算法效率分析——排序与TopK问题

数学与统计学院

王曦 2021192010

实验内容

- 实现选择排序、冒泡排序、归并排序、快速排序、插入排序算法.
- 比较上述算法在不同输入数据规模下的效率.
- 设计一个在10亿个数据中快速选出10个数的算法.

创新点1：控制变量法、离散化、特殊数据

- 为比较不同排序算法的效率，应保证各个算法的输入数据相同，即**控制变量法**.
- 实现随机数生成器gen.cpp，将生成的随机数据保存到文件中.

- 因本次实验的算法与元素的值域无关，且不涉及算法的稳定性，
  故生成的输入数据的元素范围为1 ~ n. 对一般的情况，可将元素离散化为1 ~ n.
- 优点：①生成随机数简单；②在TopK问题中避免产生相等的元素，且可肉眼判断答案是否正确.

- 因部分算法的效率与输入数据有关，类似于算法竞赛中通过构造极限数据卡掉非正解的做法的思路，构造了三组特殊的数据作为20组随机数据的前3组，以检测各排序算法的性能：
  ①全为1的数据；②1 ~ n的升序序列；③n ~ 1的降序序列.

- [**演示**] 用gen.cpp生成随机数据

约定

- 环境: MinGW-w64编译器, GNU C++17.
- 开启O2优化, 减短 $O(n^2)$ 的算法跑1e7的数据的时间.

- ①以将序列非降序排列为例.
- ②记录的程序运行时间不包括输入数据的时间.
- ③以n = 1000时的实际运行时间为基准点, 校准误差.

$O(n^2)$ 的排序算法演示

- [**演示**] 以选择排序为例，演示n = 10000时的效率.

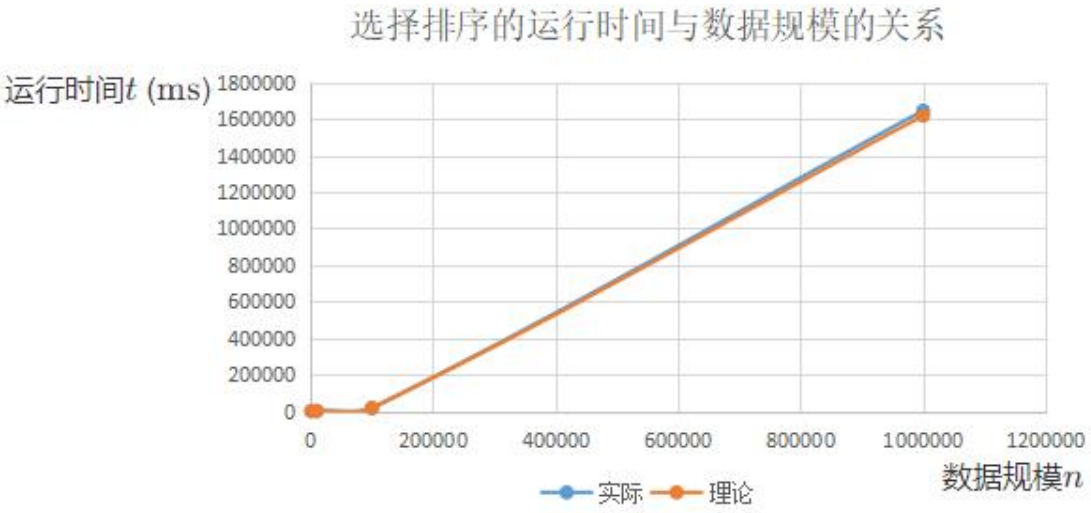
- 以n = 10000时的实际运行时间为基准点，计算其他输入规模对应的理论运行时间.

