International Journal of Engineering Science Invention (IJESI)
ISSN (Online): 2319 – 6734, ISSN (Print): 2319 – 6726
www.ijesi.org || PP. 169-175

Comparative Study of Various Sorting Algorithms

Dr. Nagamani N P[1], Suvanta A Kulkarni[2], Sonika Srinivas[3], Reethu R[4], Pratiksha Khandelwal [5]

[1]*(Department of Information Science and Engineering, JSS Academy of Technical Education, Bengaluru, India)*
[2,3,4,5]*(Department of Information Science and Engineering, JSS Academy of Technical Education, Bengaluru, India)*

Abstract: *The overwhelming amount of data generated today will require a huge paradigm shift in the techniques for handling and managing that data. One of the important data processing operations is data sorting. Since in large files, swap time dominates in many applications, algorithms that minimize swap operations are often preferred over algorithms that focus solely on CPU time optimization. A sorting algorithm is a method for rearranging many items in a specific order, such as alphabetical, ascending, and descending. Sorting algorithms take lists of items as input, perform specific operations on those lists, and provide ordered arrays as output. These algorithms can be used to build disorganized records into files and make them easier to use. Sorting is very important in most algorithms that support file structure applications. An important objective is to facilitate access, search, import, and deletion of records. Therefore, the study points to the comparative study of several sorting algorithms to develop the most efficient sorting algorithm. The method used includes performance evaluation of quick, merge and heap sort techniques using time complexity as a measure of performance.*

Keywords - *Comparative Study, Heap Sort, Merge Sort, Quick Sort, Sorting Algorithms*

I. Introduction

Sorting refers back to the technique of arranging the given records in a specific format[1]. The sorting set of rulesspecifies the manner to set up the records in a specific order. A sorting set of rules is a set of rules that arranges factors of a listing in a sure order. Efficient sorting is crucial for decreasing the usage of different algorithms which require entering records to be in looking after lists. In the technology of computing, the problem of sortinghas grabbed the eyeballs of researchers, in all likelihood because of the complexity of fixing it efficiently regardless of its easy and acquainted statement. Among the authors of the first sorting algorithms around 1951 is Betty Holberton, who worked on ENIAC and UNIVAC[1]. Asymptotically the choicest algorithms had been recognized because of the mid-twentieth century – new algorithms are nevertheless being invented, with the extensively used Time sort courting in 2002, and the library kind being first posted in 2006.

Comparison sorting algorithms have an essential requirement of Ω(n log n) comparisons (a few enter sequences would require a couple of n log n comparisons, in which n is the wide variety of factors within the array to be looked after). Sorting algorithms are prevalent in introductory computer science classes, where the abundance ofalgorithms for the problem provides a gentle introduction to a variety of core algorithm concepts, such as big O notation, divide-and-conquer algorithms, data structures such as heaps and binary trees, randomized algorithms, best, worst and average case analysis, time–space tradeoffs, and upper and lower bounds[2].

Sorting small arrays optimally (in the fewest comparisons and swaps) or fast (i.e. contemplating gadget-precise details) continues to be an open studies hassle, with answers simplest recognized for terribly small arrays optimally(in the fewest comparisons and swaps) or fast (i.e. taking into account machine-specific details) is still an open research problem, with solutions only known for very small arrays (<20 elements). Similarly, optimal (by variousdefinitions) sorting on a parallel machine is an open research topic.

II. Basis of Classification

Computational complexity

It is defined as the best, worst and average case role in terms of the size of the list. For typical serial sorting algorithms, good behavior is O(n log n), and bad behavior is O(n2). The ideal behavior for a serial sort is O(n), but this is not possible in the average case. Optimal parallel sorting is O(log n).

Swaps for "in-place" algorithms.

Memory usage (and use of other computer resources). In particular, some sorting algorithms are "in-place". Strictly, an in-place sort needs only O(1) memory beyond the items being sorted; sometimes O(log n) additional memory is considered "in-place".

Recursion
Some algorithms can be recursive or non-recursive, while the other algorithms can be both recursive and non-recursive. For example, we have merge sort.

