Marmara University Department of Computer Engineering



CSE246 Homework #2

Subject : Analysis of Sort Algorithms

Experiment : Insertion, Quick, Merge ,Tree Sort Comparing Performance

**CONTENT**

1. Analysis and Information of Sorting Algorithms

A. Insertion Sort

A.1.Best Case for Insertion Sort

A.2. Worst Case for Insertion Sort

A.3. Average Case for Insertion Sort

B. Merge Sort

C. Quick Sort

D. Tree Sort ( binary search tree)

2. Several Inputs

Different Sizes of Inputs for a Sorting Algorithm

Same Size of Inputs for Different Sorting Algorithms

3.Results

4.Comparing Performance

5.Comparing Empirical with Theoretical Results

6. Conclusion

1. Analysis and Information of Sorting Algorithms

A.Insertion Sort

Insertion sort is a simple sorting algorithm that builds the final sorted linked list (or list) one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort. However, insertion sort provides several advantages:

\* Simple implementation: Bentley shows a three-line C version, and a five-line optimized version.

\*Efficient for (quite) small data sets .

\* More efficient in practice than most other simple quadratic (i.e., O(n2)) algorithms such as selection sort or bubble sort

\*Adaptive, i.e., efficient for data sets that are already substantially sorted: the time complexity is O(nk) when each element in the input is no more than k places away from its sorted position

\*Stable; i.e., does not change the relative order of elements with equal keys.

\*In-place; i.e., only requires a constant amount O(1) of additional memory space.

\*Online; i.e., can sort a list as it receives it.

When people manually sort something (for example, a deck of playing cards), most use a method that is similar to insertion sort.

A.1.Best Case for Insertion Sort

The best case input is an linked list that is already sorted. In this case insertion sort has a linear running time (i.e., O(n)). During each iteration, the first remaining element of the input is only compared with the right-most element of the sorted subsection of the linked list.

A.2. Worst Case for Insertion Sort

The simplest worst case input is an linked list sorted in reverse order. The set of all worst case inputs consists of all linked lists where each element is the smallest or second-smallest of the elements before it. In these cases every iteration of the inner loop will scan and shift the entire sorted subsection of the linked list before inserting the next element. This gives insertion sort a quadratic running time (i.e., O(n2)).

A.3. Average Case for Insertion Sort

The average case is also quadratic, which makes insertion sort impractical for sorting large linked lists. However, insertion sort is one of the fastest algorithms for sorting very small linked lists, even faster than quicksort; indeed, good quicksort implementations use insertion sort for linked lists smaller than a certain threshold, also when arising as subproblems; the exact threshold must be determined experimentally and depends on the machine, but is commonly around ten.

B.Merge Sort

Merge sort is based on the divide-and-conquer paradigm. Its worst-case running time has a lower order of growth than insertion sort. Since we are dealing with subproblems, we state each subproblem as sorting a small linked list. Initially, p = 1 and r = n, but these values change as we recurse through subproblems.

C. Quick Sort

The task is to sort a linked list elements using the quicksort algorithm. The elements must have a strict weak order and the size of linked list can be of any discrete type. For languages where this is not possible, sort a linked list of integers. Quicksort, also known as partition-exchange sort, uses these steps.

**1.** Choose any element of the linked list to be the pivot.

**2.** Divide all other elements (except the pivot) into two partitions.

-All elements less than the pivot must be in the first partition.

-All elements greater than the pivot must be in the second partition.

**3.** Use recursion to sort both partitions.

**4**. Join the first sorted partition, the pivot, and the second sorted partition.

The best pivot creates partitions of equal length (or lengths differing by 1).

The worst pivot creates an empty partition (for example, if the pivot is the first or last element of a sorted linked list ).

The runtime of Quicksort ranges from O(n log n) with the best pivots, to O(n2) with the worst pivots, where n is the number of elements in the linked list .

