## 

## 

## Algorithms & Data Structures

## ITCS 6160: PROJECT REPORT 2016

***Lakshmi Udupa (800956319)***

***Allen Sylvester ()***

***Shreyas S. Bhat (800958406)***

**Experimental Setup**

This experiment was conducted in the following environment:

* Operating System: Windows 10 Home 64-bit
* RAM: 16 GB
* HDD: 128GB SSD, 1TB secondary
* Processor: Intel x64 Core i7 6560U 2.20 Ghz 2.21 GHz
* Compiler: Eclipse IDE
* Language: Java

The purpose of this experiment is to compare the practical efficiency of the aforementioned algorithms using empirical analysis. Consequently, we chose to use the time the program implementing the algorithms in question. We were able to measure the running time of the code fragment by asking for the system’s time right before the fragment’s start (tstart) and just after its completion (tfinish) and then computing the differene between the two (tfinish - tstart).

In order to observe the behaviour of these algorithms for practical applications, we pterformed tests for various inputs with sizes ranging from 0 to 1,000,000.

**Algorithm Analysis**

The table below shows the theoretical best, worst and the average cases for the popular algorithms that we have chosen:

|  |  |  |  |
| --- | --- | --- | --- |
| Sorting Technique | Best Case | Average Case | Worst Case |
| Insertion Sort | Sequence of entries sorted in ascending order | Any random sequence of entries | Sequence of entries sorted in descending order |
| Merge Sort | Sequence of entries where the odd occurrences are always less than the even entries. | Any random sequence of entries | A sequence where the odd occurrences are always less than the even occurrences. When n is even |
| Counting Sort | Sequence of entries where the value of the maximum element in it is comparatively less | Any random sequence of entries | Sequence of entries where the value of the maximum element is high |

**Insertion Sort:**

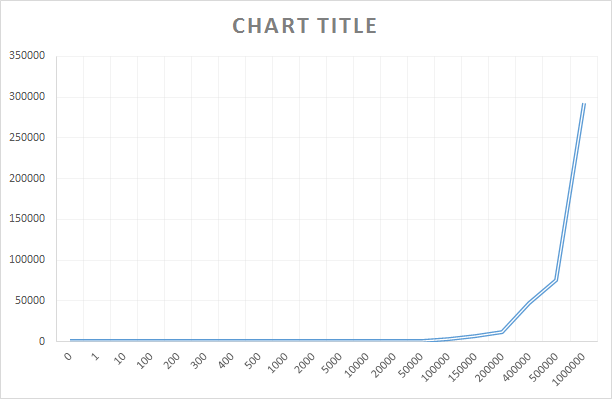
The insertion sort is the technique where the sequence is sorted by iterative insertion of the numbers into their appropriate place in a sequence.

The insertion sort technique sorts the given array in n^2 time complexity.

Time complexity - O(nlogn)

where n is the number of elements in the input.

|  |  |  |  |
| --- | --- | --- | --- |
| Insertion Sort | | | |
| N | Tavg | T1 | T2 |
| 0 | 0 | 0 | 0 |
| 1 | 0.009971 | 0.010666 | 0.009275 |
| 10 | 0.022029 | 0.021333 | 0.022725 |
| 100 | 0.280116 | 0.333449 | 0.226782 |
| 200 | 1.153159 | 0.942376 | 1.363941 |
| 300 | 2.124056 | 2.387476 | 1.860636 |
| 400 | 3.621794 | 4.443358 | 2.800229 |
| 500 | 4.02504 | 4.309329 | 3.740751 |
| 1000 | 6.948168 | 6.370313 | 7.526022 |
| 2000 | 10.86585 | 11.71013 | 10.02156 |
| 5000 | 20.35847 | 22.64068 | 18.07627 |
| 10000 | 40.39069 | 41.86431 | 38.91707 |
| 20000 | 131.5366 | 133.1625 | 129.9106 |
| 50000 | 754.8387 | 726.4624 | 783.215 |
| 100000 | 3009.742 | 3168.702 | 2850.782 |
| 150000 | 6728.601 | 6947.508 | 6509.694 |
| 200000 | 11909.17 | 12281.71 | 11536.63 |
| 400000 | 46708.68 | 46127.66 | 47289.69 |
| 500000 | 74972.06 | 77631.31 | 72312.82 |
| 1000000 | 292890.7 | 305425.3 | 280356.2 |