Stability
Stable sorting algorithms deal with the relative order of records with equal keys.

Comparison
Whether or not they are a comparison sort. A comparison sort checks the data mainly by comparing two elements with the use of a comparison operator.

Simple sorts
The two simple sorts are insertion sort and selection sort, both of which are effective on small data, due to low overhead, but not effective on large data. Insertion sort is commonly faster than selection sort, due to limited comparisons and good performance on most of the sorted data, and thus is approved in practice, but selection sortuses fewer writes.

Efficient sorts
The general sorting algorithms are consistently based on an algorithm with average time complexity and generallyworst-case complexity O(n log n), and the most common ones are heapsort, merge sort, and quicksort.

Each has pros and cons, with the most significant being that simple implementation of merge sort uses O(n) additional space, and simple implementation of quicksort has O(n2) worst-case complexity. These problems can be enhanced at the cost of a more complicated algorithm. While these algorithms are asymptotically effective on random data, for practical efficiency on real-world data various modifications are used.

III. Types Of Efficient Sorts

Quicksort
Quicksort is a dynamic sorting algorithm and is based on the segregation of an array of data into smaller arrays[3]. A large array is segregated into two arrays, one of which holds values smaller than the specified value, say pivot, based on which the partition is made, and another array holds values greater than the pivot value. Quick sort partitions an array and then calls itself recursively twice to sort the two resulting sub arrays. This algorithm is quite efficient for large-sized data sets as its average and worst case complexity are of O(n2), where n is the number of items.

Algorithm:
1. Choose an element in the list - this element is considered as the pivot.
2. Segregate the array of elements into two sets where the elements are less than the pivot and the elements whichare than the pivot (Note: quick sort method can be used to sort both integers and strings)
3. Reiterate the steps 1 and 2 on each of the two resulting sub arrays until each subarray has one or less elements.

Worst−case analysis[4]: The most asymmetric partition occurs when one of the sublists returned by the partitioning routine is of size n − 1. This may occur ifthe pivot happens to be the smallest or largest element in the list, or in some applications when all the elements are equal.If this happens constantly in every partition, then each recursive call processes a list of size one less than the previous list. Consequently, we can make n − 1 nested calls before we reach a listof size 1. This means that the call tree is a linear chain of n − 1 nested calls. Theith call does O(n − i) work to do the partition, and, so in that case quicksort takesO(n2) time.

Best-case analysis: In the most symmetrical case, each time we perform division we break down the list into two nearly equal pieces. This means each recursive call transforms a list of half the size. Therefore, we can make only log2 n nested calls before we reach a list of size 1. This means that the depth of the call tree is log2 n. But no two calls at the same level of the call tree process the same part of the original list; thus, each level of calls needs only O(n) time all together (each call has some constant overhead, but since there are only O(n) calls at each level, this is subsumed in the O(n) factor). The result is that the algorithm uses only O(n log n) time.

Average-case analysis: To sort an array of n distinct elements, quicksort takes O(n log n) time in expectation, averaged over all n! permutations of n elements with equal probability. We list here three common proofs to this claim providing different insights into quicksort's workings.

1. Percentiles
2. Recurrences
3. Binary search treee

.

Time Complexity for 3 cases of quick sort:

SI. No	No of elements	Best case	Average case	Worst case
1	8	24	24	64
2	4	8	8	16
3	10	33.22	33.22	100

Merge Sort

Merge Sort is a sorting algorithm based on the "divide and conquer" technique. It works by recursively dividing a problem into two or more related or similar subproblems, until they become simple enough to solve directly. The solutions are then combined to give a solution to the original problem. So Merge Sort first divides the array into equal halves and then combines them in sorted fashion.

In merge sort, the comparison takes place in the merging step, when two lists are combined to produce a sorted list. In the merging step, the first available element of each list is compared and the lower value is added to the output list. When one of the lists expires, the remaining elements of the reverse list are added to the output list.

