# ITCS 6114/8114 – Algorithm and Data Structures

## **Project Part - I**

*- Submitted by: Sampath Sree Kumar (800887568)*

Contents

[Comparison of Sorting Algorithms for unsorted array 3](#_Toc421295303)

[1. Heap Sort: 3](#_Toc421295304)

[2. Merge Sort: 3](#_Toc421295305)

[3. In – Place Quick Sort: 4](#_Toc421295306)

[4. In – Place Quick Sort – Median of Three and Insertion sort: 5](#_Toc421295307)

[Comparison of sorting algorithms for already Sorted Array 7](#_Toc421295308)

[Heap Sort – Ascending order 7](#_Toc421295309)

[Heap Sort – Descending order 7](#_Toc421295310)

[Merge Sort –Ascending order 8](#_Toc421295311)

[Merge Sort –Descending order 8](#_Toc421295312)

[Quick Sort – Ascending order 9](#_Toc421295313)

[Quick Sort – Descending order 9](#_Toc421295314)

[Quick Sort – Median of three & Insertion sort –Ascending order 10](#_Toc421295315)

[Quick Sort – Median of three & Insertion sort – Descending order 10](#_Toc421295316)

[Comparison of algorithms for sorted array (Ascending order) 11](#_Toc421295317)

[Comparison of algorithms for sorted array (Descending order) 11](#_Toc421295318)

[Comparisons of algorithms without quick sort for ascending order 12](#_Toc421295319)

[Observation: 12](#_Toc421295320)

[Simulation: 14](#_Toc421295321)

[Result: 15](#_Toc421295322)

# Comparison of Sorting Algorithms for unsorted array

## Heap Sort:

Data structures chosen: Array of N elements which are randomly generated using random function.

Methods involved:

* Max\_heapify (A, i, heap\_size)
* Build\_max\_heap (A, heap\_size)
* Heap\_sort (A, heap\_size)

Complexity Analysis:

* 1. Max Heapify: This method runs in O (logn) time. It is used to maintain max heap property
  2. Build max heap: This method runs in linear time and is used to produce max heap from unordered input array
  3. Heap sort: This method runs in O (nlogn) time. It is used to sort an array in place.

Complexity in worst, best and average cases:

|  |  |  |
| --- | --- | --- |
| Best Case | Average Case | Worst Case |
| O (n log(n)) | O (n log(n)) | O (n log(n)) |

Result: Graph for Input size vs Time for unsorted array of numbers

Figure 1: Heap sort for Unsorted Input

## Merge Sort:

Data structures chosen: Array of N elements which are randomly generated using random function.

Methods involved:

* Merge (left, right)
* Merge\_sort (A)

Complexity Analysis:

1. Merge sort: This method runs in O (nlogn) time. It is used to sort an array in place.

Complexity in worst, best and average cases:

|  |  |  |
| --- | --- | --- |
| Best Case | Average Case | Worst Case |
| O (n log(n)) | O (n log(n)) | O (n log(n)) |

Result: Graph for Input size vs Time for unsorted array of numbers

Figure 2: Merge Sort for Unsorted Input

## In – Place Quick Sort:

Data structures chosen: Array of N elements which are randomly generated using random function.

Methods involved:

* partition (A, start, end)
* quicksortendpivot (A, p, r)
* quicksortstack (A, p, r)

Complexity Analysis:

1. Partition: It is used to partition the array based on end element as pivot
2. quicksortendpivot: This method runs in O(nlogn) and is used to sort unordered input array
3. Quicksortstack: This method is used when we sort sorted array to overcome recursion depth exceed problem

Complexity in worst, best and average cases:

|  |  |  |
| --- | --- | --- |
| Best Case | Average Case | Worst Case |
| O (n log(n)) | O (n log(n)) | O (n ^ 2) |

Result: Graph for Input size vs Time for unsorted array of numbers

Figure 3: Quick Sort for Unsorted Input

## In – Place Quick Sort – Median of Three and Insertion sort:

Data structures chosen: Array of N elements which are randomly generated using random function.

