European Journal of Academic Essays, Special Issue (1): 1-4, 2014
ISSN: 2183-1904
www.euroessays.org

Recent Trends in Analysis of Algorithms and Complexity Theory

Performance Analysis of Bubble Sort and Insertion Sort With Computational Complexity

< Mishra Sambit Kumar >[1], < Patnaik Dulu(Dr.) >[2] , < Mishra Bibhuti Bhusan(Dr.) >[3]
1 Associate professor, Department of Computer Sc.& Engg.
Ajay Binay Institute of Technology, Cuttack
sambit_pr@rediffmail.com

2 Principal, Government College of Engineering, Bhawanipatna
patnaik_d@yahoo.com

3 Asst. Professor, Institute of Business & Computer Studies
Siksha 'O' Anusandhan University, Bhubaneswar
bbmishra@hotmail.com

Abstract: The complexity of a problem, algorithm or structure, and to obtain bounds for complexity arises in computer science. The traditional branches of mathematics, statistical physics, biology, medicine, social sciences and engineering are also confronted more and more frequently with this problem. In the approach taken by computer science, complexity is measured by the quantity of computational resources, e.g. time, storage, program, communication used up by a particular task. Many results on the computational complexity of sorting algorithms are usually obtained using incompressibility method. Usually hard average-case analysis is applicable to this method.

The results about bubble sort, insertion sort in sequential or parallel mode are being already surveyed. Especially in the case of bubble sort the uses of incompressibility easily resolved problems that had stayed open for a long time. Also in the case of insertion sort it is also possible to resolve problems by applying incompressibility method.

Keywords: Sorting, Incompressibility, Complexity, Average case analysis, Bubble sort, Shaker Sort, Insertion Sort

1. Introduction

In bubble sort it compares adjacent array elements and exchanges their values if they are out of order. The smaller values bubble up to the top of the array and larger values sink to the bottom.

The bubble sort provides excellent performance in some cases and very poor performances in some cases. It works best when array is already nearly sorted. The worst case number of comparisons is $O(n^2)$. The worst case number of exchanges is $O(n^2)$. The best case occurs when the array is already sorted, e.g. O(n) comparisons and O(1) exchanges. In insertion sort it sorts the elements in range [0,m] for m = 0,..., n−1 . In this case no action is needed for m=0. While going from m to m+1, it inserts the element in index m+1, to its appropriate location. The number of comparisons is $O(n^2)$. The last condition can be checked by remembering whether any exchange has occurred during the execution of the for-loop. Maximum number of comparisons is $O(n^2)$. In the best case, number of comparisons is O(n). The number of shifts performed during an insertion is one less than the number of comparisons or, when the new value is the smallest so far, the same as the number of comparisons. A shift in an insertion sort requires the movement of only one item whereas in a bubble or selection sort an exchange involves a temporary item and requires the movement of three items.

2. Review of Literature

W.Dobosiewicz et.al[1] have already proposed to use only one pass of bubble sort on each subsequence instead of sorting the subsequences at each stage.

They have also used two passes of bubble sort in each stage, one going left-to-right and the other going right-to-left, which is called as shaker sort. It reminds one of shaking a cocktail. In both cases the sequence may be unsorted, even if the last increment is 1. A final phase, a straight insertion sort, is required to guaranty a fully sorted list.

Knuth et.al[2] have discussed in their paper that the bubble sort seems to have nothing to recommend , except a catchy name and the fact that it leads to some interesting theoretical problems. Although bubble sort may not be a best practice sort, perhaps the weight of history is more than enough to compensate and provide for its longevity.

Sedgewick et.al[3]have discussed in their paper that bubble sort runs in O(n) time on sorted data and works well on nearly sorted data. But the insertion sort is better than bubble sort, is stable.

Klerlein et.al[4] have discussed in their paper that bubble sort is the best known sort, seems

Corresponding Author :Mishra Sambit Kumar

relegated to the status of an example of inefficiency. Bubble sort is covered in many texts, occasionally as the only O(n^2) sort, but often compared to another sort like insertion or selection.

Dale et.al[5] have discussed in their paper that in a popular new breadth-first text , bubble sort is given equal footing with selection sort and quick sort in online student exercises.

Cooper et.al[6] have stated in their paper that the bubble sort is worse than selection sort for a jumbled array. It may require many more component exchanges. But it is just as good as insertion sort for a well ordered array.

