International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 6 | ISSN : 2456-3307

A Study on Performance Analysis of Data Structures

Anne Srijanya K
Assistant Professor, CMR Engineering College, Hyderabad, Telangana, India

ABSTRACT

The Need for Data Structures is to organize data more efficaciously for complex applications. Many data structures exist but we need to select the confiscated data structure to meet the solution. A survey has been carried out on different types of data structures to identify their qualities and demarcations. This paper describes prominent data structures in a consistent manner to provide a concise comparison on performance of data structures. This paper presents a brief study on performance, time complexity and applications of data structures. This paper classifies data structures into seven categories that group them according to their time complexity.

Keywords : Data Structure, Time Complexity, Performance

I. INTRODUCTION

Data structures are used in the situations where logical relationship is required between the data elements in order to store the data. The logical or mathematical model of a particular organization of data is called as data structure [1]. Data structures are designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways. In computer programming, a data structure may be selected or designed to store data for the purpose of working on it with various algorithms [2]. Data structures provide a means to manage huge amounts of data efficiently. Some formal design methods and programming languages emphasize data structures, rather than algorithms. While Selecting a Data Structure first we need to analyze the problem to determine the resource constraints a solution must meet, and then determine the basic operations that must be supported. We need to calculate the resource constraints for each operation and at last select the data structure that best meets these requirements. In general we can change the data structures dynamically to prepare the data for a given algorithm. The implementation of a data structure usually requires writing a set of procedures that create and manipulate instances of that structure.

This paper gives clear description about the Data Structures, time complexity analysis and their applications. Section II presents related work that is carried out for analyzing the time complexity of data structures and also the classification based on their time complexity. Section III describes the time complexity analysis for insertion, deletion and search operations for different range of elements (N). Section IV presents the experimental results and discusses its performance over each scenario (different values of N). Section V presents real time applications and finally section VI presents the conclusions of the paper.

II. RELATED WORK

Premeditation and analysis of each and every data structure are done and based on their run time for each operation with different range of input data (N).

Data structures are classified in to seven categories that group them according to their time complexity. These data structures works efficiently according to the user's problem definition with unique performance. The following Table:1 shows the time complexity of each data structure which is further used for classification.

Table 1. Time Complexity of each Data Structure for Insertion, Deletion and Search operations

	Name of the Data Structure	**Time Complexity**		
		Insertion	**Deletion**	**Search**
Category 1	Stack	O(1)	O(1)	O(1)
	Queue			
Category 2	List	O(1)	O(N)	O(N)
	Linked list			
Category 3	Heap	O(LOG N)	O(LOG N)	O(N)
	Binary Heap			
Category 4	B-Tree	O(LOG N)	O(LOG N)	O(LOG N)
	2-3-4 Tree			
	B+ Tree			
	Red Black Tree			
	Splay Tree			
Category 5	Priority Queue	O(1)	O(LOG N)	O(1)
	Fibonacci Heap			
Category 6	Dequeue	O(1)	O(1)	O(N)
Category 7	Binary Tree	O(N)	O(N)	O(LOG N)

Table 2. Run time for different input data size (N)

Notation	**Complexity**	**Description**	**Run time (in Seconds) for different input data size N**		
			N=10000	**N=1000000**	**N=10000000**
O(1)	Constant	Constant number of operations, not depending on the input data size	Constant time	Constant time	Constant time
O(log N)	Logarithmic	Number of operations proportional to $log_2(N)$	10^{-5} secs (0.00001 secs)	$1.7*10^{-5}$ secs (0.000017 secs)	$2*10^{-5}$ secs (0.00002 secs)
O(N)	Linear	Number of operations proportional to the input data.	10^{-3} secs (0.001 secs)	0.1 secs	1 sec

III. TIME COMPLEXITY ANALYSIS

Time complexity analysis is required to predict the resources that the algorithm requires and to estimate the running time of an algorithm. Here Different values for N (input data size) were taken and analysed the run time for insertion, deletion and search operations for the data structures discussed above. The

Based on the above Table 2 for different values of N and the based on Table: 1, graphs are depicted for insertion, deletion and search operations to illustrate the performance of data structures graphically. Table: 2 illustrate the big O notation, complexity and run time in seconds.