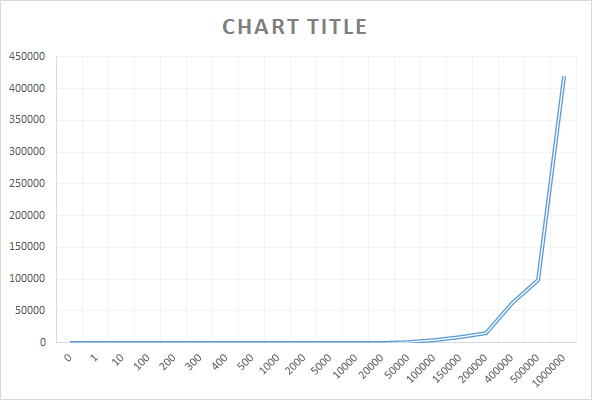


*Fig. time vs n graph for Insertion Sort (avg case)*

**Worst Case:**

The insertion sort takes maximum time to sort a sequence of entries when the entries are pre-sorted in descending order.

|  |  |  |  |
| --- | --- | --- | --- |
| Insertion Sort | | | |
| N | Tavg | T1 | T2 |
| 0 | 0 | 0 | 0 |
| 1 | 0.009971 | 0.010666 | 0.009275 |
| 10 | 0.019479 | 0.017623 | 0.021334 |
| 100 | 0.542376 | 0.539362 | 0.54539 |
| 200 | 1.213216 | 1.233622 | 1.19281 |
| 300 | 2.786317 | 2.956519 | 2.616114 |
| 400 | 4.88672 | 5.110256 | 4.663184 |
| 500 | 5.043242 | 5.525792 | 4.560691 |
| 1000 | 7.366718 | 6.717211 | 8.016225 |
| 2000 | 11.85135 | 10.64857 | 13.05413 |
| 5000 | 32.34571 | 33.55916 | 31.13226 |
| 10000 | 73.94242 | 71.55008 | 76.33477 |
| 20000 | 189.2045 | 192.1686 | 186.2403 |
| 50000 | 991.5828 | 996.621 | 986.5447 |
| 100000 | 3984.186 | 3960.327 | 4008.046 |
| 150000 | 8887.576 | 8884.527 | 8890.625 |
| 200000 | 15606.8 | 15768.89 | 15444.7 |
| 400000 | 62835.31 | 62403.77 | 63266.85 |
| 500000 | 98494.79 | 98692.14 | 98297.44 |
| 1000000 | 419915.8 | 427817.5 | 412014.2 |

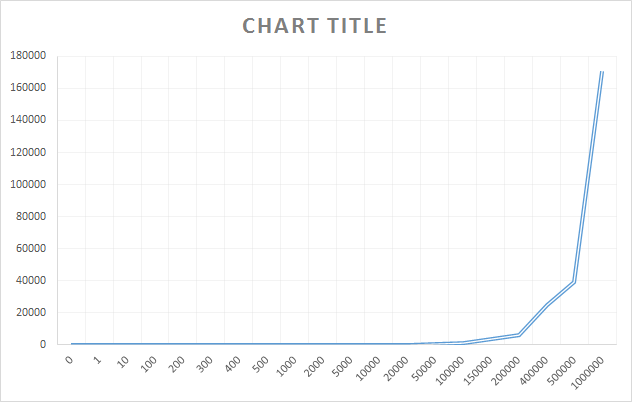


*Fig. time vs n graph for Insertion Sort (worst case)*

**Best Case:**

The insertion sort takes minimum time to sort a sequence of entries when the entries are pre-sorted in ascending order.