Sedgewick et.al[7] have already reflected in their paper about bubble sort‟s prime virtue, that it is easy to implement, but whether it is actually easier to implement than insertion or selection sort is arguable. The bubble sort algorithm is not very useful in practice, since it runs more slowly than insertion sort and selection sort, yet is more complicated to program.

A. V. Aho et.al[8] have discussed in their paper about the efficiency of an algorithm that depends on its use of resources, such as: the time it takes the algorithm to execute, the memory it uses for its variables, the network traffic it generates and the number of disk accesses it makes.

M. T. Goodrich et.al[9] have focused in their paper the technique to obtain cheap-to-compute rough-and-ready formulae. It means that when two algorithms have even fairly similar time complexities, the firm conclusions may be drawn about which is the more efficient, and it may be needed to use benchmarking.

3. Example with analysis

An example of the application of the incompressibility method is the average-case analysis of bubble sort. It is well-known that bubble sort uses $\Theta(n^2)$ comparisons/exchanges on the average. There are $n!$ different permutations. Given the sorting process one can recover the correct permutation from the sorted list. Hence one requires $n!$ pair wise different sorting processes. This gives a lower bound on the minimum of the maximal length of a process. The formulation of the proof in the crisp format of incompressibility was implemented. In bubble sort the passes are made from left to right over the permutation to be sorted and moved the currently largest element right by exchanges between it and the right-adjacent element. At most n- 1 passes are made, since after moving all but one element in the correct place the single remaining element must be also in its correct place. The total number of exchanges is obviously at most n^2, so we only need to consider the lower bound. Assume B be a bubble sort algorithm. For a permutation $\prod$ of the elements 1,.... n, the total number of exchanges may be described by $M := \Sigma$ i=1..n (mi) where mi is the initial distance of element n- i to its final position. Note that in every pass more than one element may bubble" right but that means simply that in the future passes of the sorting process an equal number of exchanges will be saved for the element to reach its final position. That is, every element executes a number of exchanges going right that equals precisely the initial distance between its start position to its final position.

4. Problem Formulation

4.1. Algorithm-1 : Bubble sort

```
do
for each pair of adjacent array elements if the
values in the pair are out of order then
exchange the two
while the array is not sorted
```

The number of comparisons is O(n^2).

```
void bubbleSort(int arr[ ]){ int i;
int j;
int temp;
for(i = arr[length]-1; i > 0; i--){ for(j =
0; j < i; j++){
if(arr[j] > arr[j+1]){ temp
= arr[j];
arr[j] = arr[j+1]; arr[j+1]
= temp; }//
}// end inner loop }// end
bubble sort
```

4.2. Algorithm 2 – Insertion sort

```
for  q = 1 to n do
// determine the position for the element at index m
itemcode =q−1
item = element at position m
while (itemcode >= 0) and (the element at index pos is
greater than item)
place the element at index pos to index itemcode+1
decrement itemcode
place item at index itemcode+1 for q =
1 to n do
// determine the position for the element at index m
itemcode = q−1
item = element at position q
while (itemcode >= 0) and (the element at index itemcode is
greater than item)
place the element at index itemcode to index itemcode+1
decrement itemcode
place  item at index itemcode+1
void insertionSort(int arr[ ]){ int
i,j,temp;
for(j=1; j < arr.length-1; j++){ temp =
arr[j];
i = j; // range 0 to j-1 is sorted while(i > 0
&& arr[i-1] >= temp){ arr[i] = arr[i-1];
i--;
}
arr[i] = temp;
} // end outer for loop } // end
insertion sort
```

4.3. Complexity analysis of bubble sort and insertion sort

While considering bubble sort, it may be seen that for an array of size n, in the worst case:

1st passage through the inner loop: n-1 comparisons and n-1 swaps.

(n-1)st passage through the inner loop: one comparison and one swap.

All together: c ((n-1) + (n-2) + ... + 1), where c is the time required to do one comparison, one swap, check the inner loop condition and increment j.

The constant time k may be spent declaring i, j, temp and initializing i. Outer loop is executed n- 1times, suppose the cost of checking the loop condition and

decrementing i is c_1.

c ((n-1) + (n-2) + ... + 1) + k + c1(n-1)+(n-1) + (n-2) + ... + 1 = n(n-1)/2

so the function equals c n*(n-1)/2 + k + c1(n-1) = 1/2c (n^2-n) + c(n-1) + k complexity $O(n^2)$.