Methods involved:

* partition (A, start, end)
* quicksort (A, p, r)
* medianofthree(A, left, right)

Complexity Analysis:

1. Partition: It is used to partition the array based on end element as pivot
2. quicksortendpivot: This method runs in O(nlogn) and is used to sort unordered input array
3. medianofthree: This method is used to return position of pivot which is selected as median of three numbers.

Complexity in worst, best and average cases for quick sort:

|  |  |  |
| --- | --- | --- |
| Best Case | Average Case | Worst Case |
| O (n log(n)) | O (n log(n)) | O (n ^ 2) |

Result: Graph for Input size vs Time for unsorted array of numbers

Figure 4: Quick Sort - MOT - Insertionsort - Unsorted Input

Comparison of All the above sorting algorithms using Average execution times for given inputs.

Figure 5: Comparison for Unsorted Inputs

# Comparison of sorting algorithms for already Sorted Array

## Heap Sort – Ascending order

Figure 6: Heap Sort - Sorted Input - ASC

## Heap Sort – Descending order

Figure 7: Heap Sort - Sorted Input - Desc

## Merge Sort –Ascending order

Figure 8: Merge Sort - Sorted Input - ASC

## Merge Sort –Descending order

Figure 9: Merge Sort - Sorted Input - Desc

## Quick Sort – Ascending order

Figure 10: Quick Sort - Sorted Input - Asc

## Quick Sort – Descending order

Figure 11: Quick Sort - Sorted Input - Desc

## Quick Sort – Median of three & Insertion sort –Ascending order

Figure 12: Quick Sort - MOT - Insertionsort - Sorted Input - Asc

## Quick Sort – Median of three & Insertion sort – Descending order

Figure 13: Quick Sort - MOT - Insertionsort - Sorted Input - Desc

## 

## Comparison of algorithms for sorted array (Ascending order)

Figure 14: Comparison - Sorted Input - Asc

## Comparison of algorithms for sorted array (Descending order)

Figure 15: Comparison - Sorted Input -Desc

## Comparisons of algorithms without quick sort for ascending order

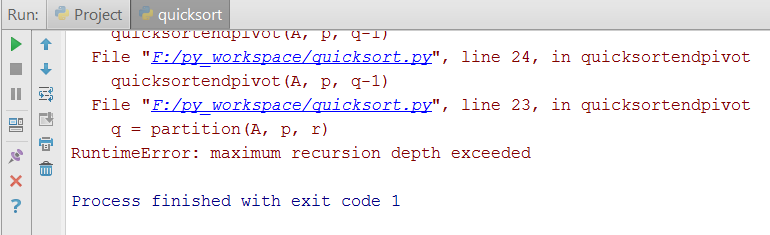
Figure 16: Comparison - Sorted Input - Without Quick sort

# Observation:

1. In all cases, Heap sort is taking more time for execution than merge sort even though both algorithms have same time complexity because for every number we remove from Heap we need to Heapify the tree again. Hence it is taking more time than merge sort.
2. Quick sort with insertion sort is much faster than randomized quick sort (where end element is pivot). Quick sort with insertion sort is implemented along with median of three as pivot. Hence it executed faster than quick sort and results in best case (O (n logn)) for almost every execution.
3. For sorted array and pivot as end element, quick sort runs in worst case scenario which is O (n ^ 2). If we observe figure 14, 15 Quick sort is taking more time compared to other sorting algorithms.

Limitation in Python:

* Recurrence is not recommended for recurring more than 999 times. In order to overcome this we need to increase recursion limit implicitly.
* If recursion limit is not set implicitly, below error is thrown when we execute recurrence methods like quick sort more than 999 times.



* To overcome the above error, I have implemented using iterative approach for quick sort for sorted input. Following is code snippet.

**def** quicksortstack(A, p, r):  
 temp\_stack = []  
 temp\_stack.append((p,r))  
  
 **while** temp\_stack:  
 pos = temp\_stack.pop()  
 right, left = pos[1], pos[0]  
 piv = partition(A,left,right)  
 **if** piv-1 > left:  
 temp\_stack.append((left,piv-1))  
 **if** piv+1 < right:  
 temp\_stack.append((piv+1,right))  
 **return** A

The above code is for only used for sorting sorted array for large N. I have also implemented normal quick sort for unsorted input.

In terms of (asymptotic) time complexity - they are both the same.

