# Algorithm Analysis Tool based on Execution Time-Input Instance-based Runtime Performance Benchmarking

Piyush Mishra
University of Petroleum and Energy Studies, Dehradun, India

Vivek Patel
University of Petroleum and Energy Studies, Dehradun, India

Parul Mittal
University of Petroleum and Energy Studies, Dehradun, India

J. C. Patni
University of Petroleum and Energy Studies, Dehradun, India

## ABSTRACT
Algorithms are a fundamental component of Computer Science, with every development in this field based on or around them. Each algorithm is evaluated for its performance using some technique, with asymptotic analysis being a frequently used one. Algorithms that have best time complexity theoretically (be it Oh, Theta or Omega Notation), may not have the best execution time in practice which depends on implementation efficacy, input dataset, constants and factors that are overlooked in asymptotic analysis. The lack of software which allows a user to compare various algorithms available for an operation for a given input dataset, supplemented with its graphical analysis encourages for the creation of the same. In this paper, we present a software tool which provides a range of algorithms for a given operation and measures the execution time for each of them. It then provides a graphical analysis of the algorithms executed, showing the performance of the algorithms belonging to a particular operation when run against a custom, input data set.

## Keywords
Algorithms, Benchmark, Runtime-performance, Graphical-analysis

## 1. INTRODUCTION
The analysis of algorithms in terms of time complexity and space complexity is important in order to determine its usability and efficiency while performing an operation. For example, while sorting an input data set of say 50 random elements, the option of using merge sort/quick sort/heap sort is available, with all performing in O(*nlogn*) time depending on whether it is average case or worst case [1]. While a person familiar with the subject of data structures and algorithms may be able to decide which sorting algorithm is suitable in the given situation on the basis of asymptotic analysis, the same can't be said for a layman. He or she won't be able to determine the suitable algorithm for the given input data set solely on the basis of asymptotic complexity. For such a user, the execution time of an algorithm is the sole factor to evaluate its efficiency.

Moreover, the execution time takes into consideration the various constants that get hidden while approximating the time complexities [1]. Also, actual performance depends highly on the input datasets [2, 3]. Hence, if the sorting algorithms are compared on the basis of their execution time while operating on the above-mentioned input data set of 50 random elements, the result will be evident without any ambiguity.

## 2. PREVIOUS WORK
There are few libraries available which provide similar functionality of runtime performance monitoring of algorithms and methods. These are –

- JETM (Java Execution-Time Management Library)
- Google Benchmark
- Google Caliper
- Apache JMeter

Another popular library, Speedy McBenchmark exists for use in java. It provided trend analysis by performing operations on similar modules. Its main focus is to check if an algorithm scales in terms of O(n), O(*nlogn*), etc. We declare a constraint which shows if a particular implementation is faster than a group of some other implementations. However, it scales all the algorithms relative to the Oh notation. The drawback of this library is that it is complex to use and optimization is complicated. In general, none of these tools provide a graphical analysis and statistics for algorithms relevant to an operation based on input dataset instances [4]. Since the developed software is for educational purposes, ease of use and installation has also been ensured. Ease of modification is also to be considered if further operations are to be added. The addition of new operations in the developed application is relatively simple as compared to the above-mentioned tools.

## 3. PROPOSED ALGORITHMIC TOOL
The tool (application) has been defined for cross platforms, thus enabling it to be run on windows operating system as well as on various flavors of Linux. A graphical user interface (GUI) which has been created entirely using GIMP Toolkit(GTK prompts the user to select the desired operation. It is used along with CAIRO library. GTK stands for gimp toolkit. GTK+ is the latest development of GTK. Many desktop environments like GNOME, Unity and Sugar utilize GTK.

Each operation is computed using predefined algorithms which run in parallel. The sorting operation is performed using the following six algorithms: insertion sort, selection sort, merge sort, quick sort, randomized quick sort and bubble sort. The searching operation supports linear search and binary search. The following algorithms are executed under disk scheduling: FCFS, SJF, Round robin and.

