Journal of Computer Sciences and Applications, 2015, Vol. 3, No. 2, 40-45
Available online at http://pubs.sciepub.com/jcsa/3/2/4
© Science and Education Publishing
DOI:10.12691/jcsa-3-2-4

Genetic Algorithm Based on Sorting Techniques

Parul Aggrawal*, Faisal Naved, Mohd Haider

Department of Computer Science, Jamia Hamdard University, New Delhi-62, India
*Corresponding author: pagarwal@jamiahamdard.ac.in

Received March 18, 2015; Revised April 03, 2015; Accepted April 16, 2015

Abstract Genetic Algorithm, an Artificial Intelligence approach is based on the theory of natural selection and evolution. Traditional methods of sorting data are too slow in finding an efficient solution when the input data is too large. In contrast, Genetic Algorithm generates fittest solutions to a problem by exploiting new regions in the search space. This paper targets the three most commonly used Bubble, Selection and Insertion sorting techniques and executes memory on an input ranging from 1,000 to 10,000 where the input is entered in increasing, decreasing and random order. It mainly uses the Genetic Algorithm approach to optimize the effect of the three algorithms by generating an output which is consistent in terms of time variations which is not otherwise. This has been achieved by exploiting the property of Genetic Algorithm by choosing best parameter for population size, encoding, selection criteria, operator choice and optimized fitness function.

Keywords: *genetic algorithm, sorting, selection, crossover, mutation*

Cite This Article: Parul Aggrawal, Faisal Naved, and Mohd Haider, "Genetic Algorithm Based on Sorting Techniques." *Journal of Computer Sciences and Applications*, vol. 3, no. 2 (2015): 40-45. doi: 10.12691/jcsa-3-2-4.

1. Introduction

Section 1 of this paper comprises of introduction to the basic sorting and Genetic Algorithm. Section 2 comprises of related work in which average elapsed time is calculated at different inputs for Bubble, Insertion and Selection Sort. Section 3 contains the Experimental Results in which Inconsistency in Bubble Sort, Insertion sort and Selection sort is shown when inputs are entered in increasing order, decreasing order and randomly. Section 4 is the proposed work containing the Algorithm. Section 5 is the Conclusion and future scope. Section 6 are the references in support to the research paper

Sorting is considered as the most fundamental problem while studying Algorithms. The basic principle behind sorting a sequence of n numbers is that its elements are placed randomly and needs to be reordered in ascending order..Search preprocessing is the most important application of sorting. When the values are in sorted order, a better approach is to use binary search. O (n2) time proportion will be taken by an inefficient algorithm where n is equal to the number of elements in an array, whereas, an efficient algorithm takes O (n. lgn) time proportion to sort sequence of numbers where n is the size of the array.For small inputs, we cannot see a big difference, but if we have a larger input (say Population of a city) we can see that an enormous amount of usability played off between the efficient and inefficient algorithms:

Population of a city: =6 million people

$n=6 \text{ X } 10^6$ steps needed

$n^2=36 \text{ X } 10^{12}$

$=3600 \text{ X } 10^{10}$

$=3600$ seconds (Assuming that 10^{10} steps are executed in 1 sec)

$n.\lg n = 10^6 \text{ X } 20 \text{ X } 36 \Rightarrow 1$ sec

So, if we have to sort a population of 6 million people, only 1 sec is taken by efficient algorithm.

n	$n^2/4$	n. logn
10	25	33
100	2,500	664
1,000	25,000	9,954
10,000	250,000	132,877
100,000	2,500,000	1,660,960

O (n log n) time is needed to sort large data because, sorting by 0 (n^2) becomes impossible if we have a large data. There are lots of applications, besides simply looking for the maximum or minimum:-- For finding duplicates in a set: sort first, then duplicates will appear next to each other, and can be found by scanning through the sorted array. In finding similar values. In Histograms (counting frequencies): sort first, then do a single pass: repeated items will occur in bunches, and can be counted easily. Note that there are other ways to this. Intersection: given two arrays, what values do they have in common? Sort both of them, and march down both lists, advancing down the list with the smaller value each time. You will find the common values easily. Setting data up for later fast searching: if your data is sorted, you can use binary search to find values in O (log (n)) time.

