实验一: 排序算法时间复杂度比较

PB20061372　朱云沁　Apr. 21, 2023

1 实验说明

1.1 实验目的

熟悉主要排序算法, 对排序算法的时间复杂度及其影响因素有直观认识.

1.2 实验内容

1. 随机生成一个包括 n 个整数的数组, 元素取值范围是 $[1, 1000]$, 利用插入排序, 归并排序, 快速排序, 堆排序, 基数排序, 桶排序等算法对数组进行非降序排序, 记录不同算法的运行时间.
2. 改变数组规模 $n = 5\times10^4, 1\times10^5, 2\times10^5, 3\times10^5, 5\times10^5$, 记录不同规模下各个算法的排序时间.
3. 对固定规模 $n = 1\times10^5$ 的数组进行随机扰乱, 对扰乱后的数组进行排序并记录各个算法的排序时间. 本实验要求重复 5 次, 观察输入数据分布和运行时间的关系.
4. 各个算法的时间复杂度理论分析, 并与实验二进行对比.

1.3 检查代码正确性

1. 产生一个长度为 20 随机数组, 允许有重复元素, 分别用各种算法进行排序并输出排序结果.
2. 产生一个长度为 1000 的随机数组, 允许有重复元素, 分别用各种算法进行排序并输出排序结果以及耗费时间, 使用 `for` 循环比较其他算法的排序结果是否与插入排序的结果相同.
3. 固定随机数种子, 方便复现实验结果.

2 排序算法及其时间复杂度

2.1 插入排序

2.1.1 伪代码

Algorithm 1: Insertion Sort

Input: An array A of n numbers.
Output: The array A sorted in nondecreasing order.

```
for i = 2 to n:
    key ← A[i]
    j ← i
    while j > 0 and A[j − 1] > key:
        A[j] ← A[j − 1]
        j ← j − 1
    A[j] ← key
```

2.1.2 时间复杂度

插入排序算法中, **for** 循环的迭代次数为 $n-1$, 最坏情况下, **while** 内循环的迭代次数依次为 $n-1, n-2, \cdots, 1$, 因此插入排序的运行时间为

$$T(n) = \Theta(n) + \sum_{i=1}^{n-1} O(i) = O(n^2).$$

最好情况下, 数组已有序, 不执行交换, 时间复杂度为 $O(n)$. 平均情况下, 比较次数可大致视为最坏情况的一半, 时间复杂度仍为 $O(n^2)$.

2.2 归并排序

2.2.1 伪代码

Algorithm 2: Merge Sort

Input: An array A of n numbers.
Output: The array A sorted in nondecreasing order, returned by Merge-Sort$(A, 1, n)$.

```
function Merge-Sort(A, p, r):
    if p < r:
        q ← ⌊(p+r)/2⌋
        Merge-Sort(A, p, q)
        Merge-Sort(A, q + 1, r)
        Merge(A, p, q, r)
function Merge(A, p, q, r):
    n1 ← q − p + 1, n2 ← r − q
    Let L[1..n1 + 1] and R[1..n2 + 1] be new arrays.
    for i = 1 to n1:
        L[i] ← A[p + i − 1]
    for j = 1 to n2:
        R[j] ← A[q + j]
    L[n1 + 1] ← ∞, R[n2 + 1] ← ∞
    i ← 1, j ← 1
    for k = p to r:
        if L[i] ≤ R[j]:
            A[k] ← L[i]
            i ← i + 1
        else:
            A[k] ← R[j]
            j ← j + 1
```

2.2.2 时间复杂度

Merge 操作的时间复杂度为 $\Theta(n)$. 故归并排序的运行时间 $T(n)$ 的递归式:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n).$$

由主定理第二种情况可得, $T(n) = O(n\log n)$.

