**Project Title: Hybrid and Adaptive Sorting Algorithms**

**Project components**

Shiv Patel

CMPSC-463

1. Insertion Sort

In insertion sort we are using one for loop to go through each element in the array from second element to last element. Then inside the for loop we are using inner while loop to place each iteration element into the correct sorting position in the sorted portion of the array. For each element in the unsorted portion, the algorithm compares it to the elements in the sorted portion, moving from right to left. The goal is to find the correct position for the current element within the sorted portion. Time complexity of bubble sort is O(n2)

1. Merge Sort

Merge sort is a comparison-based algorithm. Merge sort is not an in-place algorithm and it is an out-of-place algorithm where merge sort doesn’t change the input array and it needed additional space to store the sorted array. Similar to quick sort it is a very efficient algorithm. Merge sort uses divide and conquer approach where problem is divided into sub problems then solve each sub problem and finally sums them up to find the final solution. Merge sort is highly efficient and time complexity of the merge sort is O(n log n). Main steps of the merge sort are described in below.

* Divide - Input array is divided into 2 equal-sized sub arrays at first. This process is executed recursively until each sub array size has only 1 or 0 element.
* Conquer – In this step sorting step is executed. The algorithm merges the smaller sub-arrays together in a sorted way. In merge step adjacent sub-arrays are compared and arranges the values in sorted order. This merge process is continuing until the whole array is sorted.

**Algorithm Design**

Here from the 2 topics hybrid sorting and adaptive sorting algorithms I have researched about hybrid sorting algorithm and I’m going to explained that in below.

**Hybrid Sorting Algorithms**

Hybrid sorting algorithms are built by combining 2 more general sorting algorithms. Here we can improve the strengths of the merge sort and quick sort by introducing other sorting algorithms like insertion sort, heap sort etc. Quick sort is well known for solve average case scenarios very efficiently while merge sort is well known for solving worst-case scenarios. One of the hybrid sorting algorithms is Tim Sort and that is explain in below.

**Tim Sort**

Tim sort is one of the hybrid sorting algorithms which is built by combining merge sort and insertion sort algorithms. It is a very efficient algorithm compared to merge sort and insertion sort when considered individually. Python sorted() function and Java Arrays.sort() functions built using Tim sort.

Let’s go through the tim sort briefly.

Since tim sort using merge sort, tim sort also uses divide and conquer approach. In tim sort we are dividing the input array into small parts and then individually sorted using insertion sort. We are applying insertion sort for very small chunks only since it is performs well on nearly sorted data.

Then as in the merge sort we need to execute the merge process in order combine the small chunks such a way that the overall array remains sorted.

Tim Sort using a method called “galloping” to identify and merge chunks efficiently. In there we are comparing the last value of one run with the first value of next run and if the next run first value is greater it is counting the number of values that can be merged in a single step. Therefore, it is hugely increasing the algorithm efficiency.

Tim sort has the ability to automatically adapt to the new data irrespective of the characteristics. Insertion sort can take the benefit with the minimal work if the chunk is partially sorted.

**Benchmarking Results**

We conducted rigorous testing and benchmarking experiments to assess the performance of our hybrid and adaptive sorting algorithms. The tests were executed on various input data scenarios, including:

1. Randomly shuffled data.
2. Nearly sorted data.
3. Inversely sorted data.

The benchmarking results for each sorting algorithms summarized in below

Benchmarking Results for Sorting algorithms and comparison

Execution time measured in milliseconds (ms) and memory usage measured in MiB (A mebibyte is a unit of measure. It represents 1024 \* 1024 bytes. Mebibyte is almost close to the MegaByte - MB)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input Data/Sorting Algorithm | Insertion Sort | | Merge Sort | | Tim Sort | |
| ms | MiB | ms | MiB | ms | MiB |
| Random Data | 0.000079 | 47.5 | 0.001504 | 47.6 | 0.000470 | 47.6 |
| Nearly Sorted | 0.000082 | 47.6 | 0.000942 | 47.6 | 0.000396 | 47.6 |
| Inversely Sorted | 0.000084 | 47.6 | 0.001159 | 47.6 | 0.000396 | 47.6 |

Also from the algorithms we can see that the results are correct and accurate.

**Performance Analysis**

According to the above results we can clearly see that tim sort algorithm is very efficient that quick sort and merge sort algorithms. Even though tim sort worst case time complexity also O(n log n) as quick sort and merge sort, tim sort uses insertion sort to minimal the processing needed and faster the process. Therefore, we can conclude that hybrid sorting algorithms performs way better than individual algorithms. Also, there adaptability is very high. Real world applications like python sorted and Java’s Arrays.Sort() uses this tim sort algorithm to sort the arrays very efficiently.