Program Structures and Algorithms

Spring 2024

NAME: Gaurav Popat Gunjal

NUID: 002820367

GITHUB LINK: https://github.com/gunjalga/INFO6205

**Task: ASSIGNMENT 6 (Hits as time predictor)**

**Charts:**

**A graph of different colored lines

Description automatically generated**

**A graph of different colored lines

Description automatically generated**

**A graph of different colored lines

Description automatically generated**

Spreadsheet for the data is in the repository named as Assignment 6.xlsx

1.Merge Sort:

* Merge sort is a divide-and-conquer algorithm that recursively splits the array into halves, sorts each half, and then merges the sorted halves.
* While merge sort has a relatively smaller number of hits compared to some other sorting algorithms, such as quick sort, it often involves a significant number of copies.
* Each merge operation requires copying elements into temporary arrays, which can contribute significantly to the overall runtime, especially for large arrays.
* Despite having fewer hits, the overhead of copying elements can still make merge sort's runtime relatively high, especially compared to algorithms where the number of copies is minimal.

2. Quick Sort:

* Quick sort is also a divide-and-conquer algorithm but works by selecting a pivot element and partitioning the array around the pivot, such that elements less than the pivot are on one side and elements greater than the pivot are on the other side.
* Quick sort tends to have a higher number of hits compared to merge sort because it involves more frequent swapping of elements during the partitioning process.
* However, quick sort typically avoids copying elements as extensively as merge sort, leading to potentially lower overhead in terms of copies.
* Despite having more hits than merge sort, quick sort's runtime might be lower due to its efficient partitioning and reduced copying overhead.

3.Heap Sort:

* Heap sort involves building a heap data structure from the input array and then repeatedly extracting the maximum (for a max-heap) element from the heap and reconstructing the heap until the array is sorted.
* Heap sort often involves a large number of hits because each heap operation, such as heapifying the array or extracting elements from the heap, requires accessing and modifying elements in the array.
* Similar to quick sort, heap sort generally doesn't involve significant copying overhead, as the main operations revolve around rearranging elements within the original array.
* Therefore, the number of hits (array accesses) plays a crucial role in determining heap sort's runtime, and optimizations to minimize these accesses can lead to performance improvements.

**Conclusion:** In summary, while each sorting algorithm has its unique characteristics, the number of hits (array accesses) remains a crucial factor in determining their runtime performance. We can also see from the graphs and spreadsheets that hits are the major contributor to the run time in sorting algorithms and after that there is addition to the time if the algorithm includes a copying overhead.