Analysis of Sort Algorithms

Kexin Ding
09/10/2019

1. Empirical analysis

1) The input size changes from 100 to 10,000, and running times are calculated by nanosecond.

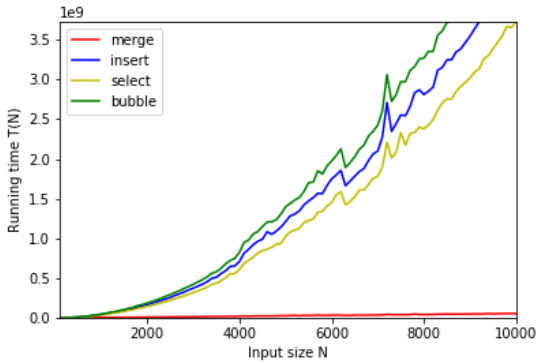

Fig 1. The average case of each sort algorithm

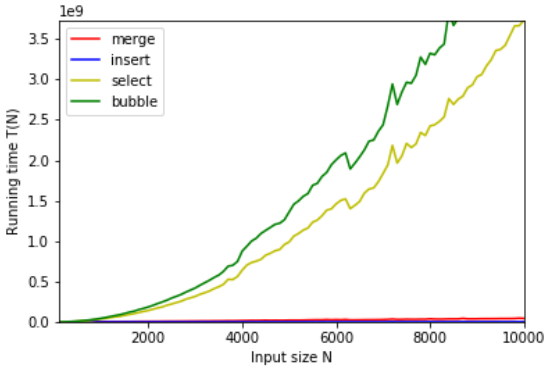

Fig 2. The best case of each sort algorithm

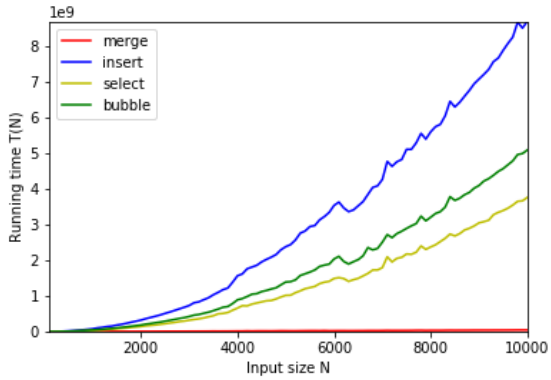

Fig 3. The worst case of each sort algorithm

2) Code for Bubble Sort(Python)

```
def BubbleSort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1] :
                tmp = arr[j]
                arr[j] = arr[j+1]
                arr[j+1] = tmp
```

2. Asymptotic analysis (theoretical analysis)

	Average Case	Best Case	Worst Case
Bubble Sort	O(N^2)	O(N^2)	O(N^2)
Insertion Sort	O(N^2)	O(N)	O(N^2)
Selection Sort	O(N^2)	O(N^2)	O(N^2)
Merge Sort	O(N*logN)	O(N*logN)	O(N*logN)

Table 1. the order of growth function for each sort algorithm

Explanation for Table 1:

1) Bubble Sort and Merge Sort: both of those algorithms will spend the same comparing time in their sort process and ignore the different input size.

2) Insertion Sort:

- The best case will occur when the array is sorted. The algorithm needs to
- The worst case will occur when a sorted array is reversed. The algorithm needs to compare N + (N-1) + (N-2) + … + 1 = O(N^2) times.
- The average case will occur when an array is generated randomly. The algorithm needs to compare N + (N-1) + (N-2) + … + 1 = O(N^2) times.

3) Selection Sort: the algorithm ignores the different input size and traverses each element in arrays no matter what the array looks like(sorted, reversed, and random generated).