Algorithm:

1. Start
2. Declare array and left, right, mid variable
3. Perform merge function.

mergesort(array,left,right) mergesort (array, left, right)if left > right
Return
mid= (left+right)/2 mergesort(array, left, mid) mergesort(array, mid+1, right)merge(array, left, mid, right)

4. Stop

Best-case, Worst-case and Average-case analysis:

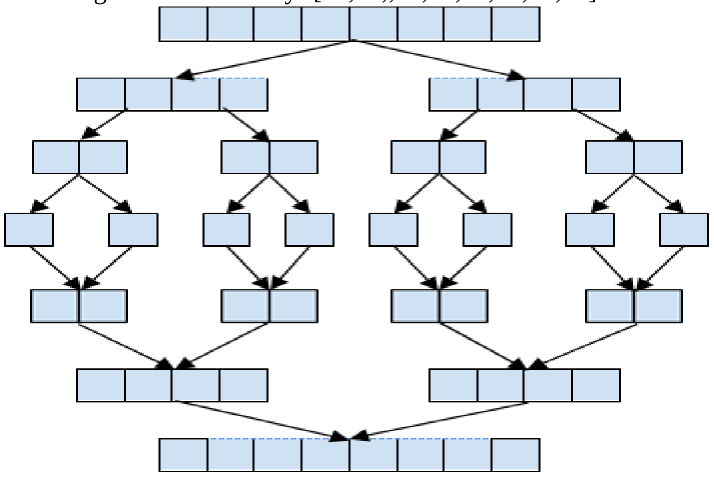
1. Worst Case: The case when all the array elements are sorted in the reverse order. Using Mastwe found thecomplexity of Merge sort in such case is Θ(nlogn).

2. Average Case: This is the case when the elements are partially sorted. The complexity of merge sort, in this case, is Θ(nlogn).

3. Best Case: This is when all the elements are already sorted, but still recursive calls are made thus, complexityis Θ(nlogn).

Example:

Step 1: Consider an array of numbers to be sorted - [82,90,10,12,15,77,55,23]. Step 2: Divide the given array into 2 equal parts -[{82,90,10,12},{15,77,55,23}]
Step 3: Repeat the process until the smallest units are obtained -[{82}{90}{10}{12}{15}{77}{55}{23}] Step 4: Once the smallest units are obtained,we start the merging process -[{82,90}{10,12}{15,77}{23,55}]Step 5: The elements are merged in ascending order.
Step 6: This is repeated until we get the sorted array -[10,12,,15,23,55,77,82,90]



Time Complexity for 3 cases of merge sort:

SI. No	No of elements (n)	Best case O(n log n)	Average case O(n log n)	Worst case O(n log n)
1	8	24	24	24
2	4	8	8	8
3	10	33.22	33.22	33.22

Heap sort

Heap sort is the sorting algorithm generally used as a comparison algorithm that sorts elements in a particular order[9]. The usually used orders are arranging numbers according to ascending or descending order and arranging alphabets according to the dictionary. This sorting is based on trees of data structures and it has two properties.

1. Shape property

This data structure is always a complete Binary tree which means it has all nodes with exactly 2 childrenexcept the last level which is not completely full.

2. Heap property

In this type either all nodes are greater than or equal to the Maximum heap or less than or equal to theminimum heap to each child node.

Algorithm[8]:

1: Create: Construct a heap from the available input data.We need to sort the elements in such a way that parent should be greater or equal to children and this is termed as max heap else we can sort it in min heap.

2: Swap Root. Replace the root element with the last element of the heap.3: Reduction of Heap Size. The size of the heap should be reduced by 1.
4: Re-Heapify. Heapify the elements which are left out into a heap of the new heap size by calling the heapify function on the root node.
5: Recursive calling. We need to repeat steps 2, 3 and 4 until the size of the heap is greater than 2.
Every time the last element is deleted from the heap once it contains the right element. This is recursively done until all the inputs are sorted. This happens when the size of the heap is decreased to 2, as it will satisfy the heap property, the first two elements will naturally be in order.

