Advances in Computational Research
ISSN: 0975-3273 & E-ISSN: 0975-9085, Volume 7, Issue 1, 2015, pp.-182-184.
Available online at http://www.bioinfopublication.org/jouarchive.php?opt=&jouid=BPJ0000187

ANALYSIS OF INSERTION SORT IN JAVA

SAMEE S., CHITTE A.R.* AND TANGDE Y.S.

MCA Department, Marathwada Institute of Technology, Aurangabad- 431 028, MS, India.
*Corresponding Author: Email- amol.chitte@gmail.com

Received: December 18, 2014; Revised: January 05, 2015; Accepted: January 15, 2015

Abstract- An algorithm is any well-defined procedure or set of instructions, that takes some input in the form of some values, processes them and gives some values as output. An Insertion sort algorithms has been developed to enhance the performance in terms of computational complexity, memory and other factors. This paper is an attempt to analysis the performance of insertion sort algorithm: The paper implements the insertion sort, complexity and analysis with different input elements. The paper contains code of insertion sort for inputting the values requires no. of elements, create random generator function it generate random numbers and measures the running time of program.

Keywords- Algorithm, Time Complexity Sorting, Insertion Sort, Run-time Analysis

Citation: Samee S., Chitte A.R. and Tangde Y. (2015) Analysis of Insertion Sort in Java. Advances in Computational Research, ISSN: 0975-3273 & E-ISSN: 0975-9085, Volume 7, Issue 1, pp.-182-184.

Introduction

Algorithm is an unambiguous, step-by-step procedure for solving a problem, which is guaranteed to terminate after a finite number of steps. In computer science, sorting is one of the most extensively researched subjects because of the need to speed up the operation on thousands or millions of records during a search operation. Arranging the data in either ascending or descending order based on certain key in the record is known as sorting. Many times we are supposed to arrange the data in some order according to the requirement. Sorting is generally understood to be the process of rearranging a given set of objects in a specific order and therefore, the analysis and design of useful sorting algorithms has remained one of the most important research areas in the field. Insertion sort is one of the algorithm for sorting, it is a simple sorting algorithm that builds the final sorted array or list one item at a time.

Working Procedure of Insertion Sort

An insertion sort sorts records by inserting records into an existing sorted file. It is just like while playing cards we pick one card at a time and place it in proper position in the card held in hand. Whenever a new element is to be inserted its position is found, the elements next to this element position are shifted to right so that this element can be inserted in the proper position.

The insertion sort works just like its name- it inserts each item into its proper place in the final list. In Insertion sort, the first iteration starts with comparison of 1st element with 0th element. In the second iteration the element is compared with 0th and 1st element. In general in every iteration an element is compared with all elements. If at same point it is found the element can be inserted at a position then space is created for it by shifting the other element one position right and inserting the element at the suitable position. This procedure is repeated for all the element in the array.

Algorithm

The algorithm for insertion sort having DATA as an array with N elements is as follows:

Insertion (Data, N)

- Set A[0] = -∞ [Initializes sentinel elements]
- Repeat Steps 3 to 5 for K = 2, 3,…N
- Set TEMP = DATA[K] and PTR = K-1
- Repeat while TEMP < DATA[PTR] :
  Set DATA[PTR+1] = A[PTR]
  [Moves element forward]
  Set PTR =PTR – 1
  [End of loop]
- Set DATA[PTR+1] = TEMP
  [Inserts the elements in proper place] [End of step 2 loop]
- Exit.

Analysis

Time complexity is sum of compilation time and execution time of an algorithm. The compile time do not depend on instance characteristics also once compiled program can be executed several times without recompilation. Since the running time of an algorithm on a particular input is the number of steps executed, we must define "step" independent of machine. We say that a statement that takes *ci* steps to execute and executed *n* times contributes *ci*n* to the total running time of the algorithm. To compute the running time, T (*n*), we sum the products of the cost and times column. That is, the

running time of the algorithm is the sum of running times for each statement executed.

So, we have

$T(n) = c1n + c2\ (n-1) + 0(n-1) + c4\ (n-1) + c5\ \Sigma 2 \le j \le n\ (\ tj\ ) + c6\ \Sigma 2 \le j \le n\ (tj-1) + c7\ \Sigma 2 \le j \le n\ (tj-1) + c8\ (n-1)$ (1)

In the above equation we supposed that *tj* be the number of times the while-loop (in line 5) is executed for that value of *j*. Note that the value of *j* runs from 2 to $(n-1)$.

