuickSort(A, p, r):
    if p < r:
        q=BB(A,p,r,(r-p+1)//2)# 使用 BB 函数找到中位数作为 pivot
        pivot = Partition(A, p, r, A[q])
        ModifiedQuickSort(A, p, pivot-1)
        ModifiedQuickSort(A, pivot+1, r)
Partition(A, p, r, pivot):
```

```
    swap(A[pivot],A[r])# 把 pivot 移至数组末尾
    x = A[r]
    i= p-1
    for j= p to r-1:
      if A[j] <= x:
        i=i+ 1
        swap(A[i], A[j])
    swap(A[i+ 1], A[r])
    return i+1
```

递归操作时间复杂度为：$T(n)=2T(n/2)+n=2^2T(n/2^2)+2n=\cdots=2^tT(n/2^t)+tn$，令 $n/2^t=1$，$t=\log n$，$T(n)=O(n+n\log n)=O(n\log n)$

2. 算法伪代码如下：

```
QuickSelect(A, p, r, k)
  if p == r
    return A[p:k]//当数组缩小到只有 k 个元素时，直接返回这 k 个元素
  q=BB(A,p,r,k) // 在数组 A[p...]中找到第 k 小元素的索引
  if q == k-1
    return A[p:q+1]// 如果 q 正好是 k-1，则 A[p..q]就是前 k 小的元素
  else if q < k-1
    return QuickSelect(A,q+1,r,k)// 第 k 小元素在右边的子数组中
  else
    return QuickSelect(A,p,q-1,k)// 第 k 小元素在左边的子数组中，继续搜索
```

时间复杂度为

$$T(n)=T(\frac{n}{2})+n=\cdots=T(\frac{n}{2^t})+n\sum_{i=1}^{t}\frac{1}{2^{i-1}}$$

$$=T(\frac{n}{2^t})+n\frac{1-(\frac{1}{2})^t}{1-\frac{1}{2}}=T(\frac{n}{2^t})+2n\left(1-\left(\frac{1}{2}\right)^t\right)$$

令 $n/2^t=1$, $t=\log n$, $T(n)=O(n)$

3. 算法伪代码如下：

```
PartitionArray(A, p, r, k)
  if k<= 1 or p >=r
    return

  if k == 2
    q=Partition(A,p,r)# 标准的快速排序分区
    PartitionArray(A,p,q,1)# 对左半部分递归进行排序
    PartitionArray(A,q+1,r,1)# 对右半部分递归进行排序
    return
  q=Partition(A,p,r)# 标准的快速排序分区
  left_size=q-p+1 #左侧部分的大小
  num_left_parts=k//2 #左侧应该分的部分数
  num_right_parts=k-num_left_parts #右侧的部分数
  PartitionArray(A,p,q,num_left_parts) # 对左侧递归分区
```

```
    PartitionArray(A,q+1,r,num_right_parts)#  对右侧递归分区
```

时间复杂度为 $T(n,k)=2T(n/2,k/2)+n$

$$=2^2T(n/2^2,k/2^2)+2n$$

...

$$=2^tT(n/2^t,k/2^t)+tn$$

令 $k/2^t=1$， $t=\log k, T(n,k)=O(n\log n)$

问题 10：

给定一个包含m个**字符串**的数组A, 其中不同的字符串可能有不同的字符数, 但数组中所有字符串的字符总数为n。设计一个算法在 $O(n)$ 时间内对字符串进行排序, 分析算法设计方案, 计算其时间复杂度, 并基于python编程实现该算法。请注意, 假设字符串只包含"a","b",...,"z",

举例1：数组A=["a", "da", "bde", "ab", "bc", "abdc", "cdba"], 排序后的数组应该为：['a', 'ab', 'abdc', 'bc', 'bde', 'cdba', 'da']

举例2：数组A=['ab', 'a', 'b', 'abc', 'ba', 'c'], 排序后的数组应该为：

['a', 'ab', 'abc', 'b', 'ba', 'c']

举例3：数组A=['aef', 'yzr', 'wr', 'ab', 'bhjc', 'lkabdc', 'pwcdba'], 排序后的数组应该为：['ab', 'aef', 'bhjc', 'lkabdc', 'pwcdba', 'wr', 'yzr']

note：

- 两个字符之间的比较可以考虑比较他们对应的ASCII码值;
- python中可以使用 `ord("a")` 返回字符 “a”对应的ASCII值

使用基数排序，确定字符串中最长的长度，用来决定排序轮数，通过计数排序处理每一位，算法伪代码如下：

```
RadixSort(strings):
    max_length = max(length(s) for each s in strings)
    for i from max_length - 1 to 0:
        # Initialize counting and output arrays
        count = array of size 27 initialized to 0 #26 letters + 1 for empty charactei
positions
        output = array of size length(strings) initialized to None
        # Count each character's occurrence
        for string in strings:
            if i < length(string):
                character_index=ord(string[i])- ord('a')+ 1
                count[character_index]+=1
            else:
                count[0]+=1
        # Accumulate the counts
        for j from 1 to 26:
            count[]+= count[j-1]
        # Build the output array using the accumulated counts
        for string in reversed(strings):
            if i < length(string):
                character_index = ord(string[i])- ord('a")+ 1
            else:
                character index=0
            index = count[character_index]-1
            output[index= string
            count[character_index]-= 1
        # Copy sorted strings back to original list
```

```
        strings:] = output
return strings
```

时间复杂度为 T(n,k)=O(kn+n)=O(kn)，其中 n 是求最大长度操作，k 是最大字符串长度。

注：本作业在完成中使用了 chatgpt