Sorting is defined as follow:

Input: A sequence of n numbers $< A_1, A_2, A_3, \ldots\ldots\ldots\ldots A_n >$

Output: A permutation (reordering) $< A_{1'}$, A2, ', A3, '$\ldots\ldots\ldots\ldots\ldots A_{n'} >$ of input sequence such that $A_{1'} <= A_{2'} <= \ldots\ldots\ldots <= A_{n'}$. [4]

There are several sorting algorithms. Some of them are Bubble sort, Selection Sort, Insertion Sort, Quick sort,

Merge sort, Heap sort, Counting Sort, Radix sort, Bucket sort. The Time and Space Complexities of these sorting techniques are:

SORTING	TIME COMPLEXITY		SPACE COMPLEXITY	
	BEST	AVG.	WORST	
BUBBLE SORT	O (n^2)	O (n^2)	O (n^2)	O (1)
SELECTION SORT	O (n^2)	O (n^2)	O (n^2)	O (1)
INSERTION SORT	O (n)	O (n^2)	O (n^2)	O (1)
QUICK SORT	O (n.lg (n))	O (n^2)	O (n.lg (n))	O (1)
MERGE SORT	O (n.lg (n))	O (n.lg (n))	O (n.lg (n))	O (1)
HEAP SORT	O (n.lg (n))	O (n.lg (n))	O (n.lg (n))	O (1)
COUNTING SORT	O (n+k)	O (n+k)	O (n+k)	O (n+2^k)
RADIX SORT	O (n.k/s)	O (2^s.n.k/s)	O (n.k/s)	O (n)
BUCKET SORT	O (n.k)	O (n^2. k)	O (n.k)	O (n.k)

1.1. Genetic Algorithm

Genetic Algorithms [1] are the adaptive heuristic search algorithms which are based on the process of growth and development [2]. The basic concept of genetic algorithm follows the principle of survival of the fittest, which was given by Charles Darwin. In 1960's [2], John Holland discovered the genetic algorithm. This is the process of moving the chromosome of one population to a new population with the help of some operators such as selection, crossover and mutation.

A population is comprised of a finite value of chromosomes, each chromosome contains a number of genes [3].

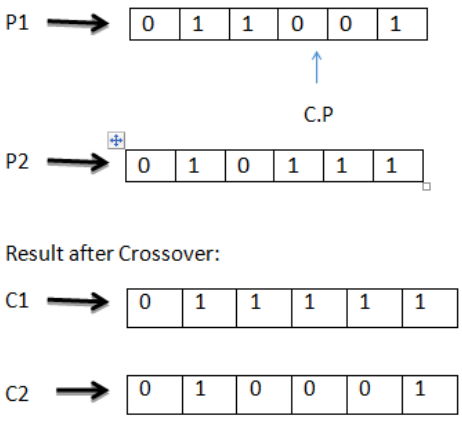
A gene is represented by a binary number or decimal number according to the problem or just the sake of simplicity, this gene value is sometimes called as an allele.

1.2. Some Genetic Algorithm Operators

SELECTION: - This operator is used to select chromosomes for reproduction. Fitter chromosome has a higher probability of selection.

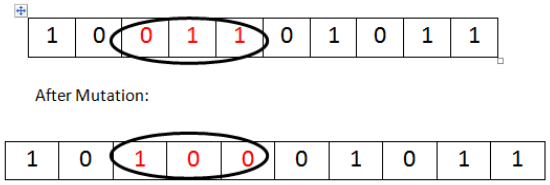
CROSSOVER: - This operator moves the genes of two parent chromosomes to create two new child's chromosomes [6].

No. of crossover = (Crossover rate*number of a chromosome * number of genes in a chromosome) /100



MUTATION: - Genetic diversity from one population to another is performed by mutation operator. This alters the genes or chromosome of the population.

Steps of Genetic Algorithm: Generate an initial population randomly as you want either in binary or in decimal format. Then the fitness of each chromosome is calculated. Selection operator is applied on the population. Crossover operator is applied on the population. Mutation operator is applied on whole population. These processes are repeated until we get the fittest chromosome or until we get the output.