2.3 快速排序

2.3.1 伪代码

Algorithm 3: Quick Sort

Input: An array A of n numbers.
Output: The array A sorted in nondecreasing order, returned by Quick-Sort$(A, 1, n)$.

```
function Quick-Sort(A, p, r):
    while p < r:
        q ← Partition(A, p, r)
        Quick-Sort(A, p, q − 1)
        p ← q + 1
function Partition(A, p, r):
    x ← A[r]
    i ← p
    for j = p to r − 1:
        if A[j] ≤ x:
            Swap(A[i], A[j])
            i ← i + 1
    Swap(A[i], A[r])
    return i + 1
```

2.3.2 时间复杂度

Partition 操作的时间复杂度为 $\Theta(n)$. 最坏情况下, 划分产生的子数组规模为 $n-1$ 和 0, 运行时间为

$$T(n) = T(n-1) + \Theta(n) = O(n^2).$$

最好情况下, 划分产生的子数组规模平衡, 运行时间为

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) = O(n\log n).$$

当划分为常数比例时, 运行时间仍为 $O(n\log n)$.

平均情况下, 为方便分析, 将数组 A 的各个元素重新命名为 $a_1, a_2, \ldots, a_n$ 其中 a_i 是第 i 小的元素. 定义 $A_{ij} = \{a_i, a_{i+1}, \ldots, a_j\}$. 假定使用 Randomized-Partition 操作, 主元的选取独立且随机. 令随机变量 X_{ij} 指示 a_i 与 a_j 是否被比较, X 为总比较次数, 则

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \Pr\{a_i \text{ and } a_j \text{ are compared}\} \\
&= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \Pr\{a_i \text{ or } a_j \text{ is the first pivot chosen from } A_{ij}\} \\
&= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{2}{j-i+1} < n\sum_{k=1}^{n}\frac{2}{k} = O(n\log n).
\end{aligned}$$

故平均情况的运行时间为 $O(n + n\log n) = O(n\log n)$.

2.4 堆排序

2.4.1 伪代码

Algorithm 4: Heap Sort

Input: An array A of n numbers.
Output: The array A sorted in nondecreasing order.

```
function Heap-Sort(A, n):
    Build-Max-Heap(A, n)
    for i = n to 2:
        Swap(A[1], A[i])
        Max-Heapify(A, 1, i − 1)
function Build-Max-Heap(A, n):
    h ← ⌊n/2⌋
    for i = h to 1:
        Max-Heapify(A, i, n)
function Max-Heapify(A, n, i):
    largest ← i
    repeat:
        i ← largest
        l ← 2i, r ← 2i + 1
        if l ≤ n and A[l] > A[largest]:
            largest ← l
        if r ≤ n and A[r] > A[largest]:
            largest ← r
        Swap(A[i], A[largest])
    until i ≠ largest
```

2.4.2 时间复杂度

由于树高为 $O(\log n)$, Max-Heapify 的时间复杂度为 $O(\log n)$, 故 Build-Max-Heap 的时间复杂度为 $O(n\log n)$ (事实上, 更紧确的界为 $O(n)$, 此处不赘述), 进而 Heap-Sort 的时间复杂度为 $O(n\log n) + nO(\log n) = O(n\log n)$.

最好情况下, 建堆后数组已有序, Max-Heapify 花费 $\Theta(1)$ 时间, 总时间复杂度为 $O(n)$.

2.5 基数排序

2.5.1 伪代码

Algorithm 5: Radix Sort

Input: An array A of n numbers, each of which has d digits and is in base k.
Output: The array A sorted in nondecreasing order.

```
Let C[0..k − 1] be an array of size k, and B[1..n] be an array of size n.
for j = 1 to d:
    for i = 0 to k − 1:
        C[i] ← 0
    for i = 1 to n:
        C[⌊A[i]/k^(j−1)⌋ mod k] ← C[⌊A[i]/k^(j−1)⌋ mod k] + 1
    for i = 1 to k − 1:
        C[i] ← C[i] + C[i − 1]
    for i = n to 1:
        B[C[⌊A[i]/k^(j−1)⌋ mod k]] ← A[i]
        C[⌊A[i]/k^(j−1)⌋ mod k] ← C[⌊A[i]/k^(j−1)⌋ mod k] − 1
    A[1..n] ← B[1..n]
```

2.5.2 时间复杂度

对每一位均使用 Counting-Sort 进行稳定排序, 耗时 $\Theta(n+k)$, 共进行 d 轮, 故总时间复杂度为 $O(d(n+k))$.