"Recursive is slower than iterative" - the rationale behind this statement is because of the overhead of the recursive stack.

Both recursive and iterative quicksort are O(nlogn) average case and O(n^2) worst case

# Steps to execute:

Following is the file containing following programs.



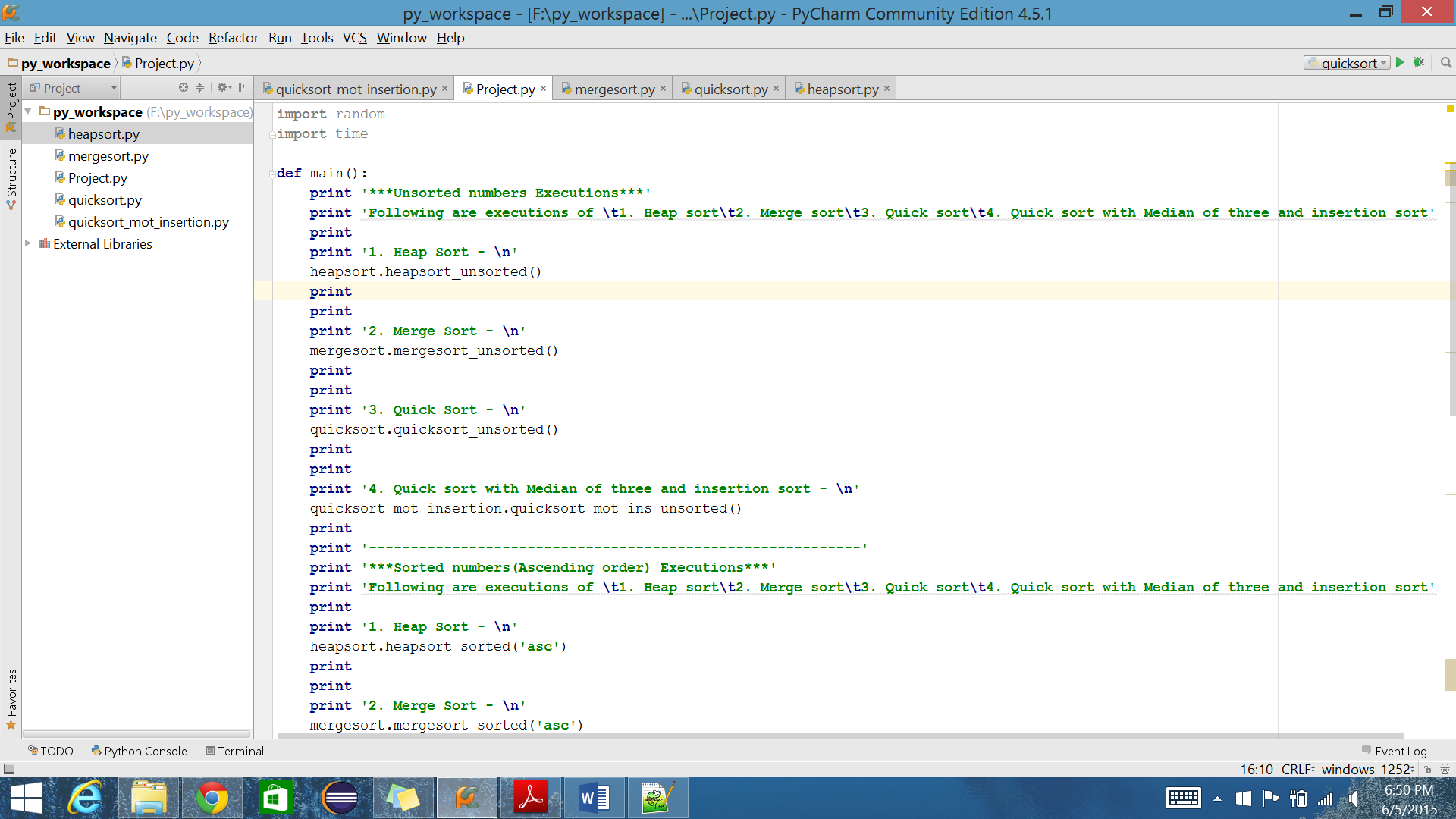
1. Project.py
2. Heapsort.py
3. Mergesort.py
4. Quicksort.py
5. Quicksort\_mot\_insertion.py