2. Related Work

In our approach, we have taken the three sorting algorithms namely bubble sort, insertion sort and selection sort in increasing order, decreasing order and randomly in order to check the elapsed time taken by each sorting technique. In this section we have shown the elapsed time taken by each sorting technique with the help of tables and charts. Table 1 and the corresponding chart (Figure 1) shows the average elapsed time taken by insertion sort when inputs are entered ranging from 1,000 to 10,000 first in increasing order then in decreasing order and then the inputs are taken randomly.

Table 1. Average Elapsed Time of Insertion sort at different inputs

Input	INCREASING	DECREASING	RANDOM
1000	0.238	0.340	0.252
2000	0.598	0.736	0.472
3000	1.263	1.296	0.747
4000	2.263	2.087	1.186
5000	3.166	3.164	1.681
6000	4.656	4.406	2.318
7000	6.065	6.032	3.164
8000	7.683	7.659	4.01
9000	8.331	9.686	5.073
10000	13.084	11.714	6.076

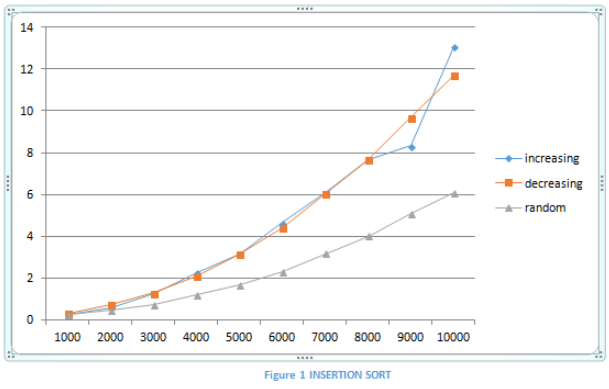


Figure 1. Insertion sort at different inputs

Table 2 and the corresponding chart (Figure 2) shows the average elapsed time taken by Bubble sort when inputs are entered ranging from 1,000 to 10,000 first in increasing order then in decreasing order and then the inputs are taken randomly.

Table 2. Average Elapsed Time of Bubble sort at different inputs

	INCREASING	DECREASING	RANDOM
1000	0.219	0.263	0.296
2000	0.373	0.659	0.571
3000	0.626	1.142	0.912
4000	0.89	1.846	1.439
5000	1.252	2.274	2.076
6000	1.681	3.824	2.933
7000	2.241	S.214	3.960
8000	2.813	6.604	5.054
9000	3.51	8.406	6.359
10000	4.230	10.208	7.659

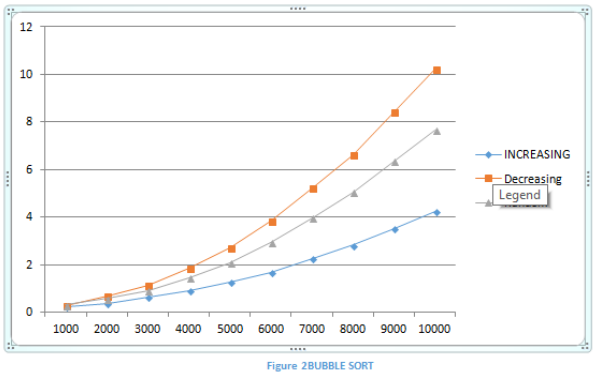


Figure 2. Bubble sort at different inputs

Similarly, Table 3 and the corresponding chart (Figure 3) shows the average elapsed time taken by selection sort when inputs are entered ranging from 1,000 to 10,000 first in increasing order then in decreasing order and then the inputs are taken randomly.