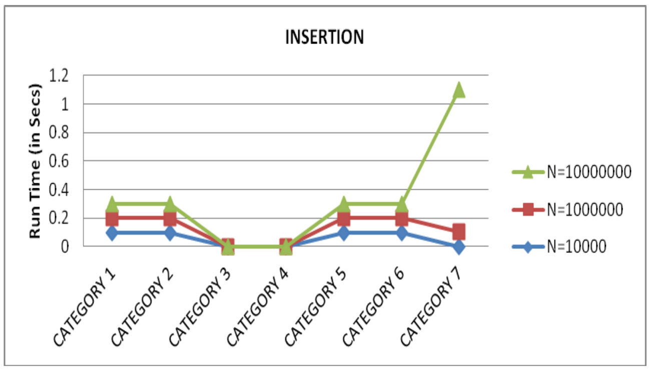

Figure 1. Comparison of Time Complexity of data structures for Inserting an element considering different values of N

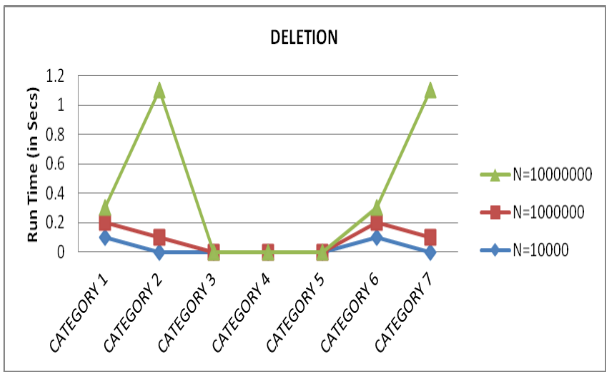

Figure 2. Comparison of Time Complexity of data structures for Deleting an element considering different values of N

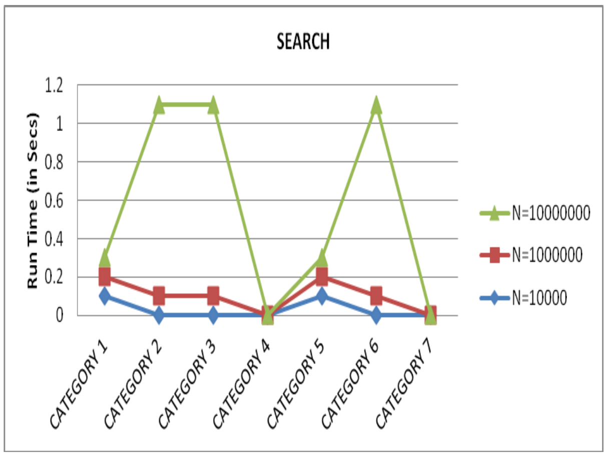

Figure 3. Comparison of Time Complexity of data structures for Searching an element considering different values of N

IV. OBSERVATIONS

Table 3 show the observations, an input explain how the run time will vary for different data sizes for performing each operation. The Table clearly describes how the run time will change when the size of the input data increases and it also tells which data structure is best suited for performing specific operations.

- Stack and queue take constant time for performing all the operations irrespective of size of input data.
- As the input data size increases the run time rapidly increases for searching an element in a linked list.
- The run time for inserting and deleting an element from the binary heap is very less, but as the input data size increases run time for searching an element rapidly increases. If the algorithm involves appending a lot of data then heaps can be used and is best suited if a large number of insertions and deletions are needed.

- Irrespective of the size of the input data the run time for the data structures which belongs to category 4 for performing insertion, deletion and search operations is very less. To keep data sorted; despite arbitrary inserts and deletes then a red black tree can be preferred. The run time for performing insertion and search operations is constant for priority queue and Fibonacci heap, but for deleting an element run time is very less. Priority queue can be used to order a list by some kind of importance.
- The run time for dequeue is constant for insertion and deletion operations, but as the size of the input data increases run time increases rapidly.
- In contrast to other data structures, for a binary tree the run time for inserting and deleting an element rapidly increases as the input data size increases. But once after inserting all the elements run time for searching an element is very less irrespective of size of input data. A binary tree is a good data structure to use for searching sorted data.