The graph analysis displayed as the final result is in the form of pie chart. The implementation of the pie chart is done using the Cairo library support for GTK. The input for any of the operation can be provided in two ways: the

user loads his custom text file which contains the input data set. A button to load the file has been created in the main application window. It allows the user to navigate to the directory location containing his/her file. Apart from this if the user does not ant to provide an input data set, the application is preloaded with certain text files containing default input data sets. The default text file for say, the sorting operation, contains variations of inputs that are possible for a sorting program *i.e*. numbers arranged in ascending order, numbers arranged in descending orders and numbers arranged in random manner. This allows testing the efficiency of each sorting algorithm in various situations. The text files for the other two operations have also been created keeping in mind the various input scenarios possible for the respective operations. To maintain a log of the operation performed, a text file is generated at after the graphical analysis is displayed [5]. It contains statistics of the algorithms related to the operations. The numbers include execution times, and the number of test cases in which an algorithm was superior than the others.

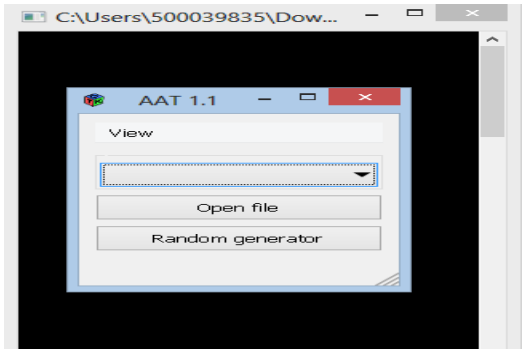


**Fig1: Application front end on windows platform**

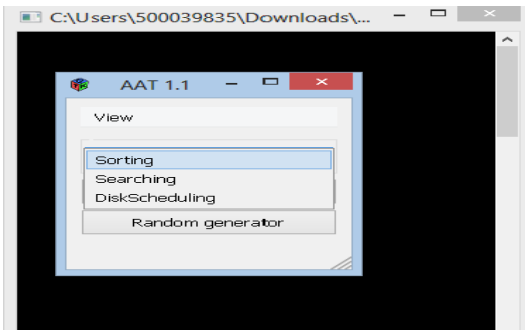


**Fig 2: Dropdown box displaying list of available operations**

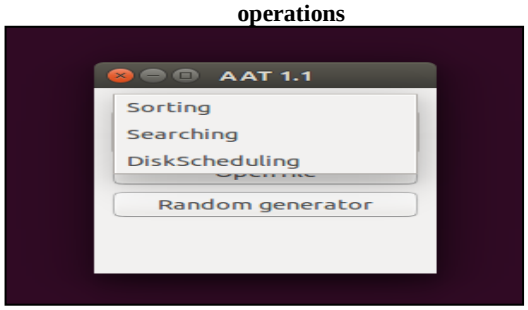


**Fig 3: Application front end in Ubuntu**

## 4. SOFTWARE REQUIREMENTS

The following requirements ensure the smooth running of the application:

a) Operating system: Linux (any flavor) or windows
b) Code Blocks IDE (software tested on Code Blocks 16)
c) Gimp Toolkit(GTK), preferably GTK+2

## 5. RESULTS

Figure 4 displays the graphical output of the searching performed and figure 5 shows the output text file generated. Figure 6 displays the graphical output of the sorting performed and figure 7 shows the output text file generated for the sorting operation. Figure 8 shows the input text file for the disk scheduling. Figure 9 displays the graphical output of the algorithms executed on the inputs.

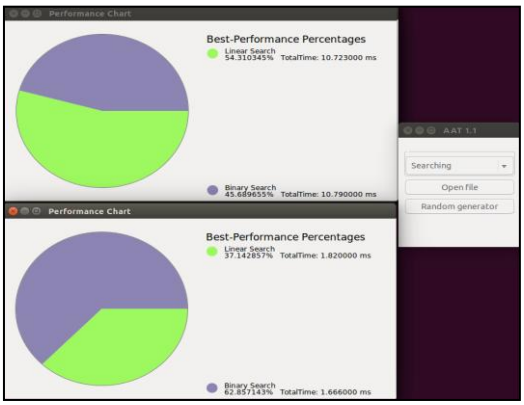


**Fig 4: Searching performed.**

The pie chart is divided into 2 parts denoted by green and blue respectively. The percentage indicates the number of test cases (in percentage) in which the algorithm was chosen for computation over other algorithms in the same category. In the above input, linear search was successful in 37.142% of the total test cases and the total time *i.e*., 1.82 milliseconds denotes the total time that linear search ran for the 37 % of the test cases where it was superior over binary search.

