CS 5120 Project Report

Akoh Atadoga

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Identical data | Random data | Sorted data | Reverse Sorted data |
| Heap Sort | Ο (n ) | Ο (n ) | Ο (n ) | Ο (n ) |
| Insertion Sort | Ο (n) | Ο (n2) | Ο (n) | Ο (n2) |
| Merge sort | Ο (n ) | Ο (n ) | Ο (n ) | Ο (n ) |
| 3-way merge sort | Ο (n ) | Ο (n ) | Ο (n ) | Ο (n ) |

Figure. 1 Table showing the theoretical time it takes each algorithm to sort an array of input size, n

Theoretically, as seen from the table above, for each set of data, ranking all the algorithms from the fastest to slowest:

* Identical data: 3-way merge sort, merge sort, heap sort then insertion sort
* Random data: 3-way merge sort, merge sort, heap sort then insertion sort
* Sorted data: 3-way merge sort, merge sort, heap sort then insertion sort
* Reverse sorted data: 3-way merge sort, merge sort, heap sort then insertion sort

NB: Heap sort and merge sort takes the same amount time; however, merge sort takes more space than heap sort.

**Identical data**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 20,000 | 40,000 | 60,000 | 80,000 | 100,000 |
| Heap sort | 36.14306449890137 | 88.97089958190918 | 142.78101921081543 | 125.33116340637207 | 176.19609832763672 |
| Insertion sort | 7.437705993652344 | 16.07489585876465 | 37.901878356933594 | 29.88409996032715 | 40.01307487487793 |
| Merge sort | 338.75203132629395 | 504.3458938598633 | 864.3879890441895 | 1265.1069164276123 | 1416.2018299102783 |
| 3-way merge sort | 309.48710441589355 | 671.3290214538574 | 1189.9793148040771 | 1274.7530937194824 | 2500.498056411743 |

Figure. 2a Table showing the time it takes each algorithm to sort an array of the same values

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 20,000 | 40,000 | 60,000 | 80,000 | 100,000 |
| Heap sort | 0 | 0 | 0 | 0 | 0 |
| Insertion sort | 0 | 0 | 0 | 0 | 0 |
| Merge sort | 139216 | 298432 | 469008 | 636864 | 815024 |
| 3-way merge sort | 185280 | 395530 | 615840 | 862236 | 1092924 |

Figure. 2b Table showing the number of comparisons it takes each algorithm to sort an array of the same values

Figure 2c

Analysis and discussion: It can be seen from the graph in Figure 2c and table in Figure 2a that in sorting identical data, Insertion sort is the fastest for the given input size followed by heapsort while 3-way merge sort is the slowest which is not in accordance with the theoretical performance. The reason for this behavior is that in insertion sort, there is no comparison (as shown in Figure 2b) amongst elements of the array saving time while in merge sort and 3-way merge sort, the algorithm still performs recursion and loops over the elements the array even though they are similar making it to perform slower that insertion sort and merge sort. Though heap sort and insertion sort do not perform comparison, heap sort takes longer because of the time it takes to make a heap and loop over each element.

**Random data**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 20,000 | 40,000 | 60,000 | 80,000 | 100,000 |
| Heap sort | 351.79901123046875 | 751.2271404266357 | 1251.682996749878 | 2755.9080123901367 | 1929.973840713501 |
| Insertion sort | 38084.548234939575 | 160945.7242488861 | 402914.8099422455 | 787838.9632701874 | 1087545.9368228912 |
| Merge sort | 289.1521453857422 | 560.9371662139893 | 914.2529964447021 | 1285.233974456787 | 1550.8859157562256 |
| 3-way merge sort | 279.9191474914551 | 512.5730037689209 | 909.6128940582275 | 1372.4210262298584 | 1597.8450775146484 |

Figure. 3a Table showing the time it takes each algorithm to sort an array of the random values

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 20,000 | 40,000 | 60,000 | 80,000 | 100,000 |
| Heap sort | 2525941 | 3311449 | 4542804 | 6234487 | 2163095 |
| Insertion sort | 100519486 | 399123865 | 897357824 | 1592845170 | 2499108289 |
| Merge sort | 260950 | 561756 | 877258 | 1203417 | 1536809 |
| 3-way merge sort | 185280 | 395530 | 615840 | 862236 | 1092924 |

Figure. 3b Table showing the number of comparisons it takes each algorithm to sort an array of the random values

Figure. 3c

Analysis and discussion: It can be seen from the graph in Figure 3c and Figure 3a that in sorting random data, 3-way merge sort is the fastest for the given input size followed by merge sort then heap sort while insertion sort is the slowest which is in accordance with the theoretical performance. This behavior is in accordance with Figure 1 which shows the expected running time of each algorithm.