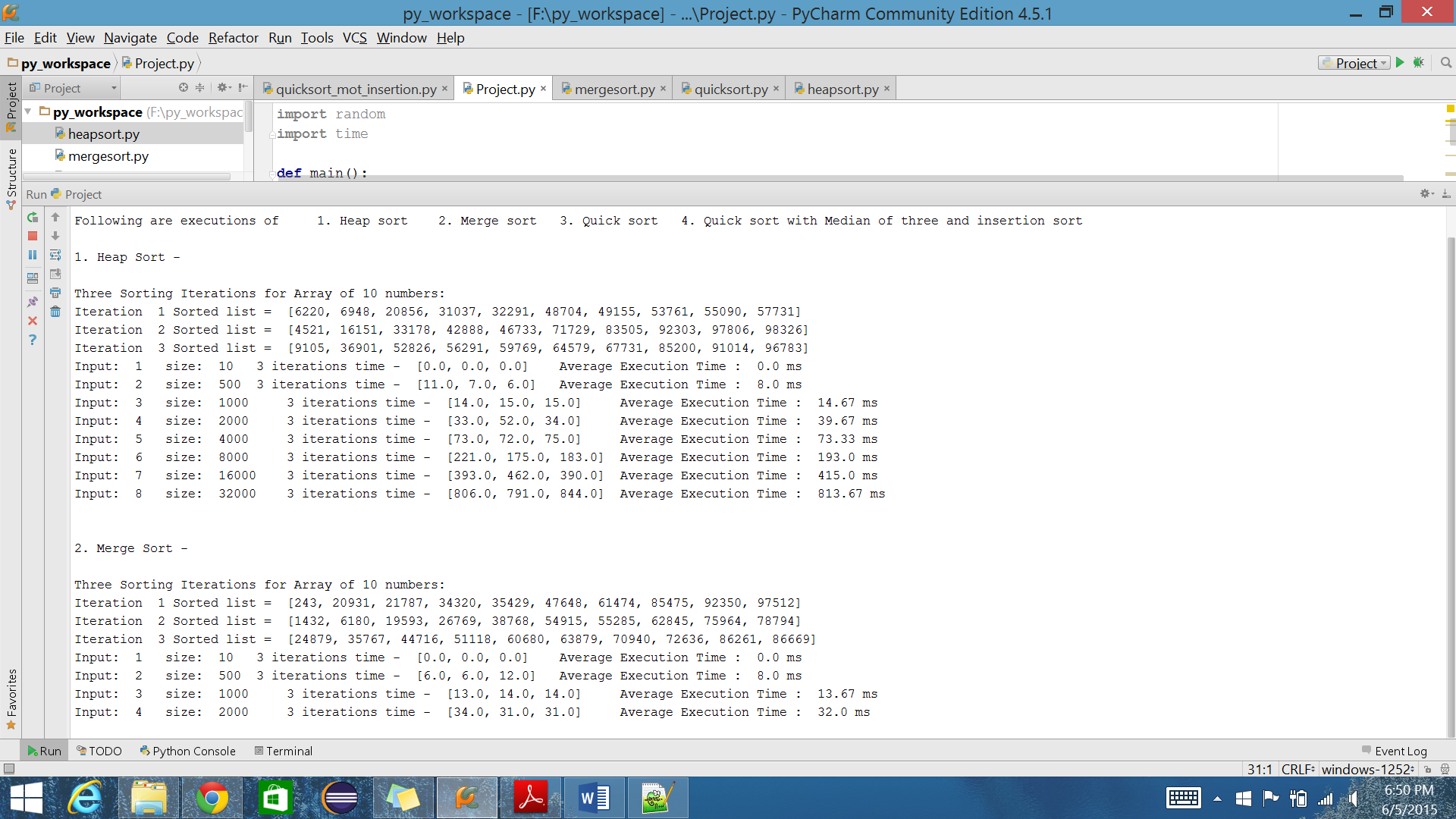
In cmd, run “python Project.py”. Following results are displayed in order below:

1. ‘Heap sort’, ‘merge sort’, ‘quick sort’ and ‘quicksort with median of three and insertion sort’ for unsorted input
2. ‘Heap sort’, ‘merge sort’, ‘quicksort with median of three and insertion sort’ and ‘quick sort’ for sorted input in ascending order
3. ‘Heap sort’, ‘merge sort’, ‘quicksort with median of three and insertion sort’ and ‘quick sort’ for sorted input in descending order

# Simulation:

Output:





## Result:

C:\Python27\python.exe F:/py\_workspace/Project.py

\*\*\*Unsorted numbers Executions\*\*\*

Following are executions of 1. Heap sort 2. Merge sort 3. Quick sort 4. Quick sort with Median of three and insertion sort

1. Heap Sort -

Three Sorting Iterations for Array of 10 numbers:

Iteration 1 Sorted list = [8319, 20355, 22167, 32528, 36432, 40096, 57305, 79849, 87508, 98539]

Iteration 2 Sorted list = [3835, 6432, 8211, 8657, 33039, 46090, 55642, 76937, 78577, 84938]

Iteration 3 Sorted list = [19427, 19474, 42191, 52826, 59228, 59865, 64141, 65774, 69053, 99373]

Input: 1 size: 10 3 iterations time - [1.0, 0.0, 0.0] Average Execution Time : 0.33 ms

Input: 2 size: 500 3 iterations time - [7.0, 6.0, 7.0] Average Execution Time : 6.67 ms

Input: 3 size: 1000 3 iterations time - [15.0, 15.0, 14.0] Average Execution Time : 14.67 ms

Input: 4 size: 2000 3 iterations time - [34.0, 38.0, 46.0] Average Execution Time : 39.33 ms

Input: 5 size: 4000 3 iterations time - [72.0, 73.0, 78.0] Average Execution Time : 74.33 ms

Input: 6 size: 8000 3 iterations time - [160.0, 171.0, 160.0] Average Execution Time : 163.67 ms

Input: 7 size: 16000 3 iterations time - [357.0, 353.0, 360.0] Average Execution Time : 356.67 ms

Input: 8 size: 32000 3 iterations time - [848.0, 782.0, 782.0] Average Execution Time : 804.0 ms

2. Merge Sort -

Three Sorting Iterations for Array of 10 numbers:

Iteration 1 Sorted list = [1855, 16044, 25148, 25151, 30711, 61945, 65389, 67320, 79983, 94408]

Iteration 2 Sorted list = [48185, 53856, 61039, 74906, 78984, 79955, 84961, 86637, 87324, 95970]

Iteration 3 Sorted list = [5240, 16895, 18484, 31357, 33137, 63163, 73873, 76747, 83741, 89658]

Input: 1 size: 10 3 iterations time - [0.0, 0.0, 0.0] Average Execution Time : 0.0 ms

Input: 2 size: 500 3 iterations time - [6.0, 6.0, 6.0] Average Execution Time : 6.0 ms

Input: 3 size: 1000 3 iterations time - [14.0, 14.0, 14.0] Average Execution Time : 14.0 ms

Input: 4 size: 2000 3 iterations time - [33.0, 34.0, 51.0] Average Execution Time : 39.33 ms

Input: 5 size: 4000 3 iterations time - [88.0, 65.0, 65.0] Average Execution Time : 72.67 ms

Input: 6 size: 8000 3 iterations time - [140.0, 150.0, 163.0] Average Execution Time : 151.0 ms

Input: 7 size: 16000 3 iterations time - [301.0, 301.0, 296.0] Average Execution Time : 299.33 ms

Input: 8 size: 32000 3 iterations time - [637.0, 638.0, 735.0] Average Execution Time : 670.0 ms

3. Quick Sort -

Three Sorting Iterations for Array of 10 numbers:

Iteration 1 Sorted list = [1663, 4193, 12911, 19938, 21615, 38713, 69003, 69097, 70064, 71132]

Iteration 2 Sorted list = [7550, 34643, 42015, 43671, 47763, 54563, 72787, 91365, 94617, 96183]

Iteration 3 Sorted list = [10091, 28391, 39082, 46196, 46738, 59569, 71167, 74517, 92413, 96989]

Input: 1 size: 10 3 iterations time - [1.0, 0.0, 0.0] Average Execution Time : 0.33 ms