2.6 桶排序

2.6.1 伪代码

Algorithm 6: Bucket Sort

Input: An array A of n numbers, each of which is in the interval $[0, 1)$.
Output: The array A sorted in nondecreasing order.

```
Let B[0..n − 1] be an array of n linked lists.
for i = 0 to n − 1:
    B[i] ← ∅
for i = 1 to n:
    Insert A[i] into B[⌊nA[i]⌋].
for i = 0 to n − 1:
    Sort B[i] using Insertion-Sort.
A[1..n] ← B[0] ⊕ B[1] ⊕ ··· ⊕ B[n − 1]
```

2.6.2 时间复杂度

假设输入数据服从均匀分布, 求期望时间复杂度. 桶 $B[i]$ 中的元素位于区间 $[\frac{i}{n}, \frac{i+1}{n})$. 记

$$X_{ij} = \mathrm{I}\left\{\frac{i}{n} \le A[j] < \frac{i+1}{n}\right\} \sim \mathrm{Bernoulli}\left(\frac{1}{n}\right).$$

由于 $X_{i1}, X_{i2}, \ldots, X_{in}$ 相互独立, 故 $B[i]$ 的大小 n_i 满足

$$\begin{aligned}
n_i &= \sum_{j=1}^{n} X_{ij} \sim \mathrm{Binomial}\left(n, \frac{1}{n}\right), \\
\mathrm{E}\left[n_i^2\right] &= \mathrm{E}^2\left[n_i\right] + \mathrm{Var}\left[n_i\right] = 2 - \frac{1}{n}.
\end{aligned}$$

因此, 桶排序在平均情况下的运行时间为

$$\Theta(n) + nO\left(\mathrm{E}\left[n_i^2\right]\right) = O(n).$$

最坏情况下, 所有元素都落在同一个桶中, 此时桶排序退化为 $O(n^2)$ 的插入排序.

3 排序算法运行时间与数组规模的关系

随机生成一个包括 n 个整数的数组, 元素取值范围是 $[1, 1000]$. 改变数组规模 $n = 5\times10^4, 1\times10^5, 2\times10^5, 3\times10^5, 5\times10^5$, 利用插入排序, 归并排序, 快速排序, 堆排序, 基数排序, 桶排序等算法分别对数组进行非降序排序, 记录不同规模下各个算法的排序时间, 如下表所示.

Table 1. Time cost of different sorting algorithms with different array sizes. For each cell, the first number in parentheses is the mean value of 5 runs, and the second number is the standard deviation.

n	std :: sort	std :: qsort	Insertion-Sort
50000	$(1.571 \pm 0.063)\times10^{-3}$	$(2.460 \pm 0.085)\times10^{-3}$	$(1.619 \pm 0.029)\times10^{0}$
100000	$(3.129 \pm 0.039)\times10^{-3}$	$(4.879 \pm 0.096)\times10^{-3}$	$(6.364 \pm 0.018)\times10^{0}$
150000	$(4.705 \pm 0.086)\times10^{-3}$	$(7.179 \pm 0.072)\times10^{-3}$	$(1.436 \pm 0.004)\times10^{1}$
200000	$(6.164 \pm 0.138)\times10^{-3}$	$(9.588 \pm 0.088)\times10^{-3}$	$(2.563 \pm 0.004)\times10^{1}$
250000	$(7.820 \pm 0.193)\times10^{-3}$	$(1.198 \pm 0.029)\times10^{-2}$	$(4.006 \pm 0.004)\times10^{1}$
300000	$(9.043 \pm 0.310)\times10^{-3}$	$(1.431 \pm 0.033)\times10^{-2}$	$(5.771 \pm 0.011)\times10^{1}$
350000	$(1.057 \pm 0.032)\times10^{-2}$	$(1.658 \pm 0.009)\times10^{-2}$	$(7.849 \pm 0.009)\times10^{1}$
400000	$(1.201 \pm 0.016)\times10^{-2}$	$(1.900 \pm 0.011)\times10^{-2}$	$(1.026 \pm 0.001)\times10^{2}$
450000	$(1.383 \pm 0.023)\times10^{-2}$	$(2.131 \pm 0.025)\times10^{-2}$	$(1.302 \pm 0.002)\times10^{2}$
500000	$(1.519 \pm 0.015)\times10^{-2}$	$(2.362 \pm 0.030)\times10^{-2}$	$(1.610 \pm 0.003)\times10^{2}$