Worst-case analysis

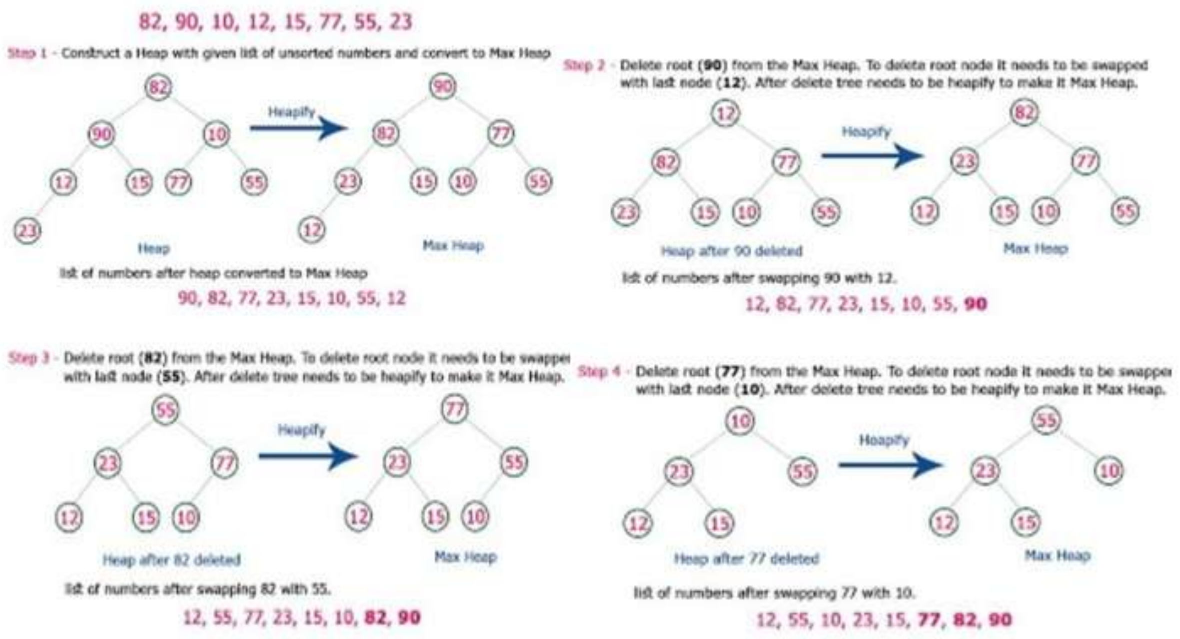
The worst case for heap sort may occur when all elements are different in the list.Hence, we might call a functionnamed max-heapify every time an element is deleted. In which the number of nodes is 'n'. The number of swaps to exclude every element of the heap would be log(n), as that is the maximum height of the heap structure. In considering for every node, the total number of would be n * (log(n)). Therefore, the runtime in the worst case will be O(n(log(n)).

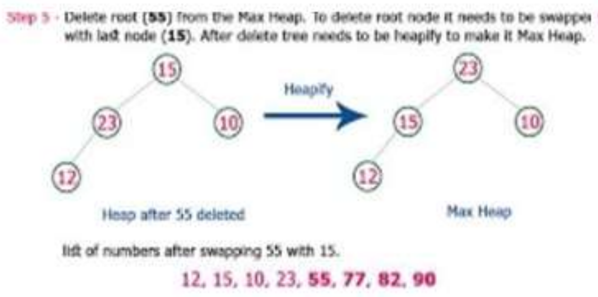
Best-case analysis[6]:

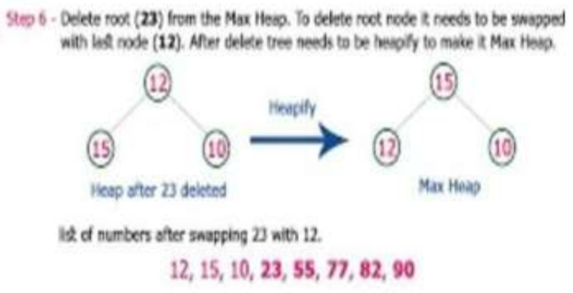
The best case for heap sorting happens when all items in the list to be sorted are indistinguishable, where n is thenumber of nodes. Deleting each node from the heap would take only a constant runtime, O(1). We don't swap theelements since all are the same. Since every node is included, the total number of moves would be n * O(1).Therefore, the runtime in the best case would be O(n)
Average-case analysis[5]:

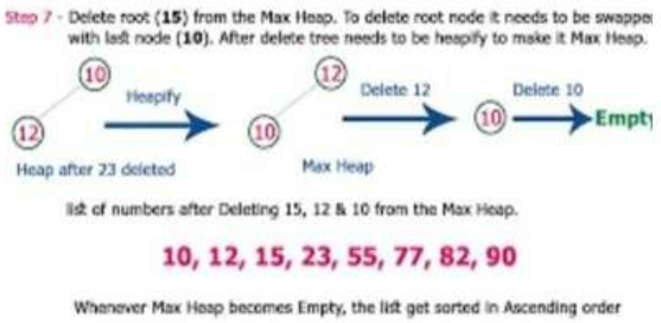
The total complexity will sum upto O(n) time and the addition or deletion of nodes in O(log(n)) time. In average time, we need to take available inputs, different elements considering the total nodes to be' n'', in such a case, the max-heapify function would need to perform:
1. log(n)/2 comparisons in the first iteration where only two values are considered.
2. log(n-1)/2 in the second iteration
3. log(n-2)/2 in the third iteration and this goes on for n iterations.

By adding the results of all iterations we will get a final average runtime to be O(n(log(n)).
Example:


82, 90, 10, 12, 15, 77, 55, 23
Step 1 - Construct a Heap with given list of unsorted numbers and convert to Max Heap
Heapify
Heap
Max Heap
list of numbers after heap converted to Max Heap
90, 82, 77, 23, 15, 10, 55, 12
Step 2 - Delete root (90) from the Max Heap. To delete root node it needs to be swapped with last node (12). After delete tree needs to be heapify to make it Max Heap.
Heapify
Heap after 90 deleted
Max Heap
list of numbers after swapping 90 with 12.
12, 82, 77, 23, 15, 10, 55, 90
Step 3 - Delete root (82) from the Max Heap. To delete root node it needs to be swapped with last node (55). After delete tree needs to be heapify to make it Max Heap.
Heapify
Heap after 82 deleted
Max Heap
list of numbers after swapping 82 with 55.
12, 55, 77, 23, 15, 10, 82, 90
Step 4 - Delete root (77) from the Max Heap. To delete root node it needs to be swapped with last node (10). After delete tree needs to be heapify to make it Max Heap.
Heapify
Heap after 77 deleted
Max Heap
list of numbers after swapping 77 with 10.
12, 55, 10, 23, 15, 77, 82, 90


Step 5 - Delete root (55) from the Max Heap. To delete root node it needs to be swapped with last node (15). After delete tree needs to be heapify to make it Max Heap.
Heapify
Heap after 55 deleted
Max Heap
list of numbers after swapping 55 with 15.
12, 15, 10, 23, 55, 77, 82, 90


Step 6 - Delete root (23) from the Max Heap. To delete root node it needs to be swapped with last node (12). After delete tree needs to be heapify to make it Max Heap.
Heapify
Heap after 23 deleted
Max Heap
list of numbers after swapping 23 with 12.
12, 15, 10, 23, 55, 77, 82, 90


Step 7 - Delete root (15) from the Max Heap. To delete root node it needs to be swapped with last node (10). After delete tree needs to be heapify to make it Max Heap.
Heapify
Delete 12
Delete 10
Empty
Heap after 23 deleted
Max Heap
list of numbers after Deleting 15, 12 & 10 from the Max Heap.
10, 12, 15, 23, 55, 77, 82, 90
Whenever Max Heap becomes Empty, the list get sorted in Ascending order

Time Complexity for 3 cases of Heap sort:

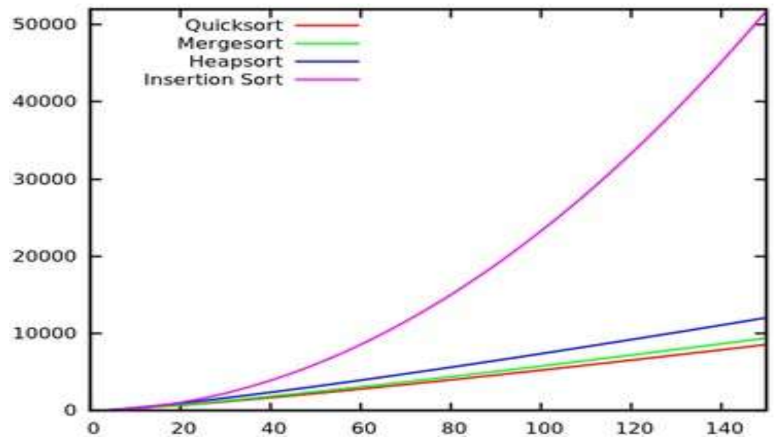
SI. No	No of elements (n)	Best case O(n log n)	Average case O(n log n)	Worst case O(n log n)
1	8	24	24	24
2	4	8	8	8
3	10	33.22	33.22	33.22

IV. Comparison

In computer science, best, worst, and average cases of a given algorithm express what the resource usage is at least, at most and on average, respectively. Usually the resource being considered is running time, i.e. time complexity, but it could also be memory or other resource. In real-time computing, the worst-case execution time is often of particular concern, since it is important to know how much time might be needed in the worst case to guarantee that the algorithm will always finish on time. [6]

Table. 1 Comparison of Various Sorting Algorithms Based On Time Complexity

Algorithm	**Best Case**	**Average Case**	**Worst Case**
Quick Sort	Ω(n log(n))	θ(n log(n))	O(n^2)
Merge Sort	Ω(n log(n))	θ(n log(n))	O(n log(n))
Heap Sort	Ω(n log(n))	θ(n log(n))	O(n log(n))



V. Conclusion

Sorting plays a very crucial role when it comes to extracting deep insights from existing data.Some quick optimizations can also be done to the proposed sorting algorithms to make it even more efficient and to save on comparisons. In this paper, we have discussed some well known sorting algorithms. Depending on the type and size of the data, different algorithms are to be selected and compared. The analysis of these algorithms are basedon the same data and on the same computer. Doing more comparisons between more different sorting algorithms is required since no specific algorithm can solve any problem in absolute. The results show that Quick sort is efficient for both small and large integers. Quick sort is significantly faster in practice than other O(n log n) algorithms. Recommender systems and image mining applications incorporate sorting algorithms[10].

References

[1]. "Meet the 'Refrigerator Ladies' Who Programmed the ENIAC". Mental Floss. 2013-10-13.Retrieved 2016-06-16.

[2]. https://en.wikipedia.org/wiki/Sorting_algorithm

[3]. https://cs.stanford.edu/people/eroberts/courses/soco/projects/2008-09/tony-hoare/quicksort.html

[4]. https://en.wikipedia.org/wiki/Quicksort#:~:text=Quicksort%20is%20a%20divide%2Dand,sometimes%20called%20partition%2Dexchange%20sort.

[5]. https://www.geeksforgeeks.org/application-and-uses-of-quicksort/#:~:text=It%20is%20an%20in%2Dplace,sort%20is%20used%20for%20sorting.

[6]. https://www.researchgate.net/publication/320715254_Heap_Sorting_Based_on_Array_Sorting

[7]. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.845.6329&rep=rep1&type=pdf

[8]. https://cs.fit.edu/~pkc/classes/writing/hw13/luis.pdf

[9]. https://medium.com/karuna-sehgal/a-simplified-explanation-of-merge-sort-77089fe03bb2

[10]. https://www.programiz.com/dsa/heap-sort

[11]. Kar, A.K., Kushwaha, A.K. Facilitators and Barriers of Artificial Intelligence Adoption in Business – Insights from Opinions Using Big Data Analytics. Inf Syst Front (2021). https://doi.org/10.1007/s10796-021-10219-4