searching_output_#.txt

```
**********************************************************************
        OVERALL EXECUTION TIME
**********************************************************************

Linear Search:  72.455000ms
Binary Search:  69.764000ms

**********************************************************************
        ALGORITHM-WISE BEST-PERFORMANCE TEST CASES
**********************************************************************

Total Test-cases: 757

Linear Search: Performs best in following 363 test-cases-
2 3 4 6 7 9 12 13 14 15 18 19 20 22 27 28 29 30 35 37 38 40 41 42 43
95 98 99 100 107 112 113 114 115 119 121 122 126 127 129 130 131 133
171 174 177 179 182 184 186 188 191 194 195 196 199 200 201 202 203
234 235 236 239 241 242 243 244 246 247 248 249 252 255 256 260 261
295 296 300 301 304 306 308 321 322 325 329 334 339 341 342 344 346
377 378 379 380 381 383 384 390 391 392 393 399 400 401 402 404 410
455 457 460 461 463 468 469 470 474 475 477 478 479 483 484 487 489
545 546 547 552 553 556 558 561 562 564 567 568 578 579 581 583 584
614 616 617 620 622 624 627 629 631 633 634 637 638 644 646 649 650
692 694 698 699 700 701 702 705 706 709 710 712 713 714 716 718 721

Binary Search: Performs best in following 394 test-cases-
1 5 8 10 11 16 17 21 23 24 25 26 31 32 33 34 36 39 45 46 48 50 52 53
102 103 104 105 106 108 109 110 111 116 117 118 120 123 124 125 128
165 169 172 173 175 176 178 180 181 183 185 187 189 190 192 193 197
251 253 254 257 258 259 262 263 267 273 275 277 279 282 283 285 287
316 317 318 319 320 323 324 326 327 328 330 331 332 333 335 336 337
386 387 388 389 394 395 396 397 398 403 405 406 407 408 409 412 413
443 447 448 449 450 451 456 458 459 462 464 465 466 467 471 472 473
506 507 510 511 512 514 516 517 518 519 520 522 524 526 528 530 531
563 565 566 569 570 571 572 573 574 575 576 577 580 582 585 586 594
635 636 639 640 641 642 643 645 647 648 652 653 654 655 656 657 658
```

**Fig 5: Output file for searching.**

Each number in the matrix for each of the search algorithm signifies the test case number that was executed using the corresponding algorithm.