Table 3. Run time analysis for different input data size (N)

	Name of the Data Structure	**Run Time analysis for input data size (N)**					
		Insertion		**Deletion**		**Search**	
		For small input data size	**As the input data size increases**	**For small input data size**	**As the input data size increases**	**For small input data size**	**As the input data size increases**
Category 1	Stack	constant time	constant time	constant time	constant time	constant time	constant time
	Queue						
Category 2	List	constant time	constant time	less	increases rapidly	less	increases rapidly
	Linked list						
Category 3	Heap	very less	very less	very less	very less	less	increases rapidly
	Binary Heap						
Category 4	B-Tree	very less	very less	very less	very less	very less	very less
	2-3-4 Tree						
	B+ Tree						
	Red Black Tree						
	Splay Tree						
Category 5	Priority Queue	constant time	constant time	very less	very less	constant time	constant time
	Fibonacci Heap						
Category 6	Dequeue	constant time	constant time	constant time	constant time	less	increases rapidly
Category 7	Binary Tree	less	increases rapidly	less	increases rapidly	very less	very less

V. APPLICATIONS

- Stacks can be used for converting a decimal number into a binary number, Towers of Hanoi problem, parsing, and in runtime memory management.

- Queue can be used for Simulation, Ordered requests and Searches.
- Priority Queue can be used for Bandwidth management, Discrete event simulation, Dijkstra's algorithm, Huffman coding, A* and SMA* search algorithms and ROAM triangulation algorithm.
- Deque is used for the A-Steal job scheduling algorithm.
- List can be used to store a list of records. The items in a list can be sorted for the purpose of fast search (binary search).
- Linked Lists are used to implement several other common abstract data types, including stacks, queues, associative arrays, and symbolic expressions.
- Heap data structure is used in Heap sort, Selection algorithms, and Graph algorithms.
- Binomial Heaps are used in discrete event simulation and Priority queues.
- B-Trees have wide range of application in Data base, Dictionaries, 1-D range search.
- 2-3-4 Trees are used as in-memory data structures so user could memory program steps rather than disc accesses when evaluating and optimizing an implementation.
- Red-Black Trees are used in time-sensitive applications such as applications and in functional programming and to construct associative sets.
- Binary Tree are Used in *many* search applications where data is constantly entering/leaving, such as the map and set objects in many languages' libraries.

VI. CONCLUSION

This survey paper analyses the run time of data structures for performing different operations by considering different range of input data size.. The data structures described in this paper are prominent and efficient. The degree of speed-up in practice will depend upon the machines on which they are implemented.

During this survey, found some points that can be further explored in the future, such as to design algorithms and data structures in order to minimize the run time even for larger input data sizes and try to explore deeper in this research area.

VII. REFERENCES

[1]. Dr. N. Kashivishwanath "Data Structure Using C++", Laxmi publications

[2]. Data structure tutorial online], Available"http://searchsqlserver.techtarget.com /definition/data-structure".

[3]. Stack tutorial online] available "http://www.cprogramming.com/tutorial/comp utersciencetheory/stack.html"

[4]. Sartaj Sahni, "Data structures, Algorithms and Applications in C++".

[5]. Gopal, Arpita. "Magnifying Data Structures" PHI.

[6]. Donald Knuth. "The Art of Computer Programming",Volume1: Fundamental Algorithms, Third Edition. Addison-Wesley, 1997. Stacks, Queues, and Deques, Binary Trees.

[7]. "Definition of a linked list". National Institute of Standards and Technology. 2004-08-16. Retrieved 2004-12-14.

[8]. Parlante, Nick (2001). "Linked list basics". Stanford University. Retrieved 2009-09-21.

[9]. Goodrich, Michael T.; Tamassia, Roberto (2004). "7.3.6. Bottom-Up Heap Construction". Data Structures and Algorithms in Java (3rd ed.). pp. 338–341

[10]. Atkinson, M.D., J.-R. Sack, N. Santoro, and T. Strothotte. "Min-max heaps and generalized

priority queues." Programming techniques and Data structures.

[11]. Comer, Douglas (June 1979), "The Ubiquitous B-Tree", Computing Surveys 11 (2): 12137.

[12]. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. "Introduction to Algorithms", MIT Press and McGraw-Hill, Chapter 18: B-Trees, Chapter 13: Red–Black Trees Chapter 20: Fibonacci Heaps

[13]. Grama, Ananth (2004). "(2,4) Trees". CS251: Data Structures Lecture Notes. Department of Computer Science, Purdue University.

[14]. Ramakrishnan, R. and Gehrke, J. "Database Management Systems", McGraw-Hill Higher Education (2002), 3rd edition.

[15]. San Diego State University: CS 660: Red–Black tree notes, by Roger Whitne

[16]. Sleator, Daniel D.; Tarjan, Robert E. (1985), "Self-Adjusting Binary Search Trees", Journal of the ACM.