D. Tree Sort( binary search tree)

A tree sort is a sort algorithm that builds a binary search tree from the elements to be sorted, and then traverses the tree (in-order) so that the elements come out in sorted order. Its typical use is sorting elements adaptively: after each insertion, the set of elements seen so far is available in sorted order.

Adding one item to a binary search tree is on average an O(log n) process (in big O notation), so adding n items is an O(n log n) process, making tree sort a 'fast sort'. But adding an item to an unbalanced binary tree needs O(n) time in the worst-case, when the tree resembles a linked list (degenerate tree), causing a worst case of O(n²) for this sorting algorithm. This worst case occurs when the algorithm operates on an already sorted set, or one that is nearly sorted. Expected O(log n) time can however be achieved in this case by shuffling the linked list .

The worst-case behaviour can be improved upon by using a self-balancing binary search tree. Using such a tree, the algorithm has an O(n log n) worst-case performance, thus being degree-optimal for a comparison sort. When using a splay tree as the binary search tree, the resulting algorithm (called splaysort) has the additional property that it is an adaptive sort, meaning that its running time is faster than O(n log n) for inputs that are nearly sorted.

|  |  |
| --- | --- |
|  | |
| **Class** | [Sorting algorithm](https://en.wikipedia.org/wiki/Sorting_algorithm) |
| **Data structure** | Linked Lists |
| [**Worst case performance**](https://en.wikipedia.org/wiki/Best,_worst_and_average_case) | *O*(*n*²) (unbalanced)  *O*(*n* log *n*) (balanced) |
| [**Best case performance**](https://en.wikipedia.org/wiki/Best,_worst_and_average_case) | *O*(*n* log *n*)[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)] |
| [**Average case performance**](https://en.wikipedia.org/wiki/Best,_worst_and_average_case) | *O*(*n* log *n*) |
| [**Worst case space complexity**](https://en.wikipedia.org/wiki/Best,_worst_and_average_case) | Θ(*n*) |

2. SEVERAL INPUTS

In this experiment there are several inputs with different sizes and different sort types. There are two different criterion to compare sorting algorithms, one of them is to check same sorting algorithm with different sizes and sort types of inputs. The other one is to check each sorting algorithm with same sizes and sort types of inputs.

The first comparison gives us the information about what a sorting algorithm do when input size or sort type change and the information about the execution time, total basic operation number and comparison number. The second comparison gives us the information about what the sorting algorithms which we want to compare do with same inputs, which is the best algorithm according to execution time, base operations and used memory.

To understand the quick, merge,tree and insertion sorts about their execution time and total base operation number. For this experiment there are seven different size of inputs chosen and for each sizes of inputs there are four different sort type chosen.

Best Sorted

Increasing order variables located in increasing current address of linked list.

Worst Sorted

Decreasing order variables located in increasing current address of linked list.

Average Sorted

Located of numbers aren’t known by users and result is that program executes a few times.

Input Size Different input sizes for different execution time.

\*We execute program 30 times and took average for table.

The table 1.1 shown below, shows the input sizes and types to compare for Insertion sort algorithms.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input Size  Sort Type | 1000 | 5000 | 10000 | 25000 | 50000 | 100000 | 250000 | 500000 |
| Average Case Sorting | 0.002600 | 0.007200 | 0.029233 | 0.441967 | 2.636200 | 10.560233 | 64.874921 | 460.00000 |
| Best Case Sorting | 0.0001025 | 0.002001 | 0.004006 | 0.007680 | 0.015626 | 0.031254 | 0.062502 | 0.093658 |
| Worst Case  Sorting | 0.046877 | 0.094026 | 0.218758 | 4.154877 | 5.138879 | 20.561544 | 128.969779 | 338.286765 |