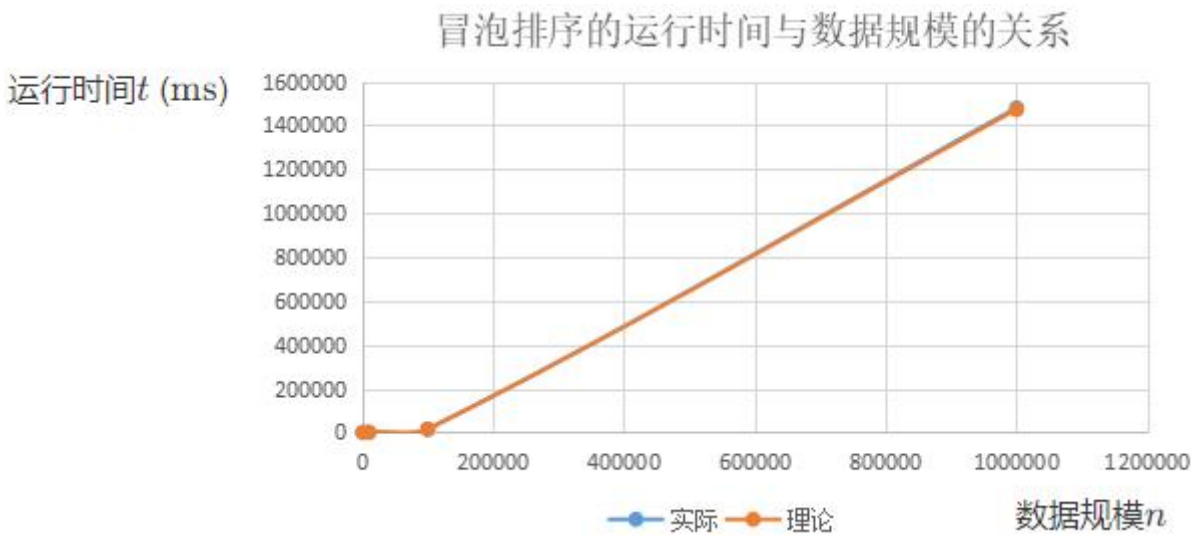
数据规模n	1000	10000	100,000	1,000,000
实际运行时间(ms)	2.250	161.850	16345.300	1646014.500
理论运行时间(ms)	1.619	161.850	16185.000	1618500.000
误差	39.0%	0.0%	1.0%	1.7%

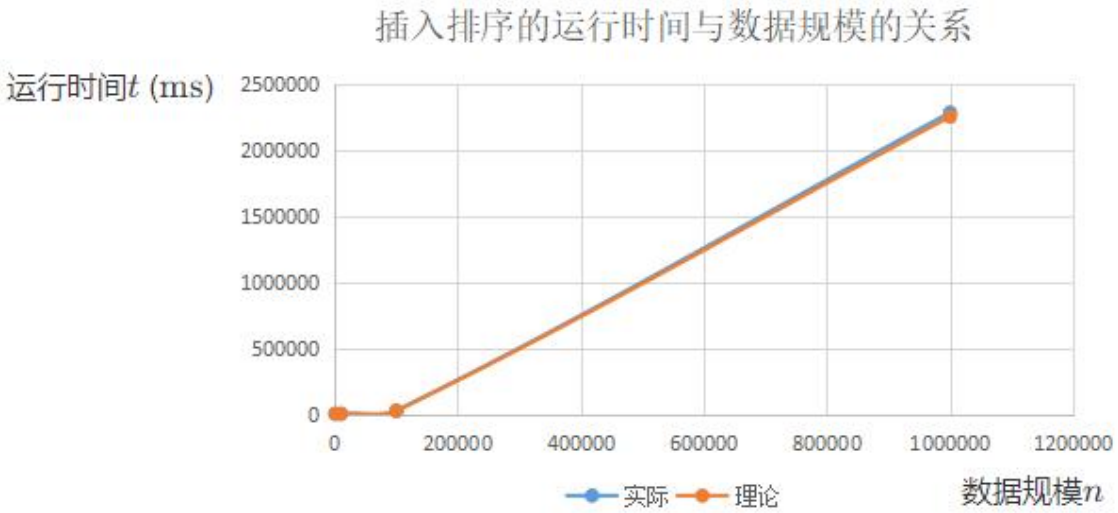
- 因 $\frac{n_1^2}{n_0^2} = \frac{t_1}{t_0}$，则n = 1000时的理论运行时间 $t_1 = \left(\frac{n_1}{n_0}\right)^2 t_0 = \left(\frac{1000}{10000}\right)^2 \times 161.850\ \text{ms} = 1.619\ \text{ms}$，

- 进而误差 $\delta = \frac{2.250 - 1.619}{1.619} \times 100\% = 39.0\%$.