|  |  |  |  |
| --- | --- | --- | --- |
| Insertion Sort | | | |
| N | Tavg | T1 | T2 |
| 0 | 0 | 0 | 0 |
| 1 | 0.009971 | 0.010666 | 0.009275 |
| 10 | 0.016928 | 0.013913 | 0.019942 |
| 100 | 0.156986 | 0.153044 | 0.160928 |
| 200 | 0.546087 | 0.539362 | 0.552811 |
| 300 | 1.36255 | 1.566144 | 1.158956 |
| 400 | 2.374491 | 2.031766 | 2.717215 |
| 500 | 3.068519 | 3.086374 | 3.050664 |
| 1000 | 6.046835 | 5.791067 | 6.302603 |
| 2000 | 9.369267 | 9.488223 | 9.250311 |
| 5000 | 23.42004 | 22.599863 | 24.24021 |
| 10000 | 29.12484 | 25.581889 | 32.6678 |
| 20000 | 69.39495 | 69.560515 | 69.22939 |
| 50000 | 383.9582 | 386.21877 | 381.6977 |
| 100000 | 1502.404 | 1493.9019 | 1510.906 |
| 150000 | 3475.517 | 3520.1312 | 3430.904 |
| 200000 | 6168.135 | 6117.5748 | 6218.696 |
| 400000 | 24896.26 | 25280.605 | 24511.91 |
| 500000 | 39034.75 | 38980.5 | 39088.99 |
| 1000000 | 170733.3 | 165672.23 | 175794.3 |



*Fig. time vs n graph for Insertion Sort (best case)*

**Merge Sort:**

The merge sort is the technique where the sequence is sorted by recursive divide and conquer fashion. (i.e) The sequence or array of numbers are first divided into two sub arrays and merged in the required sorting order recursively.

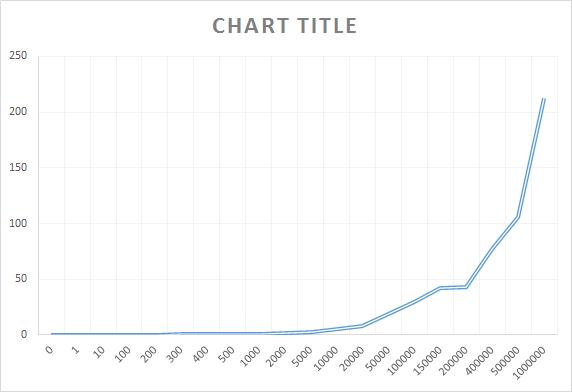
The merge sort technique sorts the given array in nlogn complexity.

Time complexity - O(nlogn)

where n is the number of elements in the input.

Random order input:

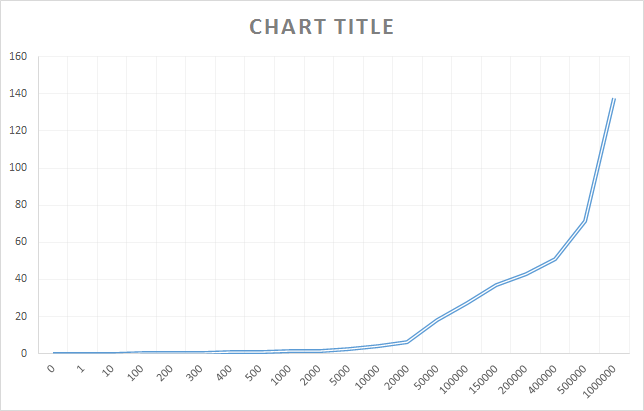
|  |  |  |  |
| --- | --- | --- | --- |
| Merge Sort | | | |
| N | Tavg | T1 | T2 |
| 0 | 0 | 0 | 0 |
| 1 | 0.017856 | 0.01716 | 0.018551 |
| 10 | 0.032232 | 0.028754 | 0.03571 |
| 100 | 0.168812 | 0.168348 | 0.169275 |
| 200 | 0.287073 | 0.267594 | 0.306551 |
| 300 | 0.473739 | 0.480463 | 0.467014 |
| 400 | 0.567883 | 0.624231 | 0.511535 |
| 500 | 0.751768 | 0.624231 | 0.879304 |
| 1000 | 1.263535 | 1.172404 | 1.354665 |
| 2000 | 1.666318 | 1.667245 | 1.66539 |
| 5000 | 2.475128 | 2.820171 | 2.130085 |
| 10000 | 5.316169 | 5.490546 | 5.141792 |
| 20000 | 7.717558 | 7.626659 | 7.808456 |
| 50000 | 18.72763 | 21.85691 | 15.59836 |
| 100000 | 29.95963 | 31.88449 | 28.03476 |
| 150000 | 42.42848 | 46.52564 | 38.33133 |
| 200000 | 43.28715 | 45.45665 | 41.11764 |
| 400000 | 76.80526 | 77.52202 | 76.08851 |
| 500000 | 106.1931 | 106.3832 | 106.0029 |
| 1000000 | 212.9232 | 207.5078 | 218.3386 |