Input: 2 size: 500 3 iterations time - [4.0, 5.0, 4.0] Average Execution Time : 4.33 ms

Input: 3 size: 1000 3 iterations time - [10.0, 10.0, 12.0] Average Execution Time : 10.67 ms

Input: 4 size: 2000 3 iterations time - [27.0, 30.0, 20.0] Average Execution Time : 25.67 ms

Input: 5 size: 4000 3 iterations time - [49.0, 38.0, 45.0] Average Execution Time : 44.0 ms

Input: 6 size: 8000 3 iterations time - [104.0, 101.0, 119.0] Average Execution Time : 108.0 ms

Input: 7 size: 16000 3 iterations time - [203.0, 211.0, 207.0] Average Execution Time : 207.0 ms

Input: 8 size: 32000 3 iterations time - [476.0, 500.0, 436.0] Average Execution Time : 470.67 ms

4. Quick sort with Median of three and insertion sort -

Three Sorting Iterations for Array of 10 numbers:

Iteration 1 Sorted list = [3759, 8719, 12527, 14567, 22820, 36916, 45536, 47133, 87420, 93789]

Iteration 2 Sorted list = [22143, 25156, 33573, 53576, 53997, 66627, 67252, 69739, 97231, 98985]

Iteration 3 Sorted list = [22822, 23914, 34115, 40183, 41493, 64065, 69060, 78305, 87652, 93349]

Input: 1 size: 10 3 iterations time - [0.0, 1.0, 0.0] Average Execution Time : 0.33 ms

Input: 2 size: 500 3 iterations time - [5.0, 3.0, 4.0] Average Execution Time : 4.0 ms

Input: 3 size: 1000 3 iterations time - [6.0, 11.0, 8.0] Average Execution Time : 8.33 ms

Input: 4 size: 2000 3 iterations time - [24.0, 30.0, 19.0] Average Execution Time : 24.33 ms

Input: 5 size: 4000 3 iterations time - [50.0, 39.0, 37.0] Average Execution Time : 42.0 ms

Input: 6 size: 8000 3 iterations time - [87.0, 86.0, 83.0] Average Execution Time : 85.33 ms

Input: 7 size: 16000 3 iterations time - [178.0, 156.0, 149.0] Average Execution Time : 161.0 ms

Input: 8 size: 32000 3 iterations time - [342.0, 333.0, 330.0] Average Execution Time : 335.0 ms

-----------------------------------------------------------

\*\*\*Sorted numbers(Ascending order) Executions\*\*\*

Following are executions of 1. Heap sort 2. Merge sort 3. Quick sort 4. Quick sort with Median of three and insertion sort

1. Heap Sort -

Three Sorting Iterations for Array of 10 numbers:

Iteration 1 Sorted list = [6144, 14160, 24007, 24412, 42370, 59284, 66247, 68083, 85416, 79646]

Iteration 2 Sorted list = [15706, 17906, 22638, 22985, 41173, 49994, 72699, 73532, 94521, 79518]

Iteration 3 Sorted list = [772, 13338, 13576, 14879, 19860, 27166, 54865, 70156, 83666, 72050]

Input: 1 size: 10 3 iterations time - [0.0, 0.0, 0.0] Average Execution Time : 0.0 ms

Input: 2 size: 500 3 iterations time - [7.0, 7.0, 7.0] Average Execution Time : 7.0 ms

Input: 3 size: 1000 3 iterations time - [16.0, 15.0, 17.0] Average Execution Time : 16.0 ms

Input: 4 size: 2000 3 iterations time - [48.0, 40.0, 35.0] Average Execution Time : 41.0 ms

Input: 5 size: 4000 3 iterations time - [80.0, 85.0, 77.0] Average Execution Time : 80.67 ms

Input: 6 size: 8000 3 iterations time - [176.0, 173.0, 174.0] Average Execution Time : 174.33 ms

Input: 7 size: 16000 3 iterations time - [379.0, 383.0, 372.0] Average Execution Time : 378.0 ms

Input: 8 size: 32000 3 iterations time - [808.0, 799.0, 791.0] Average Execution Time : 799.33 ms

2. Merge Sort -

Three Sorting Iterations for Array of 10 numbers:

Iteration 1 Sorted list = [3942, 6717, 9467, 20492, 20579, 38933, 75963, 77512, 89973, 91622]

Iteration 2 Sorted list = [7544, 10595, 14525, 30865, 36377, 44451, 45297, 64389, 68565, 97566]

Iteration 3 Sorted list = [11957, 18394, 58807, 60810, 67728, 77780, 83093, 86953, 95711, 96009]

Input: 1 size: 10 3 iterations time - [0.0, 0.0, 0.0] Average Execution Time : 0.0 ms

Input: 2 size: 500 3 iterations time - [5.0, 5.0, 5.0] Average Execution Time : 5.0 ms

Input: 3 size: 1000 3 iterations time - [10.0, 10.0, 11.0] Average Execution Time : 10.33 ms

Input: 4 size: 2000 3 iterations time - [25.0, 34.0, 27.0] Average Execution Time : 28.67 ms

Input: 5 size: 4000 3 iterations time - [46.0, 45.0, 52.0] Average Execution Time : 47.67 ms

Input: 6 size: 8000 3 iterations time - [95.0, 101.0, 97.0] Average Execution Time : 97.67 ms

Input: 7 size: 16000 3 iterations time - [205.0, 205.0, 199.0] Average Execution Time : 203.0 ms

Input: 8 size: 32000 3 iterations time - [426.0, 428.0, 421.0] Average Execution Time : 425.0 ms

3. Quick sort with Median of three and insertion sort -

Three Sorting Iterations for Array of 10 numbers:

Iteration 1 Sorted list = [14527, 18926, 19189, 38495, 48616, 51203, 70790, 79643, 94959, 95397]

Iteration 2 Sorted list = [608, 9541, 51762, 54762, 55634, 78512, 79804, 89937, 93659, 96251]

Iteration 3 Sorted list = [1292, 1816, 4711, 13699, 38195, 43497, 48878, 65514, 67197, 79386]

Input: 1 size: 10 3 iterations time - [0.0, 0.0, 0.0] Average Execution Time : 0.0 ms

Input: 2 size: 500 3 iterations time - [1.0, 3.0, 3.0] Average Execution Time : 2.33 ms

Input: 3 size: 1000 3 iterations time - [4.0, 4.0, 3.0] Average Execution Time : 3.67 ms

Input: 4 size: 2000 3 iterations time - [8.0, 13.0, 12.0] Average Execution Time : 11.0 ms

Input: 5 size: 4000 3 iterations time - [27.0, 21.0, 19.0] Average Execution Time : 22.33 ms

Input: 6 size: 8000 3 iterations time - [50.0, 45.0, 44.0] Average Execution Time : 46.33 ms

Input: 7 size: 16000 3 iterations time - [99.0, 103.0, 102.0] Average Execution Time : 101.33 ms

Input: 8 size: 32000 3 iterations time - [231.0, 224.0, 226.0] Average Execution Time : 227.0 ms

4. Quick Sort -

Three Sorting Iterations for Array of 10 numbers:

Iteration 1 Sorted list = [6874, 36011, 36809, 39575, 59679, 70252, 71415, 88070, 93177, 93267]

Iteration 2 Sorted list = [13132, 15550, 36078, 37596, 42551, 44998, 53166, 89945, 96397, 97314]

Iteration 3 Sorted list = [2429, 12671, 44619, 49529, 57911, 59372, 61076, 73334, 80776, 86334]

Input: 1 size: 10 3 iterations time - [0.0, 0.0, 0.0] Average Execution Time : 0.0 ms

Input: 2 size: 500 3 iterations time - [79.0, 89.0, 124.0] Average Execution Time : 97.33 ms

Input: 3 size: 1000 3 iterations time - [333.0, 323.0, 325.0] Average Execution Time : 327.0 ms

Input: 4 size: 2000 3 iterations time - [1335.0, 1320.0, 1333.0] Average Execution Time : 1329.33 ms

Input: 5 size: 4000 3 iterations time - [5448.0, 5477.0, 5441.0] Average Execution Time : 5455.33 ms

Input: 6 size: 8000 3 iterations time - [22445.0, 22384.0, 23710.0] Average Execution Time : 22846.33 ms