$O(n^2)$ 的排序算法的运行时间与数据规模的关系





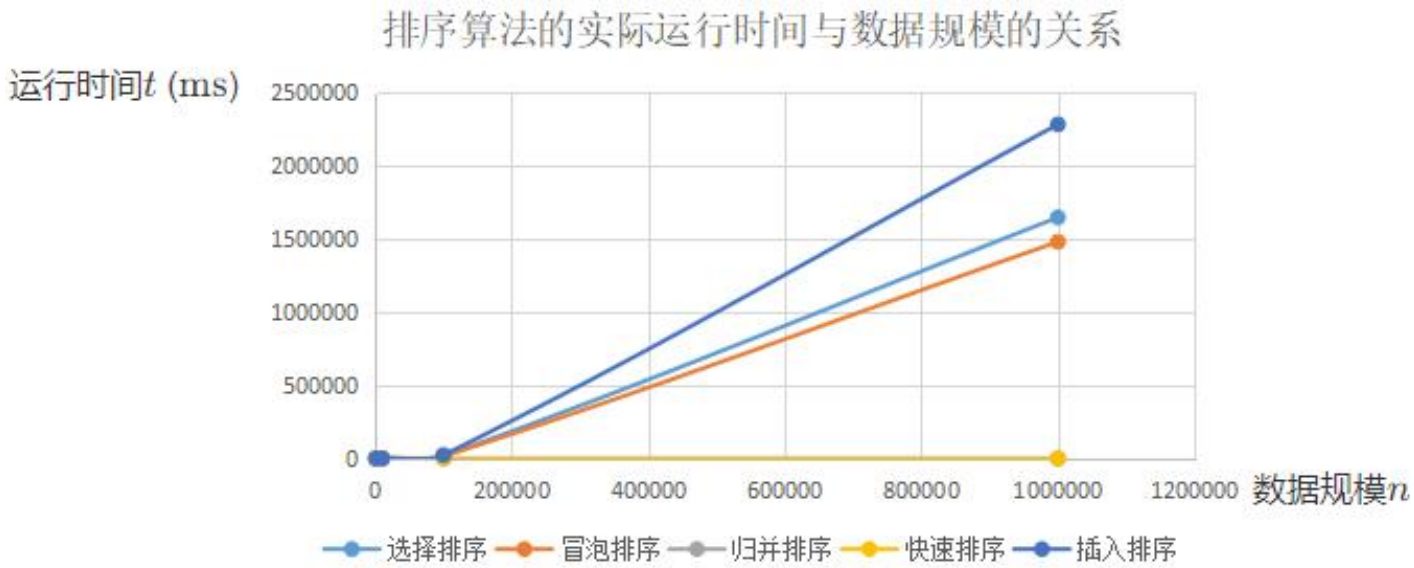


$O(n^2)$ 的排序算法的异同

- 共同点:
  - ①图象显示, 程序运行时间与数据规模的关系呈抛物线.
  - ②数据规模较小时误差较大, 数据规模较大时误差较小.
- 解释:
  - ①因大部分数据是随机生成的, 几乎未达到各排序算法的最坏情况.
  - ②几乎抛物线的图象证明了各排序算法 $O(n^2)$ 的平均时间复杂度.
  - ③数据规模小时误差较大的原因:
    - (i)数据较少时, 随机化程度不如数据较大的情况.
    - (ii)数据较少时, 前3组特殊数据的作用被放大.

$O(n^2)$ 的排序算法的异同

- 不同点:
  - 数据规模较大时，插入排序、选择排序、冒泡排序的实际运行时间差距较大.
- 解释:
  - ①数据规模较小时，上述三个排序算法的实际运行时间差距不大.
  - ②数据规模较小时，平均时间复杂度更低的算法可能因实现较为繁琐而导致时间复杂度的常数较大，进而实际运行时间高于平均时间复杂度高的算法.
  - ③冒泡排序的整体性能优于另外两个排序算法，插入排序的整体性能最差，这是因为插入排序的最优时间复杂度是 $O(n^2)$，且与输入数据无关；而选择排序和冒泡排序的最优时间复杂度为 $O(n)$，且与输入数据有关.



$O(n \log n)$ 的排序算法演示

- [**演示**] 以归并排序为例，演示n = 100,000时的效率.