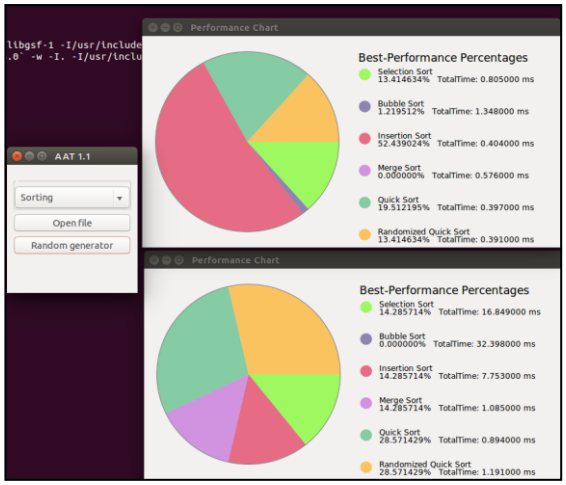


**Fig 6: Sorting performed.**

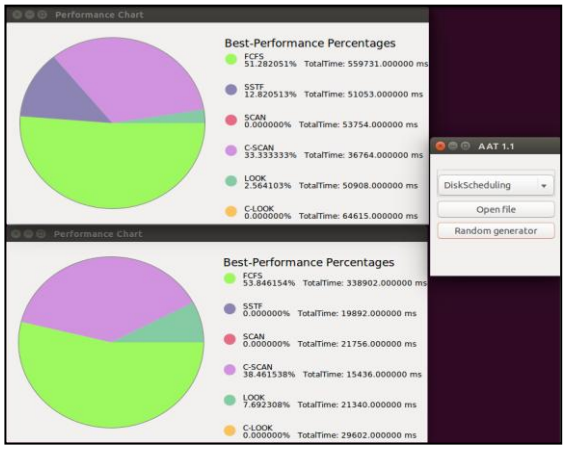


**Fig 7: Output file of sorting.**

If we consider the statistics of insertion sort, it performed best in 42 out of the 69 test cases provided in the input text file.

A total of 6 colors are used to denote the 6 disk scheduling algorithms. Consider C-SCAN, which is shown in purple. It ran for a total time of 36.764 ms for the 33.33 % of test cases where it was chosen as the best algorithm to be executed.

diskscheduling_output_#.txt

```
************************************************************
        OVERALL EXECUTION TIME
************************************************************

FCFS:   742548.000000ms
SSTF:   58354.000000ms
SCAN:   64857.000000ms
C-SCAN: 43405.000000ms
LOOK:   61132.000000ms
C-LOOK: 79353.000000ms

************************************************************
        ALGORITHM-WISE BEST-PERFORMANCE TEST CASES
************************************************************

Total Test-cases: 45

FCFS: Performs best in following 22 test-cases-
1 4 5 6 8 11 12 15 20 23 24 27 28 30 31 32 33 36 37 38 41 43

SSTF: Performs best in following 3 test-cases-
16 34 42

SCAN: Performs best in following 0 test-cases-


C-SCAN: Performs best in following 19 test-cases-
2 3 7 9 10 13 14 17 18 19 21 22 25 26 29 39 40 44 45

LOOK: Performs best in following 1 test-cases-
35

C-LOOK: Performs best in following 0 test-cases-
```

**Fig 8: Disk scheduling simulation performed.**

The two pie charts are indicative of two separate text files that were used to provide inputs for the sorting operation. A total of six colors are shown in the graph, each representing one algorithm. Consider the first pie chart. If we consider Insertion sort (shown in pink), it was used as the best strategy in 52.43 % of the total test cases. And it ran for a total of 0.404 ms.

diskscheduling_output_#.txt

```
************************************************************
        OVERALL EXECUTION TIME
************************************************************

FCFS:   742548.000000ms
SSTF:   58354.000000ms
SCAN:   64857.000000ms
C-SCAN: 43405.000000ms
LOOK:   61132.000000ms
C-LOOK: 79353.000000ms

************************************************************
        ALGORITHM-WISE BEST-PERFORMANCE TEST CASES
************************************************************

Total Test-cases: 45

FCFS: Performs best in following 22 test-cases-
1 4 5 6 8 11 12 15 20 23 24 27 28 30 31 32 33 36 37 38 41 43

SSTF: Performs best in following 3 test-cases-
16 34 42

SCAN: Performs best in following 0 test-cases-


C-SCAN: Performs best in following 19 test-cases-
2 3 7 9 10 13 14 17 18 19 21 22 25 26 29 39 40 44 45

LOOK: Performs best in following 1 test-cases-
35

C-LOOK: Performs best in following 0 test-cases-
```

**Fig 9: Output file for disk scheduling.**

Using the output file, we can determine in which test cases a specific disk scheduling algorithm performed the best. This can be done by examining the array generated under the scheduling algorithm to be analyzed. These figures confirm the working of the application. Similar results are obtained for windows based machine.

# 6. LIMITATIONS AND FUTURE WORK

## 6.1 Limitations

*6.1.1 GUI needs more robustness*

The GUI performs only searching and sorting operations. Additional functionality should be added to the interface so that it can be over for multiple purposes. The GUI should have more features which will make it robust. The GUI can have other functionality such as file handling, more operations, etc.

