# Assignment – 4 Report

Time complexity for Partially Sorted Array Using Knuth Shuffle Algorithm:

Insertion Sort:

|  |  |  |
| --- | --- | --- |
| Insertion Sort |  |  |
| S.No. | Input Size | Time (In milliseconds) |
| 1 | 1000000 | 45609 |
| 2 | 500000 | 25516 |
| 3 | 200000 | 8405 |
| 4 | 100000 | 2050 |
| 5 | 75000 | 1232 |
| 6 | 50000 | 548 |
| 7 | 25000 | 126 |
| 8 | 5000 | 8.006002 |
| 9 | 1000 | 1.692191 |
| 10 | 9 | 0.008139 |

Selection Sort:

|  |  |  |
| --- | --- | --- |
| Selection Sort |  |  |
| S.No. | Input Size | Time (In milliseconds) |
| 1 | 1000000 | 221654 |
| 2 | 500000 | 117219 |
| 3 | 200000 | 18228 |
| 4 | 100000 | 4666 |
| 5 | 75000 | 2768 |
| 6 | 50000 | 1219 |
| 7 | 25000 | 283 |
| 8 | 5000 | 13.575228 |
| 9 | 1000 | 1.495036 |
| 10 | 9 | 0.015386 |

Quick Sort:

|  |  |  |
| --- | --- | --- |
| Quick Sort |  |  |
| S.No. | Input Size | Time (In milliseconds) |
| 1 | 1000000 | 148 |
| 2 | 500000 | 69 |
| 3 | 200000 | 27 |
| 4 | 100000 | 14 |
| 5 | 75000 | 12 |
| 6 | 50000 | 7 |
| 7 | 25000 | 4 |
| 8 | 5000 | 1.259472 |
| 9 | 1000 | 0.325487 |
| 10 | 9 | 0.014186 |
|  |  |  |

Merge Sort:

|  |  |  |
| --- | --- | --- |
| Merge Sort |  |  |
| S.No. | Input Size | Time (In milliseconds) |
| 1 | 1000000 | 199 |
| 2 | 500000 | 88 |
| 3 | 200000 | 33 |
| 4 | 100000 | 16 |
| 5 | 75000 | 12 |
| 6 | 50000 | 10 |
| 7 | 25000 | 4 |
| 8 | 5000 | 1.451363 |
| 9 | 1000 | 0.508196 |
| 10 | 9 | 0.017813 |

Heap Sort:

|  |  |  |
| --- | --- | --- |
| Heap Sort |  |  |
| S.No. | Input Size | Time (In milliseconds) |
| 1 | 1000000 | 392 |
| 2 | 500000 | 117 |
| 3 | 200000 | 41 |
| 4 | 100000 | 17 |
| 5 | 75000 | 13 |
| 6 | 50000 | 12 |
| 7 | 25000 | 3 |
| 8 | 5000 | 1.619394 |
| 9 | 1000 | 0.422422 |
| 10 | 9 | 0.01417 |

Combine Sort:

|  |  |  |
| --- | --- | --- |
| Combine Sort |  |  |
| S.No. | Input Size | Time (In milliseconds) |
| 1 | 1000000 | 168 |
| 2 | 500000 | 75 |
| 3 | 200000 | 29 |
| 4 | 100000 | 14 |
| 5 | 75000 | 10 |
| 6 | 50000 | 7 |
| 7 | 25000 | 6 |
| 8 | 5000 | 0.848907 |
| 9 | 1000 | 0.366248 |
| 10 | 9 | 0.09714 |

Space Complexity of all Sorting Algorithm:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm Names | Best Case | Worst Case | Average Case | Space Complexity |
| Quick | O(n log(n)) | O(n^2) | O(n log(n)) | O(log(n)) |
| Insertion | O(n) | O(n^2) | O(n^2) | O(1) |
| Selection | O(n^2) | O(n^2) | O(n^2) | O(1) |
| Merge | O(n log(n)) | O(n log(n)) | O(n log(n)) | O(n) |
| Heap | O(n log(n)) | O(n log(n)) | O(n log(n)) | O(1) |

Time complexity for Sorted Array:

Insertion Sort:

|  |  |  |
| --- | --- | --- |
| Insertion Sort |  |  |
| S.No. | Input Size | Time (In nanoSeconds) |
| 1 | 5000 | 285702 |
| 2 | 1000 | 68020 |
| 3 | 9 | 1011 |

Selection Sort

|  |  |  |
| --- | --- | --- |
| Selection Sort |  |  |
| S.No. | Input Size | Time Time (In nanoSeconds) |
| 1 | 5000 | 6293482 |
| 2 | 1000 | 1735465 |
| 3 | 9 | 2963 |

Quick Sort:

|  |  |  |
| --- | --- | --- |
| Quick Sort |  |  |
| S.No. | Input Size | Time (In nanoSeconds) |
| 1 | 5000 | 622082 |
| 2 | 1000 | 164808 |
| 3 | 9 | 3157 |

Merge Sort:

|  |  |  |
| --- | --- | --- |
| Merge Sort |  |  |
| S.No. | Input Size | Time (In nanoSeconds) |
| 1 | 5000 | 943205 |
| 2 | 1000 | 294569 |
| 3 | 9 | 7522 |

Heap Sort:

|  |  |  |  |
| --- | --- | --- | --- |
| Heap Sort |  |  |  |
| S.No. | Input Size | Time (In nanoSeconds) |  |
| 1 | 5000 | 1519287 |  |
| 2 | 1000 | 554279 |  |
| 3 | 9 | 9372 |  |
|  |  |  |  |

Combine Sort:

|  |  |  |
| --- | --- | --- |
| Combine Sort |  |  |
| S.No. | Input Size | Time(In nanoSeconds) |
| 1 | 5000 | 19674148 |
| 2 | 1000 | 1639561 |
| 3 | 9 | 2257 |

Formula to calculate constants :

Log(T(N)) = b Log(N) + c

Where T(N) – Time taken by the algorithm to execute

N – The number of inputs

Value of b and c for Insertion Sort:

b = 2.04

c = - 6.856

Value of b and c for Selection Sort:

b = 2.04

c = - 6.856

Value of b and c for Quick Sort:

b = 0.79

c = - 2.90

Value of b and c for Heap Sort:

b = 1.955

c = - 8.106

Value of b and c for Merge Sort:

b = 1.33

c = - 5.24

Value of b and c for Combine Sort:

b = 0.21

c = - 0.143

Combine Sort : It is a hybrid sort of Insertion sort and Quick sort. As, we know that insertion sort is good for arrays which are in small in size and quick sort is good for arrays large in size. That’s why I have combined both insertion sort and quick sort for the better results.

In the Combine sort, it implements Sort.java . It has sort method which returns array in which I am calling quick sort method and within the quick sort there is an if condition i.e. if the array size is less than 9 insertion sort will take place, else quick sort will take place.

Combine Sort Switching logic:-

**public** **void** quickSort(**int** arr[], **int** low, **int** high)

{

**if** (low < high)

{

// Insertion Sort Starts!

**if** ((high - low) < 9)

{

**this**.InsertionSort(arr, low, high + 1);

}

// Merge Sort Starts!

**else**

{

**int** pi = **this**.partition(arr, low, high);

**this**.quickSort(arr, low, pi - 1);

**this**.quickSort(arr, pi + 1, high);

}

}

}