Nghi Nguyen

CS146-section 3

Assignment 2 Report

Sorting algorithms: Insertion Sort, Heap sort

Bonus problem: Quick Sort (fixed pivot and random pivot).

I. Experiment

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | InsertionSort  Θ(n2) | HeapSort  Θ(n lgn) | QuickSort Fixed Pivot O(nlgn) | QuickSort RandomPivot O(nlgn) |
| n = 10 | < 1 milliseconds | < 1 milliseconds | 1 milliseconds | 1 milliseconds |
| n = 100 | 2 milliseconds | 2 milliseconds | 2 milliseconds | 3 milliseconds |
| n = 1,000 | 8 milliseconds | 5 milliseconds | 5 milliseconds | 5 milliseconds |
| n = 10,000 | 198 milliseconds | 12 milliseconds | 49 milliseconds | 48 milliseconds |
| n = 100,000 | 11,606 milliseconds | 43 milliseconds | 282 milliseconds | 257 milliseconds |
| n = 1,000,000 | 1,137,415 milliseconds ~ 19 minutes | 320 milliseconds | 27,138 milliseconds | 24,190 milliseconds |

II. Comparing

* 1. All 4 sorting algorithms have almost the same running time with small inputs (n <= 100).
  2. InsertionSort Θ(n2)
     1. Effective for small inputs and almost sorted array
     2. the running time escalates quickly. So, in general, insertion sort is not effective.

3. HeapSort Θ(n lgn)

1. The running time slowly increase

2. Heapsort is the most effective algorithm among 4 chosen sorting algorithm since even in the worst case, the running time still Θ(n lgn)

4. QuickSort

1. quick sort running time is O(n lgn) for average case and O(n2) on the worst case. Therefore, quick sort take more time than heap sort

2. In the worst case, every element is smaller than or greater than the pivot. So, by choosing random pivot results in slightly better running times than choosing a fixed pivot position.

III. Conclusion

1. Sorting almost sorted array or array with small size

- Effective Algorithm: InsertionSort

2. Sorting large array

- Effective Algorithm: HeapSort, QuickSort

- Ineffective Algorithm: InsertionSort, QuickSort may be an ineffective algorithm in the case of almost sorted or sorted array