The table 1.2 shown below, shows the input sizes and types to compare for Merge sort algorithms.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input Size  Sort Type | 1000 | 5000 | 10000 | 25000 | 50000 | 100000 | 250000 | 500000 |
| Average Case Sorting | 0.000500 | 0.002100 | 0.004667 | 0.015467 | 0.041230 | 0.073340 | 0.084134 | 0.094526 |
| Best Case Sorting | 0.001033 | 0.004667 | 0.006767 | 0.016667 | 0.033333 | 0.057833 | 0.176067 | 0.360967 |
| Worst Case  Sorting | 0.000002 | 0.000533 | 0.004167 | 0.013567 | 0.031456 | 0.064789 | 0.105825 | 0.176789 |

The table 1.3 shown below, shows the input sizes and types to compare for Quick sort algorithms.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input Size  Sort Type | 1000 | 5000 | 10000 | 25000 | 50000 | 100000 | 250000 | 500000 |
| Average Case Sorting | 0.001033 | 0.001617 | 0.003002 | 0.014001 | 0.025018 | 0.029029 | 0.074794 | 0.155103 |
| Best Case Sorting | 0.002003 | 0.009005 | 0.017011 | 0.051460 | 0.079093 | 0.163108 | 0.450298 | 2.033356 |
| Worst Case  Sorting | 0.005023 | 0.010009 | 0.031022 | 0.403602 | 0.662462 | 3.094049 | 7.023600 | 13.41564 |

The table 1.4 shown below, shows the input sizes and types to compare for Tree sort algorithms.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input Size  Sort Type | 1000 | 5000 | 10000 | 25000 | 50000 | 100000 | 250000 | 500000 |
| Average Case Sorting | 0.000533 | 0.003133 | 0.005733 | 0.014067 | 0.033367 | 0.062400 | 0.168800 | 0.301933 |
| Best Case Sorting | 0.000001 | 0.000533 | 0.000538 | 0.000600 | 0.001033 | 0.00260 | 0.004667 | 0.010933 |
| Worst Case  Sorting | 0.001033 | 0.003100 | 0.005200 | 0.014600 | 0.028633 | 0.054200 | 0.128467 | 0.146800 |

Different Sizes of Inputs for a Sorting Algorithm

The inputs are chosen to see the difference between the execution time of a sorting algorithm. There can be an experiment with a linked list type T1, linked list size S1 and sorting algorithm A1. If we calculate the execution time of sorting algorithm A1 for this variables, we can find the execution time TM1 and base operation number B1.

T1 + S1 + A1 -> TM1 + B1

T1 + S2 + A1 -> TM2 + B2

T? + S? + A1 -> TM? + B?

After this experiment

If we just change the input size, there can be a new execution time and base operation number for the sorting algorithm S1

If we just change the sort type, there can be a new execution time and base operation number for the sorting algorithm S1

According to this opinion, if we change the input size or sort type for a sorting algorithm. We can understand the change of execution time and base operation number of the choosen sorting algorithm.

Same Size of Inputs for Different Sorting Algorithms

Use of an input with a size S1 and sort type T1, to compare different sorting algorithms (A1,A2,A3….) can show us the execution times and base operation numbers of sorting algorithms with this input to compare the algorithms.

T1 + S1 + A1 TM1 + B1

T1 + S1 + A2 TM2 + B2

T1 + S1 + A? TM? + B?

According to this opinion, if we use the same input for all sorting algorithms. We can compare the outputs of the algorithm (Execution time, Base op. Num.) to each other to see which algorithm is best.