n	Merge-Sort	Quick-Sort	Randomized-Quick-Sort
50000	$(8.823 \pm 0.190)\times10^{-3}$	$(5.210 \pm 0.147)\times10^{-3}$	$(5.262 \pm 0.119)\times10^{-3}$
100000	$(1.813 \pm 0.025)\times10^{-2}$	$(1.365 \pm 0.008)\times10^{-2}$	$(1.351 \pm 0.008)\times10^{-2}$
150000	$(2.754 \pm 0.024)\times10^{-2}$	$(2.590 \pm 0.057)\times10^{-2}$	$(2.571 \pm 0.039)\times10^{-2}$
200000	$(3.607 \pm 0.075)\times10^{-2}$	$(4.044 \pm 0.085)\times10^{-2}$	$(4.125 \pm 0.042)\times10^{-2}$
250000	$(4.485 \pm 0.071)\times10^{-2}$	$(5.938 \pm 0.035)\times10^{-2}$	$(5.994 \pm 0.046)\times10^{-2}$
300000	$(5.489 \pm 0.076)\times10^{-2}$	$(8.154 \pm 0.214)\times10^{-2}$	$(8.084 \pm 0.174)\times10^{-2}$
350000	$(6.345 \pm 0.054)\times10^{-2}$	$(1.057 \pm 0.013)\times10^{-1}$	$(1.043 \pm 0.017)\times10^{-1}$
400000	$(7.244 \pm 0.072)\times10^{-2}$	$(1.342 \pm 0.004)\times10^{-1}$	$(1.302 \pm 0.004)\times10^{-1}$
450000	$(8.116 \pm 0.179)\times10^{-2}$	$(1.655 \pm 0.031)\times10^{-1}$	$(1.619 \pm 0.010)\times10^{-1}$
500000	$(8.936 \pm 0.074)\times10^{-2}$	$(1.977 \pm 0.010)\times10^{-1}$	$(1.957 \pm 0.009)\times10^{-1}$

n	Heap-Sort	Radix-Sort	Bucket-Sort
50000	$(8.915 \pm 0.212)\times10^{-3}$	$(3.101 \pm 0.055)\times10^{-3}$	$(7.036 \pm 0.240)\times10^{-3}$
100000	$(1.914 \pm 0.008)\times10^{-2}$	$(6.167 \pm 0.050)\times10^{-3}$	$(1.615 \pm 0.142)\times10^{-2}$
150000	$(2.992 \pm 0.015)\times10^{-2}$	$(9.273 \pm 0.019)\times10^{-3}$	$(3.487 \pm 0.846)\times10^{-2}$
200000	$(4.143 \pm 0.041)\times10^{-2}$	$(1.238 \pm 0.008)\times10^{-2}$	$(5.926 \pm 0.815)\times10^{-2}$
250000	$(5.229 \pm 0.044)\times10^{-2}$	$(1.515 \pm 0.007)\times10^{-2}$	$(8.375 \pm 0.346)\times10^{-2}$
300000	$(6.380 \pm 0.156)\times10^{-2}$	$(1.854 \pm 0.016)\times10^{-2}$	$(1.070 \pm 0.109)\times10^{-1}$
350000	$(7.522 \pm 0.161)\times10^{-2}$	$(2.215 \pm 0.046)\times10^{-2}$	$(1.178 \pm 0.052)\times10^{-1}$
400000	$(8.880 \pm 0.055)\times10^{-2}$	$(2.476 \pm 0.012)\times10^{-2}$	$(1.349 \pm 0.016)\times10^{-1}$
450000	$(9.948 \pm 0.257)\times10^{-2}$	$(2.793 \pm 0.009)\times10^{-2}$	$(1.477 \pm 0.029)\times10^{-1}$
500000	$(1.142 \pm 0.019)\times10^{-1}$	$(3.060 \pm 0.007)\times10^{-2}$	$(1.630 \pm 0.032)\times10^{-1}$

在对数坐标系中, 绘制不同算法数组规模与运行时间的折线图, 如下图所示.