While comparing with the insertion sort, in the worst case, n(n-1)/2 comparisons are made and shifts are done to the right. So the worst case complexity is $O(n^2)$. In case of best case the array is already sorted, no shifts are required.

4.4. Experimental analysis

4.4.1 Table : Selection Sort

Sl. No.	*Elements*	*Time(Sec.)*	*Time/n*	*Time/n*n*	*Time/n*logn*
01	16	0.00085938	0.00005371	0.00000336	0.00001937
02	32	0.00304688	0.00009521	0.00000298	0.00002747
03	64	0.01109375	0.00017334	0.00000271	0.00004168
04	128	0.03687500	0.00028809	0.00000225	0.00005937
05	256	0.14187500	0.00055420	0.00000216	0.00009994

4.4.2 Table : Insertion Sort

Sl. No.	*Elements*	*Time(Sec.)*	*Time/n*	*Time/n*n*	*Time/n*logn*
01	16	0.0005894	0.00003662	0.00000229	0.00001321
02	32	0.00210938	0.00006592	0.00000206	0.00001902
03	64	0.00781250	0.00012207	0.00000191	0.00004830
04	128	0.03000000	0.00023437	0.00000183	0.00008365
05	256	0.11875000	0.00046387	0.00000181	0.00014754

5. Performance

The bubble sort is not so popular for its poor performance on random data. It is justified because it is nearly three times as slow as insertion sort.

6. Discussion and future direction

Sometimes bubble sort is acceptable because it runs in O(n) time on sorted data and works well on

"nearly sorted" data. But the insertion sort is better than bubble sort. It is stable. Insertion sort is used to sort small arrays in the standard libraries.

7. Conclusion

In many cases the appropriate sort to use is provided by the standard libraries. But the general sorts do not work in all situations because sorting is an implementation of algorithmic techniques. It has been properly defined and verified about origins of bubble sort and compared with insertion sort. While analyzing the complexity in coding as well as run time it is seen that insertion sort is much better than bubble sort.

References

[1] W. Dobosiewicz, An effcient variant of bubble sort, Information Processing Letters, 11:1 (1980), 5-6.

[2] Knuth, D. The Art of Computer Programming: Sorting and Searching, 2 ed., vol. 3. Addison-Wesley, 1998.

[3] Sedgewick, R. Algorithms in *C++*, 3 ed. Addison- Wesley, 1998.

[4] Klerlein, J. B., and Fullbright, C. A transition from bubble sort to shell sort. In The Papers of the Nineteenth Technical Symposium on Computer Science Eduction (February 1988), ACM Press, pp. 183– 184. SIGCSE Bulletin V. 20 N. 1.

[5] Dale, N., and Lewis, J. Computer Science Illuminated. Jones and Bartlett, 2002.

[6] Cooper, D. Oh My! Modula-2! W.W. Norton, 1990.

[7] Sedgewick, R. *Algorithms in C++*, 3 ed. Addison-Wesley, 1998.

[8] A. V. Aho and J. D. Ullman. Foundations of Computer Science. W.H. Freeman, 1992.

[9] M. T. Goodrich and R. Tamassia. Algorithm Design: Foundations, Analysis, and Internet Examples. Wiley, 2002.

Authors' Profile

Er. Sambit Kumar Mishra has obtained B.E. and M.Tech. in Computer Science & Engineering from Amaravati University, Maharastra and Indian School of Mines, Dhanbad respectively. He is now serving in ABIT, Cuttack as Associate Professor in the department of Computer Science & Engineering. He has submitted his Ph.D thesis for evaluation. He has 13 number of publications in reputed International Journals. Recently he is also reviewer of peer reviewed international Journals, e.g . International Journal of Scientific and Engineering Research (IJSER) and International Journal of Information Technology and Computer Science(IJITCS), European Journal of Academic Essays.

Prof.(Dr.)Dulu Patnaik is presently working as Principal, Government College of Engineering, Bhawanipatna. He obtained his M.E. from N.I.T., Rourkela and Ph.D from Indian School of Mines, Dhanbad. He is also Editor and Chief Editor of many reputed International Journals.

Dr. Bibhuti Bhusan Mishra, Asst.Professor in the area of Systems & Information Technology is an MCA, MBA & LLB from Utkal University. He is an Ph.D in management from Sambalpur University. He has more than eleven years of teaching experience. He is currently associated with various research and development activities in the area of technology in Institute of Business & Computer Studies, Siksha “O” Anusandhan University, Bhubaneswar.