**Reverse sorted data**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 20,000 | 40,000 | 60,000 | 80,000 | 100,000 |
| Heap sort | 310.5359077453613 | 697.6041793823242 | 1043.5819625854492 | 1487.5540733337402 | 1873.816967010498 |
| Insertion sort | 75270.50590515137 | 305333.94408226013 | 689831.8719863892 | 1242673.5479831696 | 2036792.4201488495 |
| Merge sort | 228.14011573791504 | 466.7167663574219 | 731.8370342254639 | 980.8318614959717 | 2444.6537494659424 |
| 3-way merge sort | 237.37668991088867 | 541.079044342041 | 814.0349388122559 | 1157.8197479248047 | 2303.114414215088 |

Figure. 4a Table showing the time it takes each algorithm to sort an array with reverse sorted values

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 20,000 | 40,000 | 60,000 | 80,000 | 100,000 |
| Heap sort | 8596556 | 9329782 | 10478729 | 12060592 | 8259327 |
| Insertion sort | 199989821 | 799979179 | 1799968208 | 3199956822 | 4999945046 |
| Merge sort | 148077 | 316336 | 486283 | 673360 | 855517 |
| 3-way merge sort | 185280 | 395530 | 615840 | 862236 | 1092924 |

Figure. 4b Table showing the number of comparisons takes each algorithm to sort an array with reverse sorted values

Figure. 4c

Analysis and discussion: It can be seen from the graph in Figure 4c and table 4a that in sorting reverse sorted data, merge sort is the fastest for the given input size followed by 3-way merge sort then heap sort while insertion sort is the slowest which is partially in accordance with the theoretical performance. It is partial because theoretically 3-way merge sort should perform faster that merge sort. The reason for this behavior is due to the number of comparisons shown in the Figure 4b. The number of comparisons in 3-way merge sort is more than merge sort.

**Sorted data**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 20,000 | 40,000 | 60,000 | 80,000 | 100,000 |
| Heap sort | 389.3427848815918 | 748.8059997558594 | 1154.233694076538 | 1580.4920196533203 | 2047.4510192871094 |
| Insertion sort | 8.339881896972656 | 15.529870986938477 | 20.378828048706055 | 28.99789810180664 | 37.58120536804199 |
| Merge sort | 216.49575233459473 | 485.86320877075195 | 753.6399364471436 | 1028.2139778137207 | 1278.336763381958 |
| 3-way merge sort | 258.45909118652344 | 576.012134552002 | 850.0070571899414 | 1229.7911643981934 | 1570.695161819458 |

Figure. 5a Table showing the time it takes each algorithm to sort an array of the sorted values

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 20,000 | 40,000 | 60,000 | 80,000 | 100,000 |
| Heap sort | 14751430 | 15590017 | 16904056 | 18704227 | 14361967 |
| Insertion sort | 0 | 0 | 0 | 0 | 0 |
| Merge sort | 139216 | 298432 | 469008 | 636864 | 815024 |
| 3-way merge sort | 185280 | 395530 | 615840 | 862236 | 1092924 |

Figure. 5b Table showing the number of comparisons it takes each algorithm to sort an array of the sorted values

Figure 5c.

Analysis and discussion: It can be seen from the graph in Figure 5c and table in Figure 5a that in sorting data that has already has been previously sorted, Insertion sort is the fastest for the given input size followed by merge sort then 3-way merge sort while heap sort is the slowest which is not in accordance with the theoretical performance. The reason for this behavior is that in insertion sort, there is no comparison (as shown in Figure 5b) amongst elements of the array since they are already sorted saving time while in merge sort, 3-way merge sort and heap sort, the algorithm still performs recursion and loops over the elements the array even though they are already sorted making it to perform slower that insertion sort.

**Conclusion**

From the comparisons above, it can be seen that there is a direct relation between the number of comparisons between elements of the array and time taken to sort the data. The theoretical analysis for the algorithms as shown in Figure. 1 holds best for random data and reverse sorted data. In theory heap sort and merge sort take the same amount of time; however, in practical merge sort performs more comparisons on elements of array than merge sort hence heap sort runs slower than merge sort. In practical terms when the size of the input is unknown, I would go with merge sort.