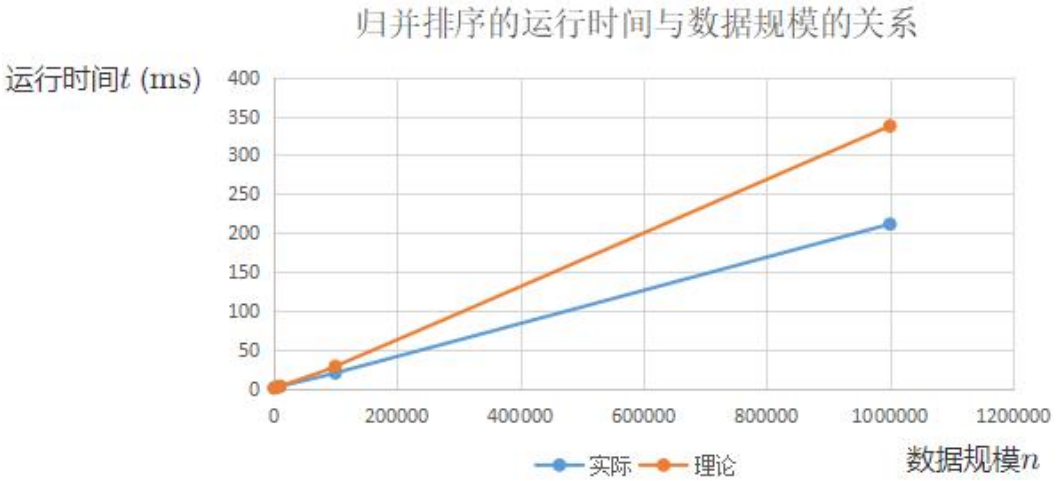
- 以n = 10000时的实际运行时间为基准点，计算其他输入规模对应的理论运行时间.

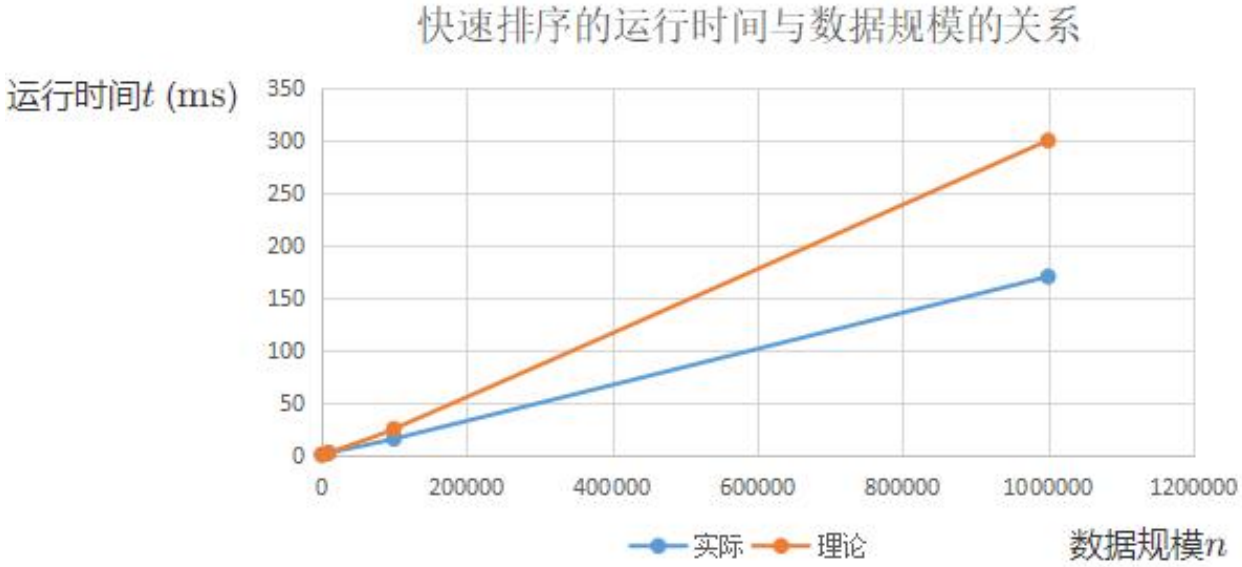
数据规模n	1000	10000	100,000	1,000,000
实际运行时间(ms)	0.150	2.250	19.500	211.150
理论运行时间(ms)	0.169	2.250	28.125	337.500
误差	-11.2%	0.0%	-30.7%	-37.4%

- 因 $\frac{n_1 \log_2 n_1}{n_0 \log_2 n_0} = \frac{t_1}{t_0}$，则n = 1000时的理论运行时间 $t_1 = \frac{n_1 \log_2 n_1}{n_0 \log_2 n_0} t_0 = \frac{1000 \times 3 \log_2 10}{10000 \times 4 \log_2 10} \times 2.250\ \text{ms} = 0.169\ \text{ms}$，

- 进而误差 $\delta = \frac{0.150 - 0.169}{0.169} \times 100\% = -11.2\%$.