Table 3. Average Elapsed Time of Selection sort at different inputs

SELSORT	INCREASING	DECREASING	RANDOM
1000	0.03	0.197	0.131
2000	0.351	0.395	0.362
3000	0.532	0.67	0.56
4000	0.363	1.109	0.368
5000	1.313	1.604	1.307
6000	1.335	2.23	1.39
7000	2.461	3.054	2.351
8000	3.131	3.9008	3.15
9000	3.901	4.857	3.911
10000	4.379	5.834	4.736

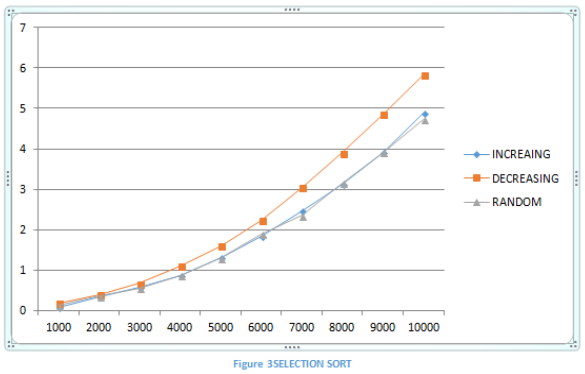


Figure 3. Selection sort at different inputs

3. Experimental Results

Each input is entered (ranging from 1,000 to 10,000) five times in order to check the consistency of the following sorting techniques. Table 4, Table 5, Table 6, Table 7, Table 8, Table 9, Table 10, Table 11, Table 12, Table 13 and the corresponding charts (Figure 4, Figure 5, Figure 6, Figure 7, Figure 8, Figure 9, Figure 10, Figure 11, Figure 12, Figure 13) shows the inconsistency in elapsed time for different sorting techniques and each input entered five times consecutively on Windows 7 operating system, 2GB RAM, Pentium® Dual-Core @ 2.2 GHz CPU.

3.1. Bubble Sort Inc

Table 4. Inconsistency in Bubble Sort when inputs are entered in increasing order

no. of input	1	2	3	4	5	average
1000	0.219	0.219	0.219	0.274	0.164	0.219
2000	0.329	0.439	0.384	0.329	0.384	0.373
3000	0.604	0.659	0.659	0.604	0.604	0.626
4000	0.934	0.879	0.934	0.879	0.824	0.89
5000	1.263	1.318	1.208	1.208	1.263	1.252
6000	1.703	1.648	1.648	1.703	1.703	1.681
7000	2.208	2.176	2.354	2.208	2.354	2.26
8000	2.857	2.747	2.802	2.802	2.857	2.813
9000	3.545	3.545	3.478	3.545	3.545	3.518
10000	4.285	4.23	4.175	4.175	4.285	4.23

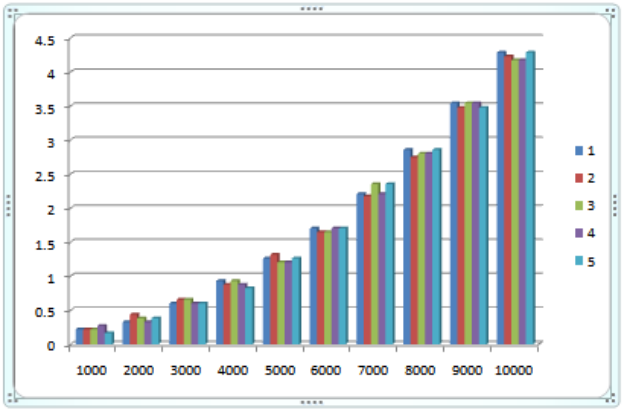


Figure 4. Inconsistency in Bubble Sort when inputs are entered in increasing order

Table 5. Inconsistency in Bubble Sort when inputs are entered in decreasing order

no. of input	1	2	3	4	5	average
1000	0.164	0.274	0.219	0.384	0.274	0.263
2000	0.659	0.659	0.714	0.604	0.659	0.659
3000	1.153	1.153	1.208	1.098	1.098	1.142
4000	1.868	1.813	1.868	1.813	1.868	1.846
5000	2.637	2.692	2.747	2.692	2.802	2.714
6000	3.846	3.846	3.791	3.791	3.846	3.824
7000	5.103	5.103	5.208	5.164	5.208	5.157
8000	6.538	6.648	6.593	6.593	6.648	6.604
9000	8.383	8.329	8.329	3.306	8.384	8.346
10000	10.164	10.219	10.274	10.164	10.219	10.208