*Fig. time vs n graph for Merge Sort (avg case)*

Ascending order input:

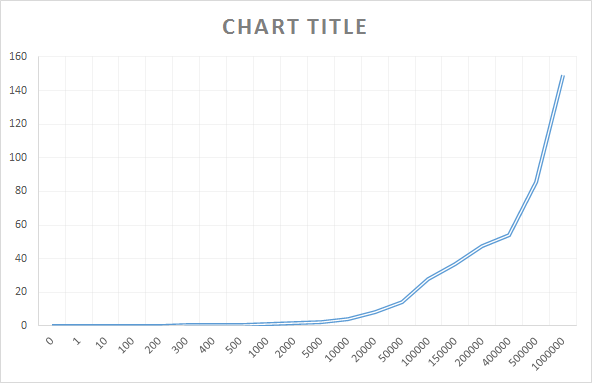
|  |  |  |  |
| --- | --- | --- | --- |
| Merge Sort | | | |
| N | Tavg | T1 | T2 |
| 0 | 0 | 0 | 0 |
| 1 | 0.017856 | 0.01716 | 0.018551 |
| 10 | 0.020638 | 0.021334 | 0.019942 |
| 100 | 0.128696 | 0.107594 | 0.149797 |
| 200 | 0.265739 | 0.270377 | 0.261101 |
| 300 | 0.406957 | 0.46284 | 0.351073 |
| 400 | 0.645333 | 0.681739 | 0.608927 |
| 500 | 0.827826 | 0.819014 | 0.836637 |
| 1000 | 1.262375 | 1.166375 | 1.358375 |
| 2000 | 1.437448 | 1.420984 | 1.453911 |
| 5000 | 2.428983 | 2.428287 | 2.429679 |
| 10000 | 4.059358 | 4.53472 | 3.583996 |
| 20000 | 6.06214 | 6.163009 | 5.96127 |
| 50000 | 18.36288 | 17.77993 | 18.94584 |
| 100000 | 27.56079 | 25.50166 | 27.56079 |
| 150000 | 36.94327 | 32.29817 | 41.58837 |
| 200000 | 42.75497 | 42.00437 | 43.50558 |
| 400000 | 51.18001 | 46.93932 | 55.4207 |
| 500000 | 71.40466 | 71.55889 | 71.25044 |
| 1000000 | 137.954 | 136.3304 | 139.5777 |



*Fig. time vs n graph for Merge Sort (ascending inputs)*

Descending order input:

|  |  |  |  |
| --- | --- | --- | --- |
| Merge Sort | | | |
| N | Tavg | T1 | T2 |
| 0 | 0 | 0 | 0 |
| 1 | 0.017856 | 0.01716 | 0.018551 |
| 10 | 0.023189 | 0.027362 | 0.019015 |
| 100 | 0.107826 | 0.104812 | 0.11084 |
| 200 | 0.268754 | 0.28058 | 0.256928 |
| 300 | 0.462841 | 0.45171 | 0.473971 |
| 400 | 0.544927 | 0.473043 | 0.616811 |
| 500 | 0.691014 | 0.596405 | 0.785622 |
| 1000 | 1.235941 | 1.181216 | 1.290666 |
| 2000 | 1.615999 | 1.747941 | 1.484056 |
| 5000 | 2.527534 | 2.61333 | 2.441737 |
| 10000 | 4.337619 | 4.572286 | 4.102952 |
| 20000 | 8.042659 | 7.539007 | 8.546311 |
| 50000 | 14.39952 | 14.68149 | 14.11755 |
| 100000 | 27.90142 | 31.72125 | 24.0816 |
| 150000 | 37.04831 | 36.33852 | 37.75811 |
| 200000 | 47.80633 | 41.96587 | 53.64679 |
| 400000 | 54.1523 | 53.9872 | 54.3174 |
| 500000 | 85.77615 | 88.7387 | 82.81361 |
| 1000000 | 149.0745 | 149.8149 | 148.3341 |