3. Results

4. Comparing Performance

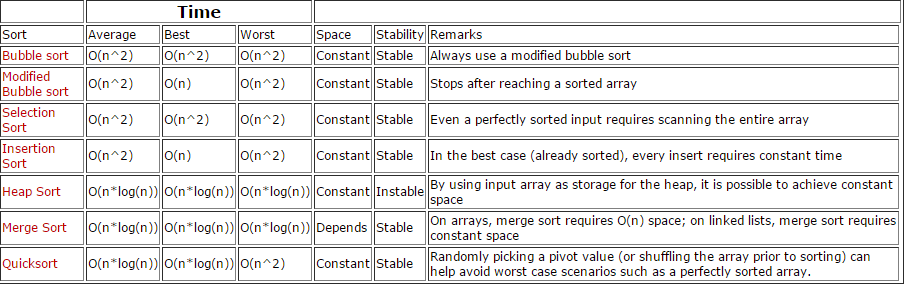
For random type of linked list with different input sizes, there are some different situations. If there is a linked list with small input sizes. All the sort algorithms are useful but if there is a larger linked list, the insertion sort is very bad. The merge and quick sorts are useful for random type and larger size of linked list . Tree sort is very good for larger list.

For already sorted linked lists with different sizes, the situation is changed. Because in insertion sort the sorting process is going one by one. There is no need to do anything because of the linked list is already sorted. So the insertion sort is already good. Merge is also useful for already sorted linked lists. But the Quick sort with pivot is first element is not useful for sorted linked lists. Because the algorithm choose the pivot as first element and for this pivot it checks all the elements of the linked lists in decreasing order and finds the right place for pivot. But pivot is already in right place so it makes more comparisons for all pivots.

For reverse sorted linked lists with different sizes, Merge sorted is useful. The Insertion sort is not useful than merge sort but it is useful than Quick with pivot first element.

For average situations of sorting algorithm, The Merge has best performance. The Quick with Pivot is First have good performance in small sizes of inputs. Getting larger input sizes for linked lists, the performance of the quick sort(pivot is first) goes down. The Insertion Sort have the worst performance

For best and worst case tree sort algorithm is very fast than quick sort but fo average case quick sort is better. So we can say quick is fastest for average case.Then , tree is fastest for best and worst and finally insertion.

5. Comparing Empirical with Theoretical Results

1. Insertıon Sort comparing

In theory the insertion sort has O(n) for best case..As the theoretical results, the empirical results have same situation. In theory the insertion sort has O(n^2) for worst case. We can check the cases with base operation number for insertion sort because of the one by one process.

If we can check the plot for insertion sort,there is a similarity with empirical and theoretical results.

The average case has same growth rate with the worst case situation. It is similar to theoratical results.

1. Merge Sort comparing

Merge sort have same growth rate O(n log (n)) in all cases in theory. If we check the plot we can see the results of the best and worst cases of merge sort. The growth rate of the best and worst cases are same and it is similar to theoretical results and if we check the base operation numbers of worst and best cases, we can see the similar growth rate for each situation.

For average case of merge sort. We can check the plot . The growth rate of the average case is almost same with best and worst case and it is similar to theoratical results.

1. Quick Sort comparing

The best and worst cases for this sorting algorithm. the best and worst cases have same growth rate and it is similar to theoratical results. For the average case we can check the plot . It has a growth rate same with theoratical results.

D. Tree Sort comparing

The best and worst cases for this sorting algorithm. the best and worst cases and also average case have same growth rate and it is similar to theoratical results. O(n log n). We can see in data table this changings.

6. Conclusion

All the algorithms have different performances for different situations.

The insertion sort is really good for really small samples or sorted input situations. But if input size grows, the performance of the insertion sort is going to be worst. If we have a very small or sorted linked lists. We can use the insertion sort.

The quick sort algorithm also have good performance for more situations. The method to choose pivot is really important. Because the pivot makes decision on comparison number. For a sorted linked lists the pivot chosen as first element, makes the process so long. For random inputs each of the methods have good performance.

The merge sort is very useful for all situations.In all cases it uses the same process (divide and conquer). So for same sizes of linked lists, it makes same base operations.

In quick sort for each method to choose pivot. The sorted linked list is not best case.

As a result, there is no rule to choose a sorting algorithm for an input. Because everything can change with your input.

In binary tree sort for best and worst case algorithm is very fast but for average case ,performance is being low .

Source for theoretical information : https://en.wikipedia.org/wiki