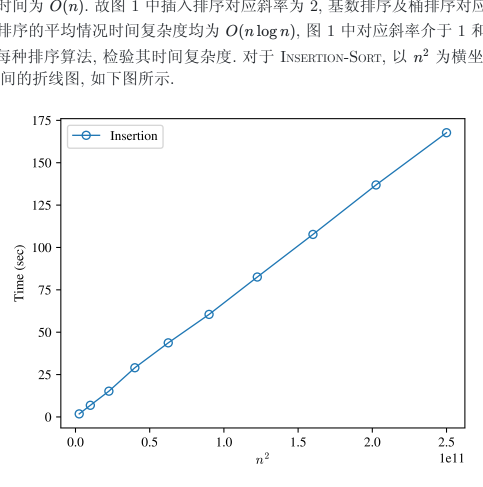

Figure 1. Time cost of different sorting algorithms with different array sizes.

由图 1 可以得出, 平均情况下, 各算法 $\log T(n)$ 随 $\log n$ 增长的速率:

1. 根据前文理论分析, 插入排序的期望运行时间为 $O(n^2)$, 基数排序 (d 与 k 固定) 及桶排序的期望运行时间为 $O(n)$. 故图 1 中插入排序对应斜率为 2, 基数排序及桶排序对应斜率为 1.
2. 其余三种排序的平均情况时间复杂度均为 $O(n\log n)$, 图 1 中对应斜率介于 1 和 2 之间.

下面, 针对每种排序算法, 检验其时间复杂度. 对于 Insertion-Sort, 以 n^2 为横坐标, 绘制数组规模与运行时间的折线图, 如下图所示.

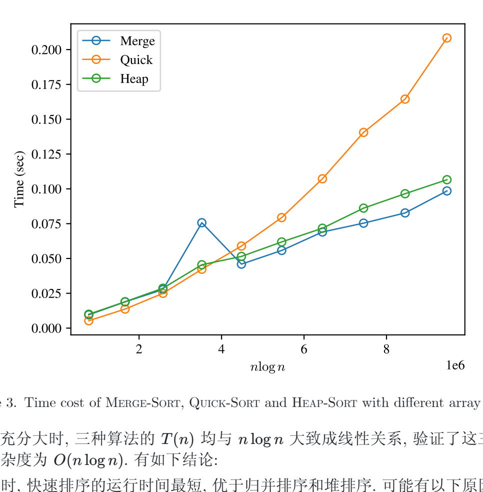

Figure 2. Time cost of Insertion-Sort with different array sizes.

可见 $T(n)$ 与 n^2 成线性关系, 验证了插入排序的平均时间复杂度为 $O(n^2)$.

对于 Merge-Sort, Quick-Sort 和 Heap-Sort, 以 $n\log n$ 为横坐标, 绘制数组规模与运行时间的折线图, 如下图所示.

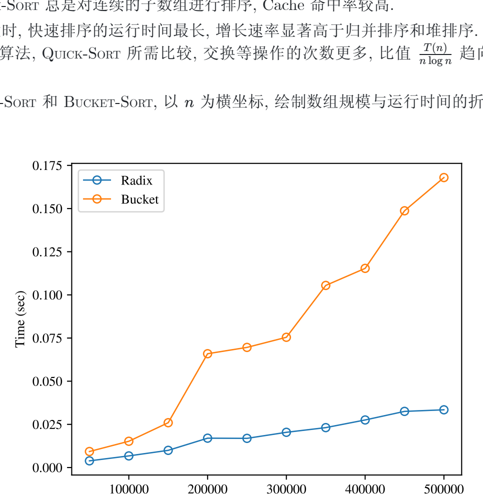

Figure 3. Time cost of Merge-Sort, Quick-Sort and Heap-Sort with different array sizes.