*Fig. time vs n graph for Merge Sort ( descending inputs)*

**Worst Case:**

The merge sort takes maximum time to sort a sequence of entries when the entries are permuted in the fashion that the odd indexed entries in the sequence are always greater than the even indexed entries. By this way the comparison activity during the merge operation involves all the numbers present in both the sub arrays, which are to be merged.

**Best Case:**

The merge sort takes maximum time to sort a sequence of entries when the entries are pre-sorted in ascending order, where the comparison operation involves only the sequence in the first sub-array and not in the second one.

**Counting Sort:**

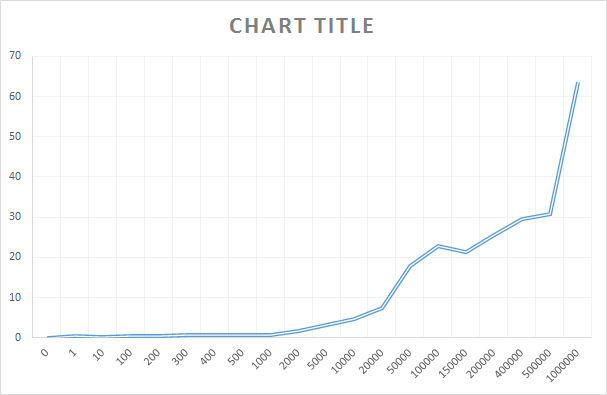
The merge sort is the technique where the sequence is sorted by counting the occurrences of the elements in the input sequence. The counting sort technique sorts the given array in linear time complexity. O(n+k)

where n is the number of elements in the input.

K is the number of digits in the maximum number in the input sequence

Random order input:

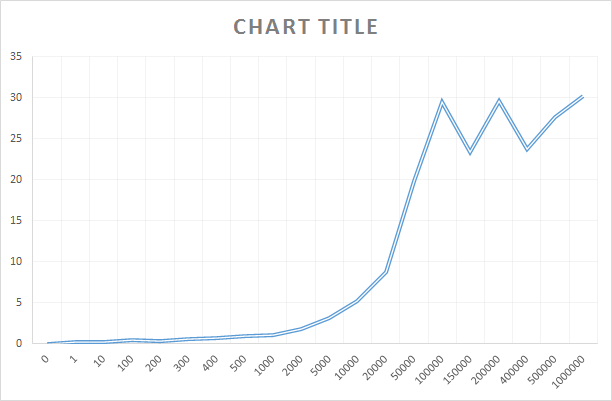
|  |  |  |  |
| --- | --- | --- | --- |
| Counting Sort | | | |
| N | Tavg | T1 | T2 |
| 0 | 0 | 0 | 0 |
| 1 | 0.30029 | 0.265739 | 0.33484 |
| 10 | 0.26458 | 0.255536 | 0.273623 |
| 100 | 0.463072 | 0.449391 | 0.476753 |
| 200 | 0.486493 | 0.436405 | 0.53658 |
| 300 | 0.553508 | 0.684058 | 0.422957 |
| 400 | 0.674319 | 0.770782 | 0.577855 |
| 500 | 0.764289 | 0.934956 | 0.593622 |
| 1000 | 0.639304 | 0.640927 | 0.63768 |
| 2000 | 1.694839 | 1.767419 | 1.622259 |
| 5000 | 3.099128 | 3.610895 | 2.58736 |
| 10000 | 4.604518 | 4.867705 | 4.34133 |
| 20000 | 7.275124 | 9.081499 | 5.468748 |
| 50000 | 17.85668 | 18.66387 | 17.04949 |
| 100000 | 22.7645 | 20.67847 | 24.85053 |
| 150000 | 21.44137 | 22.15836 | 20.72439 |
| 200000 | 25.47406 | 25.44601 | 25.50212 |
| 400000 | 29.43023 | 29.09215 | 29.76832 |
| 500000 | 30.84287 | 30.28218 | 31.40357 |
| 1000000 | 63.67174 | 64.2547 | 63.08878 |