*6.1.2 Chronometer module for Windows OS has lesser precision in measuring execution time*

The chronometer module on windows has lesser precision as compared to Linux. It can create problem in analysis and graph precision. Analysis on value having minor difference will be tough on windows. The graph may not be built properly if the

*6.1.3 Statistical analysis being shown graphically may not be intuitive to some users*

Some users might not be interested in the graphical analysis. They might just be concerned about the execution time. Many technical analysts just need the statistical values. They are not concerned of the graphical analysis.

## 6.2 Future Enhancements

*6.2.1 Improving GUI robustness*

This would involve adding more functionality to the GUI and adding more ways for analysis. GUI can be made more robust by adding more features to the GUI. GUI should be made more users friendly. The GUI should also include more features other than searching and sorting. It should be made more and more attractive so that the users use it for educational purposes. The GUI should also add functionality of storing files and other file handling operations. This would help in more robustness of the GUI.

*6.2.2 Improving chronometer module for Windows OS*

The chronometer for windows does not show precise values in the analysis. It should be improved so that proper graph based on analysis can be made. As most of the people use windows operating system, the chronometer should be made more precise so that people use the analysis tool for educational purposes.

*6.2.3 Making the graphical analysis more intuitive*

The graph should be made more intuitive to the users who might not be interested in graphical analysis. Some technicians may just want comparison of values. So, the analysis should be made more intuitive in such a way so that everyone uses it.

*6.2.4 Extending support for more operations*

The current GUI deals with searching and sorting operations. The functionality for adding more operations such as string matching and inbuilt functions should be added. This would make the GUI robust also. The tool can be used for educational purposes.

*6.2.5 Adding various other graphs for improved Understanding and insight*

Adding more types of graph such as line graphs, etc. should be added so that proper analysis can be done. Adding other graphs will also help in making GUI more attractive.

# 7. CONCLUSION

Any kind of problem requires an algorithm in order to be solved. Every algorithm is characterized by its asymptotic behavior. The asymptotic behavior is decided how long the program runs and how much space it occupies in the memory. The steps of the algorithm decide the "time" and "space" complexity. This includes the iterative functions, the function calls, etc. Now, there are multiple ways of solving a problem and this in turn gives rise to many algorithms for the same problem. Each has its own advantages and disadvantages which are deciphered from its asymptotic analysis. Moreover, the algorithms function differently on the basis of the type of input given. Determining the type of algorithm to be applied for a specified operation solely on the basis of its asymptotic behavior is not sufficient. This is because the "type of input" is also a factor [2]. The concept of the "best algorithm" *i.e.*, the one which is efficient for any kind of input data is nonexistent [6]. The above holds true for in-built function/method provided by the compiler/interpreter/library.

Keeping this in mind, the project serves as an analytical tool wherein the user selects an operation and provides the input set accordingly. The program than runs various algorithms pertaining to the operation for the given input and decides the optimal one in terms of execution time for each dataset [7]. The result is based on a graphical analysis and for ease, a GUI is provided. In future, the GUI can be enhanced to make it more attractive for the user and also other algorithms analysis can be added to the GUI.

# 8. REFERENCES

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, 2001. Introduction to Algorithms, MIT Press.

[2] Bernd Bischl, 2016. Aslib: A benchmark library for algorithm selection, Artificial Intelligence, Volume 237, pp. 41-58

[3] www.research.ijcaonline.org/volume78/number14/pxc3891325.pdf.

[4] Elizabeth D. Dolan, Jorge J. Moré, 2002. Benchmarking optimization software with performance profiles, Mathematical programming, pp. 201-213.

[5] L. Barrett, A. Marathe, M. V. Marathe, D. Cook, G. Hicks, V. Faber, A. Srinivasan, Y. J. Sussmann, H. Thornquist, 2003. Statistical Analysis of Algorithms: A Case Study of Market-Clearing Mechanisms in the Power Industry, Journal of Graph Algorithms and Applications.

[6] X. S. Yang, 2014. Nature-Inspired Optimization Algorithms, Elsevier.

[7] Joe Zhu, 2014. Quantitative models for performance evaluation and benchmarking: data envelopment analysis with spreadsheets, Springer Volume 213.