可见, 当 n 充分大时, 三种算法的 $T(n)$ 均与 $n\log n$ 大致成线性关系, 验证了这三种排序算法的平均时间复杂度为 $O(n\log n)$. 有如下结论:

1. 当 n 较小时, 快速排序的运行时间最短, 优于归并排序和堆排序. 可能有以下原因:
   (1) Quick-Sort 存在尾递归, 相较 Merge-Sort 递归开销较小.
   (2) Quick-Sort 为原地排序, 而 Merge 操作在内存分配, 复制, 释放等方面开销较大.
   (3) Quick-Sort 总是对连续的子数组进行排序, Cache 命中率较高.
2. 当 n 较大时, 快速排序的运行时间最长, 增长速率显著高于归并排序和堆排序. 这说明, 相较另外两种算法, Quick-Sort 所需比较, 交换等操作的次数更多, 比值 $\frac{T(n)}{n\log n}$ 趋向于更大的常数.

对于 Radix-Sort 和 Bucket-Sort, 以 n 为横坐标, 绘制数组规模与运行时间的折线图, 如下图所示.

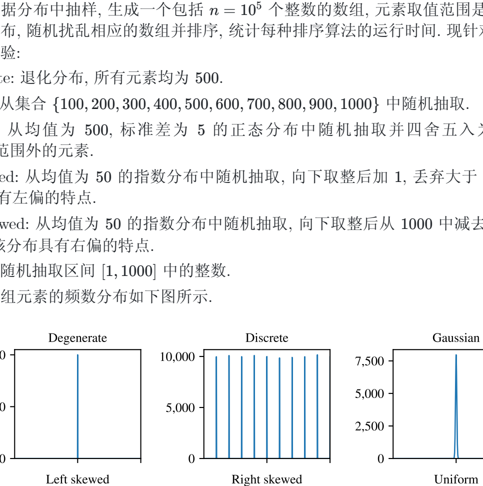

Figure 4. Time cost of Radix-Sort and Bucket-Sort with different array sizes.

可见, 平均情况下, 两种算法的 $T(n)$ 均与 n 大致成线性关系, 符合其时间复杂度. 然而, 由于实现方式的不同, 两种算法的运行时间存在较大差异. Radix-Sort 针对整数实现 $\Theta(d(n+k))$ 的排序, 在所有算法中总是具有最短的运行时间.

4 排序算法运行时间与数据分布的关系

从不同的数据分布中抽样, 生成一个包括 $n = 10^5$ 个整数的数组, 元素取值范围是 $[1, 1000]$. 针对每种数据分布, 随机扰乱相应的数组并排序, 统计每种排序算法的运行时间. 现针对以下六种数据分布进行实验:

1. Degenerate: 退化分布, 所有元素均为 500.
2. Discrete: 从集合 $\{100, 200, 300, 400, 500, 600, 700, 800, 900, 1000\}$ 中随机抽取.
3. Gaussian: 从均值为 500, 标准差为 5 的正态分布中随机抽取并四舍五入为整数, 丢弃 $[1, 1000]$ 范围外的元素.
4. Left skewed: 从均值为 50 的指数分布中随机抽取, 向下取整后加 1, 丢弃大于 1000 的元素. 该分布具有左偏的特点.
5. Right skewed: 从均值为 50 的指数分布中随机抽取, 向下取整后从 1000 中减去, 丢弃小于 1 的元素. 该分布具有右偏的特点.
6. Uniform: 随机抽取区间 $[1, 1000]$ 中的整数.

抽样所得数组元素的频数分布如下图所示.