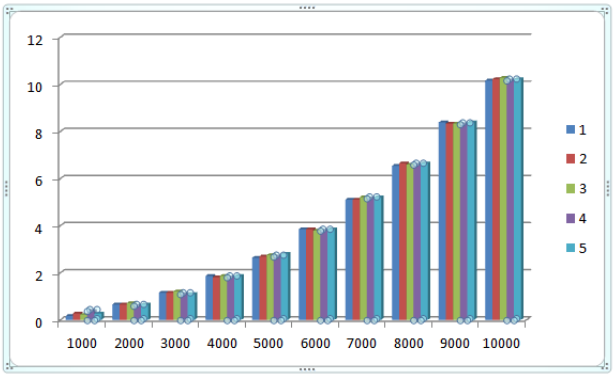


Figure 5. Inconsistency in Bubble Sort when inputs are entered in decreasing order

Table 6. Inconsistency in Bubble Sort when inputs are entered randomly

no. of input	1	2	3	4	5	average
1000	0.329	0.274	0.274	0.329	0.274	0.296
2000	0.549	0.549	0.604	0.604	0.549	0.571
3000	0.934	0.879	0.879	0.934	0.934	0.912
4000	1.483	1.373	1.373	1.483	1.483	1.439
5000	2.087	2.032	2.087	2.142	2.032	2.076
6000	2.912	2.857	3.021	2.912	2.967	2.9338
7000	4.064	4.008	4.946	4.946	4.064	4.005
8000	5.109	5.054	5.054	4.945	5.109	5.0542
9000	6.364	6.364	6.478	6.346	6.215	6.357
10000	7.692	7.747	7.692	7.637	7.527	7.659

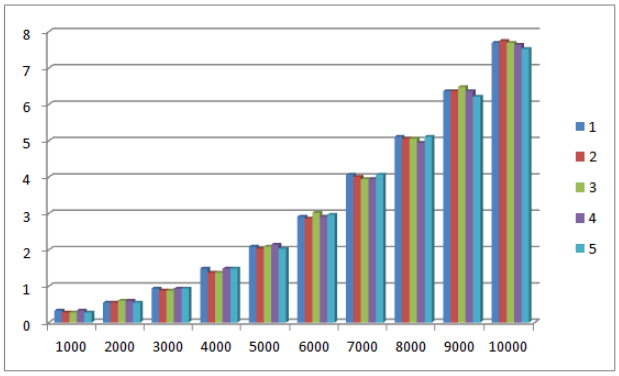


Figure 6. Inconsistency in Bubble Sort when inputs are entered randomly

Table 7. Inconsistency in Insertion Sort when inputs are entered in increasing order

no. of inputs	1	2	3	4	5	Average
1000	0.274	0.219	0.274	0.164	0.274	0.241
2000	0.1	0.659	0.659	0.549	0.659	0.5252
3000	1.208	1.318	1.153	1.318	1.318	1.263
4000	2.252	2.252	2.307	2.197	2.307	2.263
5000	3.186	3.131	3.131	3.241	3.131	3.164
6000	4.835	4.835	4.89	4.186	4.835	4.7162
7000	6.098	6.098	5.989	6.043	6.153	6.0762
8000	7.637	7.582	7.747	7.692	7.637	7.659
9000	8.241	8.357	8.461	8.186	8.357	8.3204
10000	13.131	12.857	13.296	13.076	12.912	13.0544

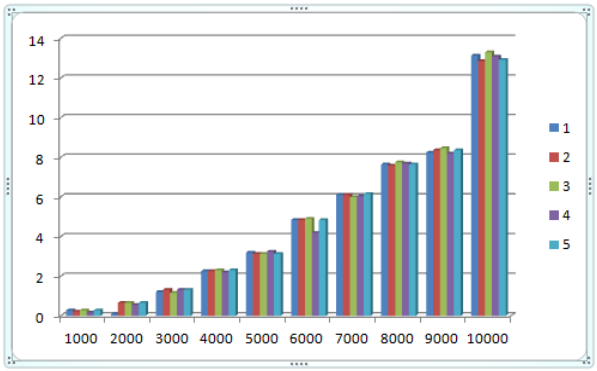


Figure 7. Inconsistency in Insertion Sort when inputs are entered in increasing order

