Homework 2：算法计量

Part A

请完成以下实验，使用图像展示你的结果，并给出简短的说明。

1. 做计时实验，验证 Python 中的 `list` 按索引取值时的时间复杂度是 `O(1)` 的。
2. 做计时实验，验证 Python 中的 `dict` 的 get item (`d[key]`) 操作和 set item (`d[key] = value`) 操作都是 `O(1)` 的。
3. 做计时实验，比较 `list` 和 `dict` 的 `del` 操作性能。提示：在测试 `list` 的删除性能时，应当区分是特定位置删除还是随机位置删除，分类讨论。
4. 做计时实验，测试 Python 中的 `set` 的插入和查询性能。

注意：在设计实验时，应尽可能排除除了**待测试操作**之外的操作的运行时间影响，数据范围也不能太小。

Part B

请完成以下实验，使用图像展示你的结果，并给出简短的说明。

1. 课上介绍了快速排序的时间复杂度，请验证快速排序的平均复杂度和最坏复杂度分别为 $O(n \log n)$ 和 $O(n^2)$。以下是一个快速排序代码样例（从小到大排序）。

   ```
   def quick_sort(numbers, start, end):
       if start >= end:
           return numbers
       mid = numbers[start]
       low = start
       high = end
       while low < high:
           while low < high and numbers[high] >= mid:
               high -= 1
           numbers[low] = numbers[high]
           while low < high and numbers[low] <= mid:
               low += 1
           numbers[high] = numbers[low]
       numbers[high] = mid
       quick_sort(numbers, start, low - 1)
       quick_sort(numbers, low + 1, end)
       return numbers
   ```

   调用方式如下：

   ```
   numbers = [1,3,0,2,8,4,4,2,5,6,10,-8,1]
   quick_sort(numbers, 0, len(numbers) - 1)
   ```

   提示：当输入的列表是一个从大到小的有序列表时，该算法会退化到 $O(n^2)$，达到最坏复杂度。

2. 课上介绍了归并排序的时间复杂度，请验证归并排序的平均复杂度为 $O(n \log n)$，并验证即使输入为从大到小的有序列表，归并排序的时间复杂度仍为 $O(n \log n)$。以下是一个归并排序的代码样例（从小到大排序）。

```python
def merge_sort(numbers):
    if len(numbers) <= 1:
        return numbers
    middle = int(len(numbers) / 2)
    left = merge_sort(numbers[:middle])
    right = merge_sort(numbers[middle:])
    merged = []
    while left and right:
        merged.append(left.pop(0) if left[0] <= right[0] else right.pop(0))
    merged.extend(right if right else left)
    return merged
```

调用方式如下：

```python
numbers = [1,3,0,2,8,4,4,2,5,6,10,-8,1]
merge_sort(numbers)
```

Part C

在本部分中，你需要根据指示完成若干个算法练习（以及 OpenJudge 上对应的题目）并验证时间复杂度。

1. 试除法判断质数。一种很简单的判断一个数字 n 是否为质数的方法是，用 n 除以所有大于 1 小于 n 的数，如果存在一个数能整除 n，则 n 为合数，否则 n 为质数。请**根据前面的表述**实现一个算法判断一个数是否为质数，给出算法的时间复杂度并简要说明原因（不用严谨证明，下同）。
2. 试除法判断质数的优化。请改进上一部分中的算法，给出一个复杂度不超过 $O(\sqrt{N})$的算法，并完成 OpenJudge 上的 **判断质数 (1)**。给出算法的时间复杂度并简要说明原因。
3. 请基于试除法判断质数的方法，完成 OpenJudge 上的 **判断质数 (2)**。在本题中，你需要批量判断一个数是否为质数，并且应该注意到查询次数非常多，因此你需要给出一个合理的算法。给出算法的时间复杂度并简要说明原因。
4. 埃拉托斯特尼筛法是一种更优的生成质数的方法，得名于古希腊数学家埃拉托斯特尼。其基本步骤是从最小的素数 2 开始，将该素数的所有倍数标记成合数，而下一个尚未被标记的最小自然数 3 即是下一个素数。如此重复这一过程，将各个素数的倍数标记为合数并找出下一个素数，最终便可找出一定范围内所有素数。其伪代码可以表示如下（用于寻找不大于 n 的所有质数）：

   ```
   输入：整数n > 1

   设 A 为 bool 列表，初始时全部设成true（忽略 A[0] 和 A[1]）。

   for i = 2, 3, 4, ..., 不超过 sqrt(n):
       if A[i] 为 true:
           for j = i^2, i * (i + 1), i * (i + 2), ..., 不超过 n:
               A[j] = false

   输出：使 A[i] 为 true 的所有 i。
   ```

   请你实现该算法并完成 OpenJudge 上的 **判断质数 (3)**，并画图验证该算法比基于试除法的算法更快（在寻找不大于 n 的所有质数这一问题上）。
5. 相信埃拉托斯特尼筛法给了你很多启示，请尝试利用类似的思路，完成 OpenJudge 上的 **求因数和**。本小题只需要在 OpenJudge 上完成即可。