5/27/2019

CSP 2348 Assignment 2

A Mini Team Project

Samuel Morris – 10456767

Riegard Lemmer - 10424383

Sam

ECU JOONDALUP – CSP2348 – Data structures – assignment 2

# Executive Summary

Table of Contents

[Executive Summary 1](#_Toc9430874)

[Introduction 3](#_Toc9430875)

[Bubble and Heap Sort Algorithm Analysis 3](#_Toc9430876)

[Bubble sort algorithm analysis 3](#_Toc9430877)

[Heap sort algorithm analysis 3](#_Toc9430878)

[Summary of Sort Algorithm Complexities 4](#_Toc9430879)

[Algorithm Analysis by Experimental Studies 5](#_Toc9430880)

[Sequencing Array Sorting Algorithms Based on their Complexity 5](#_Toc9430881)

[Bubble Sort 5](#_Toc9430882)

[Selection Sort 5](#_Toc9430883)

[Insertion Sort 5](#_Toc9430884)

[. Merge Sort 5](#_Toc9430885)

# Introduction

# Bubble and Heap Sort Algorithm Analysis

## Bubble sort algorithm analysis

## Heap sort algorithm analysis

# Summary of Sort Algorithm Complexities

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sorting Algorithm | No. of comparisons | No. of  Copies | Time complexity | Space complexity |
| Bubble | ~ O(n2) | n(n-1)/2 | BC: O(N)  WC: O(n2) | O (1) |
| Selection | *~* | *~ 2n* | *BC:* O(n2)  WC: O(n2) | *O (1)* |
| Insertion | *~* n2 */ 4* | *~*n2 */ 4* | *BC:O(N)*  *WC:* O(n2) | *O (1)* |
| Merge | *~n log2n* | *~ 2n log2n* | *BC: O (n log2n)*  WC:*O (n log2n)* | *O(n)* |
| Quick | *\*BC: ~n log2n*  *\*WC: ~* n2 */2* | *BC: ~2n/3 log2n*  *WC: 0* | *BC: O (n log2n)*  *WC:* O (n2)*)* | *\*BC: O ()*  *\*WC: O(n)* |
| Heap | O () | O () | BC: O ()  WC: O () | WC: O (n) |

https://www.geeksforgeeks.org/analysis-of-different-sorting-techniques/

# Algorithm Analysis by Experimental Studies

# Sequencing Array Sorting Algorithms Based on their Complexity

## Bubble Sort

The bubble sort algorithm has an average performance of O(n2), the performance of this algorithm is relative to its size, so It would rarely be used to sort large data sets. Bubble sort works well on small sets of data where the inefficiency doesn’t affect the overall performance as much. The most efficient use of bubble sort would be for a list that is mostly sorted as time would be relative to the number of unsorted elements. One unsorted element would give 2n time, two would take 3n time and so on. In testing bubble ranked 6th in both time and comparisons making it the worst sorting algorithm.

## Insertion Sort

The insertion sort algorithm can be compared to an improved version of selection sort. It is quite efficient on smaller set of data and mostly sorted sets of data, because of this it is often used in conjunction with more complex algorithms. The downfall of insertion is that computationally expensive due to its shifting of all following elements when it shifts any element. In testing, Insertion ranked 5th in both time and comparisons making it the 2nd worst sorting algorithm.

## Selection Sort

The selection sort algorithm has O(n2) complexity, which is like the bubble sort as its performance is relative to its size. Selection sort is notable for how simple it is, and under specific circumstances has better performance than algorithms with greater complexity. It generally performs worse than much alike insertion sort. In testing, Selection ranks 4th for time and 1st every time for comparisons.

## . Quick Sort

The quick sort algorithm users a partition and pivot to sort elements and this gives it great linear time efficiency. Quick sort can be quite a complex algorithm but in practice it is quite fast. With its mediocre space usage of O (log n) all round quick sort is one of the top algorithms, although poor pivot selection can alter the performance significantly. In testing, Quick ranks 1st in time and 4th in comparisons.

## Merge Sort

The merge sort algorithms worst run time is O (n log n), this means that it scales well when sorting large sets of data. Merge sorts efficiency stems from when it combines list as this is one if its main operations. The O (n log n) run time is enables it to scale well with large sets of data. In testing, Merge ranks 2nd for lower values in time and 3rd for higher values, for comparisons it ranks 3rd every time.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A/N | 200 T | 200 C | 400 T | 400 C | 800 T | 800 C | 1000 T | 1000 C | 2000 T | 2000 C |
| B | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| I | 5 | 5 | 5 | 5 | 4 | 5 | 5 | 5 | 5 | 5 |
| S | 4 | 1 | 4 | 1 | 5 | 1 | 4 | 1 | 4 | 1 |
| H | 2 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 2 | 2 |
| Q | 1 | 4 | 1 | 4 | 1 | 4 | 1 | 4 | 1 | 4 |
| M | 3 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 3 | 3 |

*A =algorithm, N = number, C = comparisons, T = Time, B = Bubble, I = Insertion, S = Selection, H =Heap, Q= quick, M = Merge,*

|  |  |  |
| --- | --- | --- |
| A | Time (Ms) | Comparisons (Number) |
| B | 6th every instance | 6th every instance |
| I | 5th every instance  Except 3th Worst at 800 | 5th every instance |
| S | 4th most instances  Except 5th at 800, | 1st every instance |
| H | 2nd or 3rd every instance  2nd at 200,2000, 3rd at 400,800,1000 | 2nd Best every instance |
| Q | 1st every instance | 4th every instance |
| M | 2nd or 3rd every instance  2nd at 400,800,1000 3rd at 200,2000 | 3rd every instance |

<https://resources.saylor.org/wwwresources/archived/site/wp-content/uploads/2011/06/Sorting-Algorithm.pdf>

**https://www.quora.com/Which-sorting-algorithm-is-best-and-why**