Table 8. Inconsistency in Insertion Sort when inputs are entered in decreasing order

no. of inputs	1	2	3	4	5	Average
1000	0.274	0.384	0.384	0.384	0.274	0.34
2000	0.769	0.769	0.714	0.769	0.659	0.736
3000	1.318	1.318	1.263	1.263	1.318	1.296
4000	2.032	2.142	2.032	2.142	2.087	2.087
5000	3.186	3.131	3.131	3.241	3.131	3.164
6000	4.395	4.45	4.45	4.395	4.34	4.406
7000	5.806	6.086	5.945	6.086	5.945	5.936
8000	7.637	7.582	7.747	7.692	7.637	7.659
9000	9.506	9.386	9.453	9.506	9.506	9.471
10000	11.813	11.758	11.813	11.373	11.813	11.714

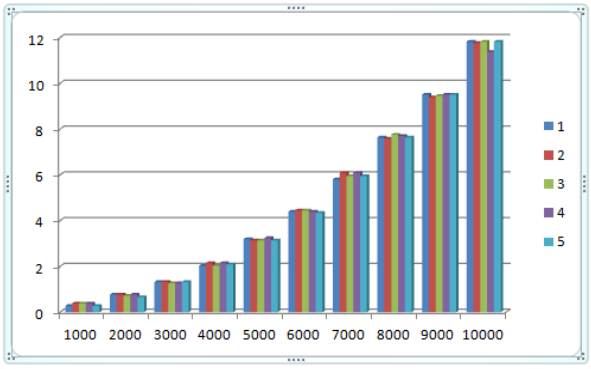


Figure 8. Inconsistency in Insertion Sort when inputs are entered in decreasing order

Table 9. Inconsistency in Insertion Sort when inputs are entered randomly

no. of inputs	1	2	3	4	5	Average
1000	0.274	0.274	0.274	0.219	0.219	0.252
2000	0.494	0.494	0.439	0.494	0.439	0.472
3000	0.769	0.769	0.769	0.714	0.714	0.747
4000	1.153	1.153	1.208	1.208	1.208	1.186
5000	1.648	1.703	1.703	1.593	1.758	1.681
6000	2.307	2.307	2.307	2.362	2.307	2.318
7000	3.106	2.806	3.045	3.045	3.106	3.021
8000	3.956	4.01	4.065	3.956	4.065	4.0104
9000	5.093	5.25	5.157	5.25	5.093	5.168
10000	6.098	6.098	5.989	6.043	6.153	6.0762

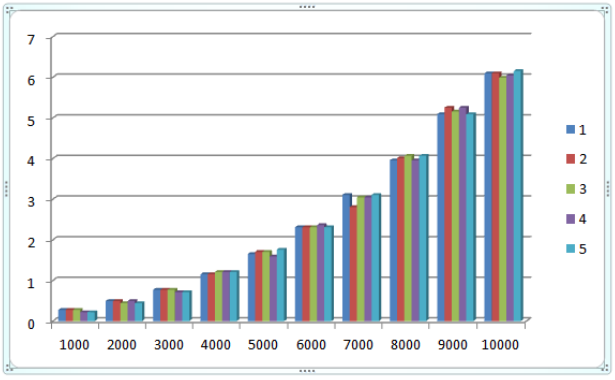


Figure 9. Inconsistency in Insertion Sort when inputs are entered randomly

Table 10. Inconsistency in Selection Sort when inputs are entered in increasing order

no. of inputs	1	2	3	4	5	average
1000	0.1	0.1	0.1	0.1	0	0.08
2000	0.329	0.329	0.329	0.384	0.384	0.351
3000	0.605	0.659	0.494	0.604	0.549	0.5822
4000	0.879	0.769	0.879	0.934	0.879	0.8638
5000	1.373	1.318	1.208	1.318	1.373	1.318
6000	1.758	1.868	1.813	1.923	1.813	1.835
7000	2.472	2.472	2.417	2.527	2.417	2.461
8000	3.076	3.131	3.241	3.076	3.131	3.131
9000	3.846	3.846	3.956	3.956	3.901	3.901
10000	4.89	4.835	4.945	4.835	4.89	4.879