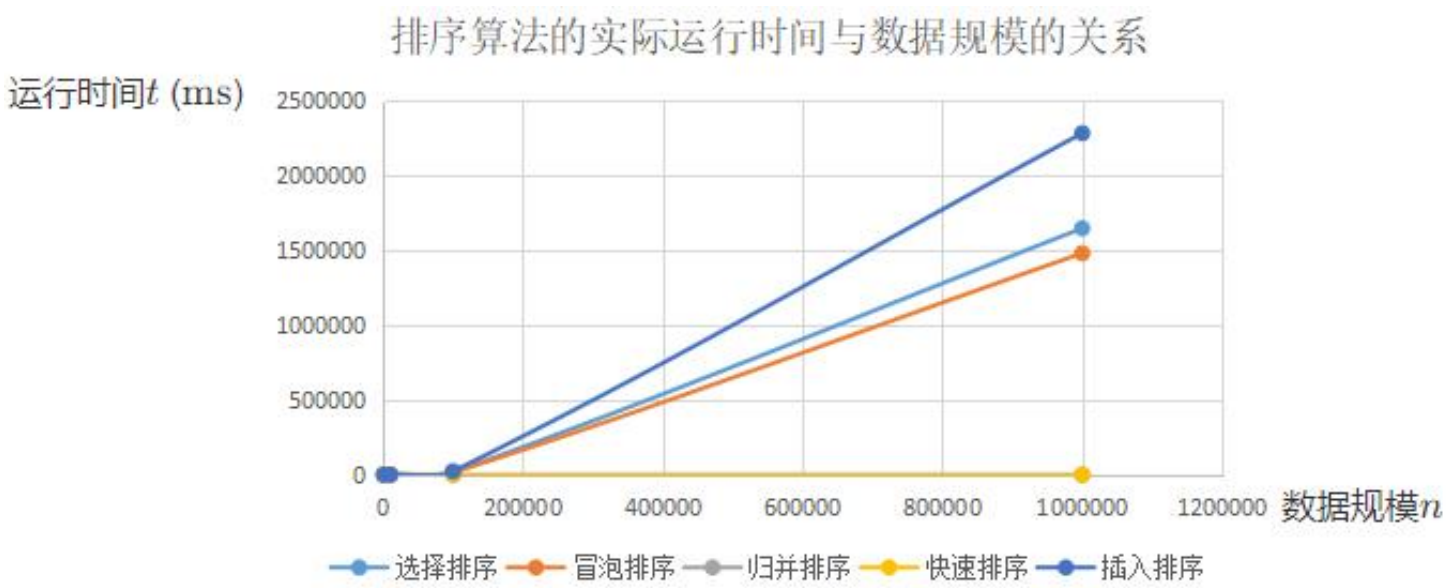
$O(n \log n)$ 的排序算法的运行时间与数据规模的关系





$O(n \log n)$ 的排序算法的异同

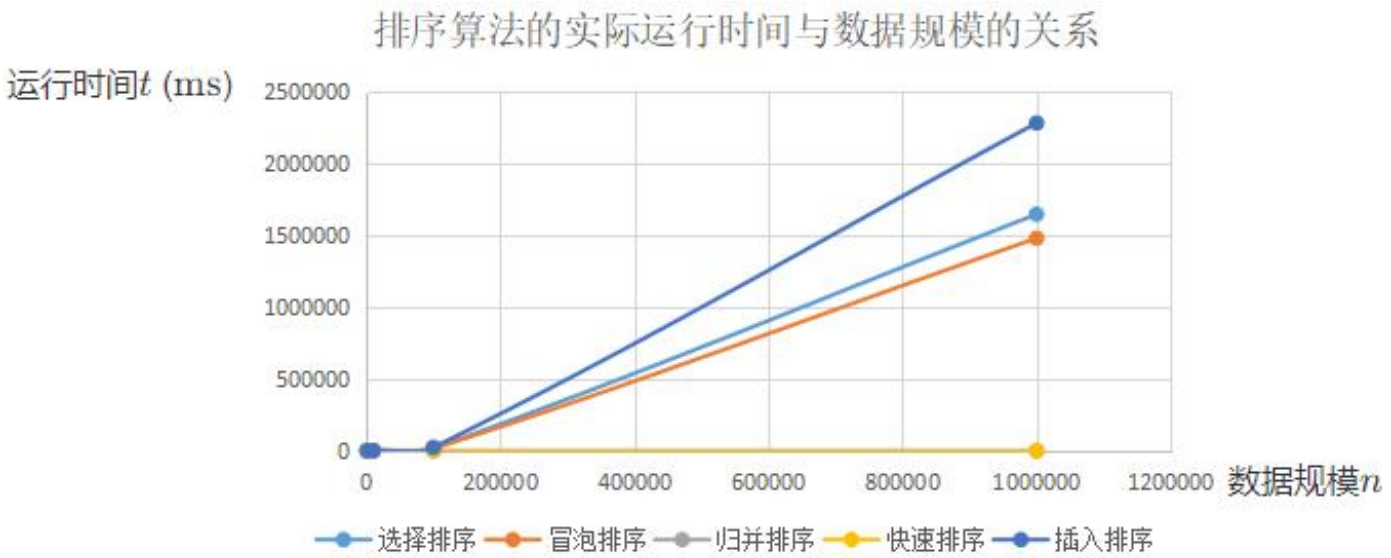
- 共同点:
  - ① 数据规模较大时，平均时间复杂度 $O(n \log n)$ 的算法的实际运行时间明显低于平均时间复杂度 $O(n^2)$ 的算法.
  - ② 因数据规模n较小时，log n对时间复杂度的贡献可视为常数，故从图象上看，归并排序和快速排序的运行时间与数据规模几乎成线性关系.
  - ③ 实验中的归并排序和快速排序都采用递归实现，而理论时间复杂度未考虑函数调用带来的开销，理应实际运行时间高于理论运行时间. 但实验结果为: 实际运行时间低于理论运行时间，且随数据规模的增大，两时间的差距越大.
- 解释:
  - 编译器开启O2优化后降低了函数递归调用的开销，优化了归并排序和快速排序的汇编语言，使其效率提高.



$O(n \log n)$ 的排序算法的异同

- 不同点:
  - ①快速排序在随机数据下的整体性能优于归并排序，且与输入数据有关.
  - ②归并排序是稳定 $O(n \log n)$ 的时间复杂度，与输入数据无关.

- 解释:
  - 快速排序的递归层数与输入数据有关，最坏递归n层，达到最坏时间复杂度 $O(n^2)$.