*Fig. time vs n graph for Counting Sort (avg case)*

Ascending order input:

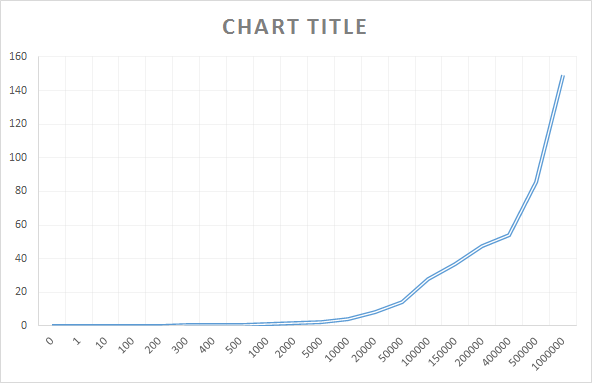
|  |  |  |  |
| --- | --- | --- | --- |
| Counting Sort | | | |
| N | Tavg | T1 | T2 |
| 0 | 0 | 0 | 0 |
| 1 | 0.30029 | 0.265739 | 0.33484 |
| 10 | 0.207768 | 0.152579 | 0.262956 |
| 100 | 0.427362 | 0.385391 | 0.469332 |
| 200 | 0.350145 | 0.373333 | 0.326956 |
| 300 | 0.566492 | 0.690086 | 0.442898 |
| 400 | 0.697275 | 0.549101 | 0.845449 |
| 500 | 1.02284 | 1.042086 | 1.003593 |
| 1000 | 1.130898 | 1.048115 | 1.21368 |
| 2000 | 1.856463 | 1.984462 | 1.728463 |
| 5000 | 3.136229 | 3.588635 | 2.683823 |
| 10000 | 5.268865 | 6.238139 | 4.29959 |
| 20000 | 8.75918 | 9.900048 | 7.618311 |
| 50000 | 19.91905 | 21.5921 | 18.24601 |
| 100000 | 29.48542 | 30.65504 | 28.3158 |
| 150000 | 23.40867 | 23.35163 | 23.46572 |
| 200000 | 29.58351 | 30.79371 | 28.37331 |
| 400000 | 23.78363 | 23.80056 | 23.7667 |
| 500000 | 27.63499 | 29.19789 | 26.07209 |
| 1000000 | 30.2041 | 30.00816 | 30.40004 |



*Fig. time vs n graph for Counting Sort (ascending inputs)*

Descending order input:

|  |  |  |  |
| --- | --- | --- | --- |
| Counting Sort | | | |
| N | Tavg | T1 | T2 |
| 0 | 0 | 0 | 0 |
| 1 | 0.30029 | 0.265739 | 0.33484 |
| 10 | 0.268986 | 0.274551 | 0.26342 |
| 100 | 0.433391 | 0.409971 | 0.456811 |
| 200 | 0.531942 | 0.645101 | 0.418782 |
| 300 | 0.557449 | 0.640463 | 0.474434 |
| 400 | 0.662725 | 0.823652 | 0.501797 |
| 500 | 0.944231 | 0.999883 | 0.888579 |
| 1000 | 0.672464 | 0.704464 | 0.640463 |
| 2000 | 1.528347 | 1.856926 | 1.199767 |
| 5000 | 3.264693 | 2.856809 | 3.672577 |
| 10000 | 4.29704 | 4.365446 | 4.228634 |
| 20000 | 8.850311 | 7.880804 | 9.819817 |
| 50000 | 22.35754 | 20.61679 | 24.0983 |
| 100000 | 33.74953 | 34.25481 | 33.24426 |
| 150000 | 32.04496 | 35.36321 | 28.7267 |
| 200000 | 27.28577 | 23.51812 | 31.05342 |
| 400000 | 25.82282 | 24.73505 | 26.91058 |
| 500000 | 24.24485 | 23.43789 | 25.0518 |
| 1000000 | 33.1546 | 33.0477 | 33.2615 |



*Fig. time vs n graph for Counting Sort (descending inputs)*