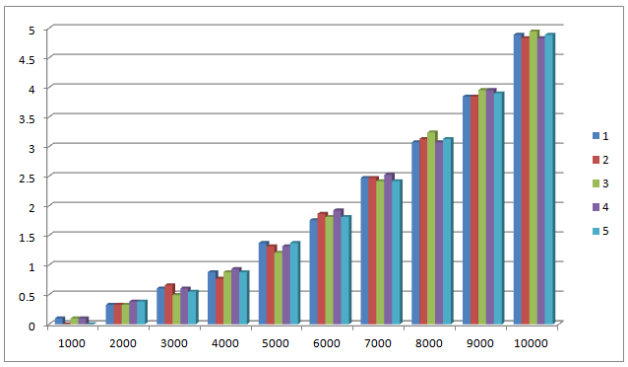


Figure 10. Inconsistency in Selection Sort when inputs are entered in increasing order

Table 11. Inconsistency in Selection Sort when inputs are entered in decreasing order

no. of inputs	1	2	3	4	5	average
1000	0.219	0.219	0.164	0.219	0.164	0.197
2000	0.439	0.329	0.384	0.384	0.439	0.395
3000	0.659	0.714	0.607	0.714	0.659	0.6706
4000	1.098	1.098	1.153	1.098	1.098	1.109
5000	1.538	1.648	1.538	1.648	1.648	1.604
6000	2.307	2.307	2.197	2.032	2.307	2.23
7000	3.076	3.021	2.967	3.131	3.076	3.0542
8000	4.01	3.901	3.846	3.846	3.901	3.9008
9000	4.945	4.835	4.835	4.78	4.89	4.857
10000	6.098	6.043	5.934	5.604	5.494	5.8346

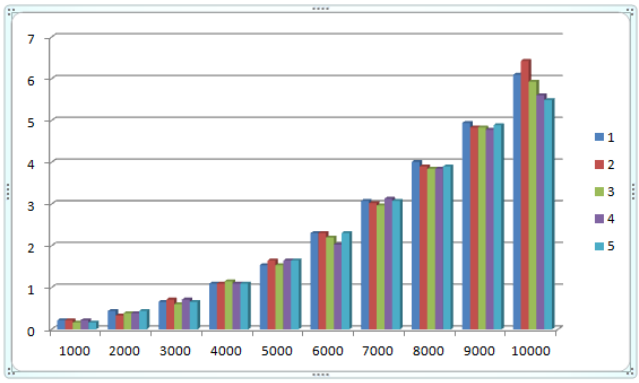


Figure 11. Inconsistency in Selection Sort when inputs are entered in decreasing order

Table 12. Inconsistency in Selection Sort when inputs are entered randomly

no. of inputs	1	2	3	4	5
1000	0.164	0.109	0.164	0.109	0.109
2000	0.329	0.384	0.384	0.329	0.384
3000	0.549	0.549	0.604	0.494	0.604
4000	0.824	0.879	0.934	0.879	0.824
5000	1.263	1.373	1.373	1.263	1.263
6000	1.868	1.923	1.868	1.868	1.923
7000	2.472	2.417	1.978	2.527	2.362
8000	3.168	3.021	3.241	3.186	3.137
9000	3.846	4.01	3.956	3.846	3.901
10000	4.89	4.945	4.285	4.835	4.725