We have

$T(n) = c1n + c2\ (n-1) + c4\ (n-1) + c5\ \Sigma 2 \le j \le n\ (\ tj\ ) + c6\ \Sigma 2 \le j \le n\ (tj-1) + c7\ \Sigma 2 \le j \le n\ (tj-1) + c8\ (n-1)$ (2)

Table 1- Passes of Insertion sort

Elements	0	1	2	3	4	5	6	7	8	9
Data	22	79	47	13	74	36	21	94	56	60
1st Pass	22	79	47	13	74	36	21	94	56	60
2nd Pass	22	47	79	13	74	36	21	94	56	60
3rd Pass	13	22	37	79	74	36	21	94	56	60
4th Pass	13	22	37	74	79	36	21	94	56	60
5th Pass	13	22	36	37	74	79	21	94	56	60
6th Pass	13	21	22	36	37	74	79	94	56	60
7th Pass	13	21	22	36	37	74	79	94	56	60
8th Pass	13	21	22	36	37	56	74	79	94	60
Sorted	13	21	22	36	37	56	60	74	79	94

Best-Case Analysis

The best case occurs if the array is already sorted. For each value of $j = 2, 3, ..., n$, we find that $A[i]$ is less than or equal to the key when *i* has its initial value of $(j-1)$. In other words, when $i = j-1$, always find the key $A[i]$ upon the first time the WHILE loop is run. Therefore, $tj = 1$ for $j = 2, 3, ..., n$ and the best-case running time can be computed using equation (2) as follows:

$T(n) = c1n + c2\ (n-1) + c4\ (n-1) + c5\ \Sigma 2 \le j \le n\ (1) + c6\ \Sigma 2 \le j \le n\ (1-1) + c7\ \Sigma 2 \le j \le n\ (1-1) + c8\ (n-1)$

$T(n) = c1n + c2\ (n-1) + c4\ (n-1) + c5\ (n-1) + c8\ (n-1)$

$T(n) = (c1 + c2 + c4 + c5 + c8\ )\ n + (c2 + c4 + c5 + c8)$

This running time can be expressed as $an + b$ for constants *a* and *b* that depend on the statement cost *ci*. Therefore, $T(n)$ it is a linear function of *n*.

The main concept here is that the while-loop in line 5 executed only once for each *j*.

This happens if given array *A* is already sorted.

$T(n) = an + b = O(n)$ It is a linear function of *n*.

Worst-Case Analysis

The worst-case occurs if the array is sorted in reverse order i.e., in decreasing order. In the reverse order, we always find that $A[i]$ is greater than the key in the while-loop test. So, we must compare each element $A[j]$ with each element in the entire sorted sub array $A[1 .. j-1]$ and so $tj = j$ for $j = 2, 3, ..., n$. Equivalently, we can say that since the while-loop exits because *i* reaches to 0, there is one additional test after $(j-1)$ tests. Therefore, $tj = j$ for $j = 2, 3, ..., n$ and the worst-case running time can be computed using equation (2) as follows:

$T(n) = c1n + c2\ (n-1) + c4\ (n-1) + c5\ \Sigma 2 \le j \le n\ (\ j\ ) + c6\ \Sigma 2 \le j \le n(j-1) + c7\ \Sigma 2 \le j \le n(j-1) + c8\ (n-1)$

And using the summations, we have:

$T(n) = c1n + c2\ (n-1) + c4\ (n-1) + c5\ \Sigma 2 \le j \le n\ [n(n+1)/2 + 1] + c6\ \Sigma 2 \le j \le n\ [n(n-1)/2] + c7\ \Sigma 2 \le j \le n\ [n(n-1)/2] + c8\ (n-1)$

$T(n) = (c5/2 + c6/2 + c7/2)\ n2 + (c1 + c2 + c4 + c5/2 - c6/2 - c7/2 + c8)\ n - (c2 + c4 + c5 + c8)$

This running time can be expressed as $(an2 + bn + c)$ for constants *a*, *b*, and *c* that again depends on the statement costs *ci*.

Therefore, $T(n)$ is a quadratic function of n. Here the main concept is that the worst-case occurs, when line 5 executed *j* times for each *j*.

This can happens if array *A* starts out in reverse order

$T(n) = an2 + bn + c = O(n^2)$

It is a quadratic function of n^2.

The implementation of insertion Sort shows that there are (n-1) passes to sort *n*. The iteration starts at position 1 and moves through position (n-1), as these are the elements that need to be inserted back into the sorted sub lists. The maximum number of comparisons for an insertion sort is $(n-1)$ .Total numbers of comparisons are:

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

Best Case = O(n)

Average Case= $O(n^2)$

Worst Case= $O\ (n^2)$

Pros

- Insertion sort exhibits a good performance when dealing with a small list.
- The insertion sort is an in-place sorting algorithm so the space requirement is minimal.

Cons

- Insertion sort is useful only when sorting a list of few elements
- The insertion sorts repeatedly scans the list of elements each time inserting the elements in the unordered sequence into its correct position.