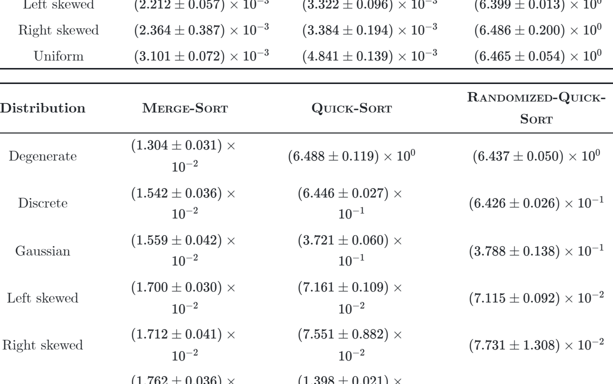

Figure 5. Histogram of the data distributions.

实验得到每种排序算法的运行时间如下表所示.

Table 2. Time cost of different sorting algorithms with different data distributions.

Distribution	std :: sort	std :: qsort	Insertion-Sort
Degenerate	$(6.694 \pm 0.132)\times10^{-5}$	$(1.711 \pm 0.036)\times10^{-4}$	$(2.345 \pm 0.043)\times10^{-4}$
Discrete	$(1.027 \pm 0.041)\times10^{-3}$	$(1.497 \pm 0.096)\times10^{-3}$	$(5.819 \pm 0.026)\times10^{0}$
Gaussian	$(1.347 \pm 0.088)\times10^{-3}$	$(2.018 \pm 0.042)\times10^{-3}$	$(6.269 \pm 0.104)\times10^{0}$
Left skewed	$(2.212 \pm 0.057)\times10^{-3}$	$(3.322 \pm 0.096)\times10^{-3}$	$(6.399 \pm 0.013)\times10^{0}$
Right skewed	$(2.364 \pm 0.387)\times10^{-3}$	$(3.384 \pm 0.194)\times10^{-3}$	$(6.486 \pm 0.200)\times10^{0}$
Uniform	$(3.101 \pm 0.072)\times10^{-3}$	$(4.841 \pm 0.139)\times10^{-3}$	$(6.465 \pm 0.054)\times10^{0}$

Distribution	Merge-Sort	Quick-Sort	Randomized-Quick-Sort
Degenerate	$(1.304 \pm 0.031)\times10^{-2}$	$(6.488 \pm 0.119)\times10^{0}$	$(6.437 \pm 0.050)\times10^{0}$
Discrete	$(1.542 \pm 0.036)\times10^{-2}$	$(6.446 \pm 0.027)\times10^{-1}$	$(6.426 \pm 0.026)\times10^{-1}$
Gaussian	$(1.559 \pm 0.042)\times10^{-2}$	$(3.721 \pm 0.060)\times10^{-1}$	$(3.788 \pm 0.138)\times10^{-1}$
Left skewed	$(1.700 \pm 0.030)\times10^{-2}$	$(7.161 \pm 0.109)\times10^{-2}$	$(7.115 \pm 0.092)\times10^{-2}$
Right skewed	$(1.712 \pm 0.041)\times10^{-2}$	$(7.551 \pm 0.882)\times10^{-2}$	$(7.731 \pm 1.308)\times10^{-2}$
Uniform	$(1.762 \pm 0.036)\times10^{-2}$	$(1.398 \pm 0.021)\times10^{-2}$	$(1.394 \pm 0.034)\times10^{-2}$

Distribution	Heap-Sort	Radix-Sort	Bucket-Sort
Degenerate	$(1.137 \pm 0.010)\times10^{-3}$	$(6.288 \pm 0.082)\times10^{-3}$	$(1.133 \pm 0.008)\times10^{-2}$
Discrete	$(1.376 \pm 0.004)\times10^{-2}$	$(6.301 \pm 0.024)\times10^{-3}$	$(1.316 \pm 0.008)\times10^{-2}$
Gaussian	$(1.587 \pm 0.023)\times10^{-2}$	$(6.213 \pm 0.013)\times10^{-3}$	$(1.339 \pm 0.022)\times10^{-2}$
Left skewed	$(1.836 \pm 0.185)\times10^{-2}$	$(6.369 \pm 0.065)\times10^{-3}$	$(1.463 \pm 0.102)\times10^{-2}$
Right skewed	$(1.781 \pm 0.041)\times10^{-2}$	$(6.281 \pm 0.163)\times10^{-3}$	$(1.837 \pm 0.508)\times10^{-2}$
Uniform	$(1.964 \pm 0.079)\times10^{-2}$	$(6.378 \pm 0.344)\times10^{-3}$	$(1.572 \pm 0.136)\times10^{-2}$

绘制得不同算法, 不同数据分布下的运行时间柱状图如下. 其中, 纵轴采用对数坐标.