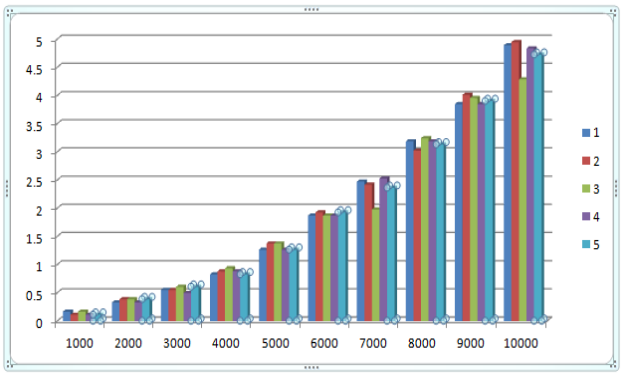


Figure 12. Inconsistency in Selection Sort when inputs are entered randomly

4. Proposed Work

The above sections shown the inconsistency of the sorting techniques for large amount of data but, in order to get consistent results we have to follow the GA approach. How we mapped the data into GA approach? Firstly, we have made a population containing 'm' number of chromosomes and 'n' number of cells (where m and n are integers). And we have taken number of cells equal to the number of elements which we want to sort. How we get the sorted array? We will be searching for the fittest chromosome using interpretation. Interpretation is done by reading those elements of the array which are at the position containing 1 in the chromosome and then the elements at position containing 0 in the chromosome. And then we check whether the array is sorted or not. Secondly, If we do not get the sorted array then crossover is done on the population as discussed in section-1. Then again, we do the interpretation and if the array is still not sorted then

we perform the mutation on the whole population as discussed in section-1. The process of interpretation is repeated to check for the sorted array.

5. Algorithm

1. gensort(A,n)
2. generate population(pop,m,n)
3. interpretation(pop,A,m,n)
4. noc=(crossover rate*m*n)/100
5. for each i € noc
6. n1=rand()%m
7. n2=rand()%m
8. cp=rand()%n
9. for each j € n
10. c1=merge the data before cp of n1 with data after cp of n2
11. c2=merge the data after cp of n1 with data before cp of n2
12. add c1 & c2 to the population
13. interpretation(pop,A,m,n)
14. for each i € m
15. mp=rand()%m
16. for each j € n
17. if j=mp
18. pop(i,j)=!pop(i,j)
19. interpretation(pop,A,m,n)

where,
noc=number of crossovers
cp= crossover point
mp=mutation point
n1, n2=randomly selected chromosome
c1, c2=chromosomes after performing crossover

1. interpretation(pop,A,m,n)
2. for each i
3. for each j € n
4. if pop(i,j)=1
5. na[k++]=A[j]
6. for each j € n
7. if pop(i,j)=0
8. na[k++]=A[j]
9. check the na[] array is sorted

where,
na=new array

6. Conclusion

This paper proposed that, for large amount of data the basic sorting techniques become inconsistent and to overcome this problem we have used the GA technique we want to sort. And GA approach have the time complexity of O(mn) where 'm' is the number of Chromosomes in a population and 'n' is the number of cells in a chromosome which is equal to the number of elements in the array. 'm' is very less as compared to 'n', if you have a large amount of data. Hence, GA approach is more efficient than the basic sorting techniques. to sort a large amount of data. Theoretically, these basic sorting techniques (i.e. Insertion sort, Bubble sort and selection sort) have the time complexity of $O(n^2)$ where 'n' is equal to the number of elements of the array. The GA approach, due to its underlying property of selecting the best parameter of chromosomes, population, encoding etc is bound to produce better results. Theoretically, this has been analysed and presented in this paper. This could not be supported currently due to hardware constraints.

In future, we shall explore and support it with experimental results on data which could not only be numeric but also text, audio, video, etc.

References

[1] D. Goldberg. Genetic Algorithms in Search, Optimization, and Machine

[2] Learning. Addison-Wesley, Reading, MA, 1989. [2] David E.Goldberg," Genetic Algorithms in search, optimization and machine learning 1st, Addison-Wesley Longman Publishing Co., Inc. Boston ©1989

[3] H. Bhasin and Neha Singla, "Cellular Genetic Test Data Generation", ACM SIGSOFT Software Enginnering Notes, Vol. 38 (5), September 2013, Pages 1-9.

[4] Introduction to Algorithm, Second Edition Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, Clifford Stein, Mc-Graw Hill Publications

[5] H. Bhasin, "Cost Priority Cognizant Regression Testing", ACM SIGSOFT Software Enginnering Notes, Vol. 39 (3), May 2014, Pages 1-7.

[6] T. Back, D. B. Fogel, and Z. Michalewicz. Evolutionary Computation Vol. I & II. Institute of Physics Publishing, 2000.