Code Written in Java Language

Now, we will determine the efficiency of the insertion sorting algorithms according to the time by using randomized trails. The elements generated using the random() function in Java Language. We discuss and implements insertion sort and also include complexity of insertion. We represent this algorithms as a way to sort an array or integers and run random trails of length. The research will provide the runtime of sorting algorithm.

To investigate, we create a class in Java named INSERTION-SORT . This class contains the main function and called 3 functions i.e populateArray(),printArray() and insertionSort()

- populateArray() generate the random numbers of elements
- insertionSort() build the actual code for insertion sort
- printArray() print the elements

```
//this is the function that generate random numbers
public static void populateArray(int[] B) {
        for (int i = 0; i < B.length; i++) {
                B[i] = (int) (Math.random() * 1000);} }
```

```
//actual code for insertion sort
private static void insertionSort(int[] arr) {
        for (int i = 1; i < arr.length; i++) {
                int valueToSort = arr[i];
                int j = i;
                while (j > 0 && arr[j - 1] > valueToSort) {
        arr[j] = arr[j - 1];
j--; }
        arr[j] = valueToSort; } }
```

Table 2- Complexity of Insertion sort

Time						
Sort	Avg.	Best	Worst	Space	Stability	Remarks
Insertion Sort	O(n^2)	O(n)	O(n^2)	Constant	Stable	In the best case (already sorted), every insert requires constant time

Table 3- Running time of Insertion sort on integers

No. of Elements	Running Time (sec)
1000	0.968
1500	0.998
2000	1.056
2500	1.189
3000	1.197
3500	1.356
4000	1.398
4500	1.437
5000	1.496

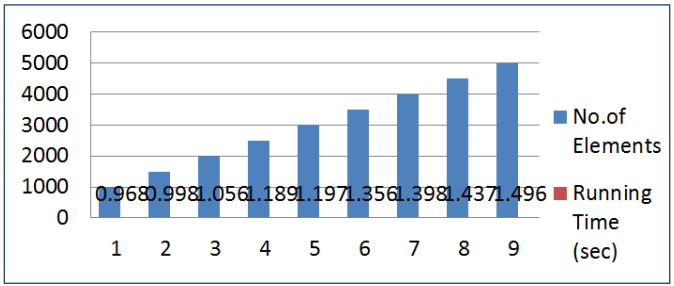

Fig. 1- Time Performance of insertion sort

Performance Analysis of Insertion Sort

The following table gives the running time of insertion sort for integer array .It is obtained by running the above code in the different input elements executed. Time selection is the time in the seconds. we will be calling sorting function to find the running time of insertion code so that we calculate the running time of the different input. For this passed different number of elements (i=1000, 1500, 2000, 2500,.....) to the sorting functions, run the program 10 times for each value i (i.e.1000, 1500,2000, 2500,....) and tried to find the running time for each input. Table shows running time of algorithm for different inputs run.

Conclusion

In this study we have studied about insert sort algorithms and its complexity. There is advantage and disadvantage of algorithm. To find the running time of algorithm for different input elements we used one program and measures running time (in seconds). After running the same program on 9 different runs (for each different value of I=1000, 1500, 2000, 2500 and so on), we calculated running time for each run and then showed the result with the help of a chart. From the chart we can conclude that insertion sort is the efficient algorithm gives relevant results for larger lists.

Conflicts of Interest: None declared.

References

[1] Sareen P. (2013) *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(3), 522-532.

[2] Chhajed N., Uddin I. & Bhatia S.S. (2013) *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(3), 373-381.

[3] Cormen T.H., Leiserson C.E., Rivest R.L. & Stein C. (2001) *Introduction to algorithms*, 2, 531-549.

[4] Sahni S., Horowitz E. & Rajasekaran S. (1998) *Computer Algorithms*, WH Freeman Press.

[5] Beniwal S. & Grover D. (2013) *International Journal of Emerging Research in Management &Technology*, 2(2), 83-88.

[6] Knuth D.E. (2014) *Art of Computer Programming*, Seminumerical Algorithms, Addison-Wesley Professional.

[7] Cormen T.H., Leiserson C.E., Rivest R.L. & Stein C. (2001) *Introduction to algorithms*, 2, 531-549.

[8] Aliyu A.M. & Zirra D.P. (2013) *The International Journal of Engineering and Science*, 2(7), 25-30.

[9] Adhikari P. (2007) *Review on Sorting Algorithms*, *A comparative study on two sorting algorithm*, Mississppi state university.

[10] Al-Kharabsheh K.S., AlTurani I.M., AlTurani A.M.I. & Zanoon N.I. (2013) *International Journal of Computer Science and Security*, 7(3), 120.

[11] Abhyankar D. & Ingle M. (2011) *International Journal of Computer Science & Engineering Technology*, 1(3), 134-136.