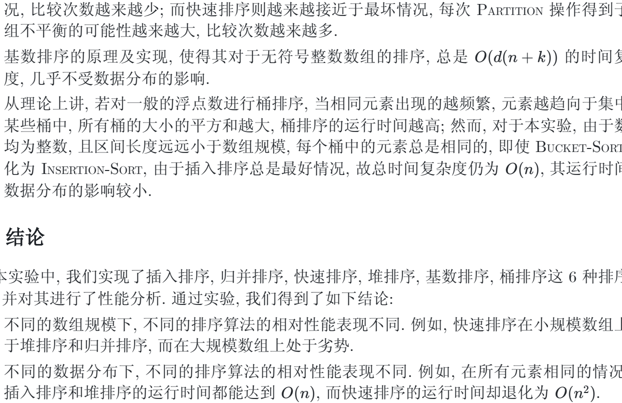

Figure 6. Time cost of different sorting algorithms with different data distributions.

根据图 6, 作如下观察:

1. 退化分布属于插入排序的最好情况, 此时 Insertion-Sort 的时间复杂度为 $O(n)$, 运行时间极短; 堆排序同样处于最好情况, 运行时间为 $O(n)$, 比其他情况显著更快; 快速排序则为最坏情况, 运行时间为 $O(n^2)$, 显著慢于其他算法.
2. 整数数组中相同元素的个数, 极大的影响了排序算法的运行时间. 总体上看, 相同元素出现的越频繁, 插入排序, 归并排序, 堆排序的运行时间越短, 快速排序的运行时间越长. 从直觉上讲, 这是由于随着相同元素出现的越频繁, 插入排序, 归并排序, 堆排序越来越接近于最好情况, 比较次数越来越少; 而快速排序则越来越接近于最坏情况, 每次 Partition 操作得到子数组不平衡的可能性越来越大, 比较次数越来越多.
3. 基数排序的原理及实现, 使得其对于无符号整数数组的排序, 总是 $O(d(n+k))$ 的时间复杂度, 几乎不受数据分布的影响.
4. 从理论上讲, 若对一般的浮点数进行桶排序, 当相同元素出现的越频繁, 元素越趋向于集中在某些桶中, 所有桶的大小的平方和越大, 桶排序的运行时间越高; 然而, 对于本实验, 由于数据均为整数, 且区间长度远远小于数组规模, 每个桶中的元素总是相同的, 即使 Bucket-Sort 退化为 Insertion-Sort, 由于插入排序总是最好情况, 故总时间复杂度仍为 $O(n)$, 其运行时间受数据分布的影响较小.

5 结论

本实验中, 我们实现了插入排序, 归并排序, 快速排序, 堆排序, 基数排序, 桶排序这 6 种排序算法, 并对其进行了性能分析. 通过实验, 我们得到了如下结论:

1. 不同的数组规模下, 不同的排序算法的相对性能表现不同. 例如, 快速排序在小规模数组上优于堆排序和归并排序, 而在大规模数组上处于劣势.
2. 不同的数据分布下, 不同的排序算法的相对性能表现不同. 例如, 在所有元素相同的情况下, 插入排序和堆排序的运行时间都能达到 $O(n)$, 而快速排序的运行时间却退化为 $O(n^2)$.
3. 不同的数据类型下, 不同的排序算法的相对性能表现不同. 例如, 当对大规模无符号整数数组进行排序时, 基数排序的运行时间总是 $O(d(n+k))$, 显著优于其他算法, 但该算法不能应用于浮点数.
4. 实际选择排序算法时, 应根据具体的应用场景, 选择合适的算法.