创新点2：优化排序算法

- 事实上，本实验中的排序算法的实现仍有优化空间，如减少非必要的比较次数、将递归改为非递归实现等.
- 经测试，这些实现上的优化都比不过打开编译器的O2优化，故不再赘述.

TopK问题

- [**题意**] 设计一个算法，在10亿个数据中快速选出最大的10个数，其中每个数据占4个字节.

- 思路来源: C++的algorithm库中的std::sort()函数的优化方式
- 思路: 堆. 具体地，将输入数据的前10个元素建堆. 对之后的每个元素，若它比当前堆中的最小元素小，则将堆的最小元素置为当前元素并调整堆的形态. 遍历完输入数据的所有元素后，做10次弹出堆顶元素并调整堆的形态的操作即可选出最大的10个数.

- 建堆时间复杂度 $O(k \log k)$，调整时间复杂度 $O(\log k)$，
  总时间复杂度 $O(k \log k + (n-k) \log k + k \log k) = O((n+k) \log k)$.

- [**演示**] 在n = 1000的数据下验证算法的正确性.

创新点3：TopK问题的加强

- 数据规模扩大100倍时，这些数据的空间会达到TB的级别，
  此时无法将数据整体放入内存，需借助外存与内存的数据交换.

- 扩展:
- 若TopK问题数据的值域较小或可离散化到较小的值域，则还可用桶排序、计数排序解决.

Thanks!