Input: 7 size: 16000 3 iterations time - [96923.0, 96140.0, 96376.0] Average Execution Time : 96479.67 ms

Input: 8 size: 32000 3 iterations time - [403704.0, 2847646.0, 226344.0] Average Execution Time : 1159231.33 ms

-----------------------------------------------------------

\*\*\*Sorted numbers(Descending order) Executions\*\*\*

Following are executions of 1. Heap sort 2. Merge sort 3. Quick sort 4. Quick sort with Median of three and insertion sort

1. Heap Sort -

Three Sorting Iterations for Array of 10 numbers:

Iteration 1 Sorted list = [402, 1718, 4672, 18549, 43365, 51096, 71381, 76511, 77485, 95034]

Iteration 2 Sorted list = [18354, 26514, 29882, 30633, 32434, 47505, 51382, 70891, 74336, 82663]

Iteration 3 Sorted list = [11748, 24070, 25637, 38422, 42567, 72922, 81692, 85390, 90831, 92843]

Input: 1 size: 10 3 iterations time - [0.0, 0.0, 0.0] Average Execution Time : 0.0 ms

Input: 2 size: 500 3 iterations time - [3.0, 4.0, 3.0] Average Execution Time : 3.33 ms

Input: 3 size: 1000 3 iterations time - [7.0, 7.0, 8.0] Average Execution Time : 7.33 ms

Input: 4 size: 2000 3 iterations time - [16.0, 16.0, 16.0] Average Execution Time : 16.0 ms

Input: 5 size: 4000 3 iterations time - [35.0, 35.0, 36.0] Average Execution Time : 35.33 ms

Input: 6 size: 8000 3 iterations time - [58.0, 47.0, 45.0] Average Execution Time : 50.0 ms

Input: 7 size: 16000 3 iterations time - [356.0, 329.0, 328.0] Average Execution Time : 337.67 ms

Input: 8 size: 32000 3 iterations time - [719.0, 735.0, 730.0] Average Execution Time : 728.0 ms

2. Merge Sort -

Three Sorting Iterations for Array of 10 numbers:

Iteration 1 Sorted list = [8834, 22899, 28220, 38026, 51197, 52521, 55957, 68081, 93781, 94466]

Iteration 2 Sorted list = [14522, 31914, 33888, 35731, 41057, 47778, 59226, 69855, 75241, 92743]

Iteration 3 Sorted list = [1883, 10801, 35062, 43603, 43866, 50821, 54070, 73206, 86583, 90059]

Input: 1 size: 10 3 iterations time - [0.0, 0.0, 0.0] Average Execution Time : 0.0 ms

Input: 2 size: 500 3 iterations time - [5.0, 4.0, 4.0] Average Execution Time : 4.33 ms

Input: 3 size: 1000 3 iterations time - [10.0, 11.0, 10.0] Average Execution Time : 10.33 ms

Input: 4 size: 2000 3 iterations time - [23.0, 22.0, 21.0] Average Execution Time : 22.0 ms

Input: 5 size: 4000 3 iterations time - [45.0, 45.0, 49.0] Average Execution Time : 46.33 ms

Input: 6 size: 8000 3 iterations time - [93.0, 95.0, 97.0] Average Execution Time : 95.0 ms

Input: 7 size: 16000 3 iterations time - [201.0, 201.0, 199.0] Average Execution Time : 200.33 ms

Input: 8 size: 32000 3 iterations time - [413.0, 416.0, 413.0] Average Execution Time : 414.0 ms

3. Quick sort with Median of three and insertion sort -

Three Sorting Iterations for Array of 10 numbers:

Iteration 1 Sorted list = [990, 5142, 5938, 14241, 28421, 41973, 59780, 66213, 82725, 85654]

Iteration 2 Sorted list = [9268, 33844, 40383, 59210, 68007, 68257, 70806, 72614, 96415, 98844]

Iteration 3 Sorted list = [4985, 11244, 11303, 24027, 25448, 52783, 72740, 77642, 93193, 96394]

Input: 1 size: 10 3 iterations time - [0.0, 0.0, 0.0] Average Execution Time : 0.0 ms

Input: 2 size: 500 3 iterations time - [3.0, 3.0, 3.0] Average Execution Time : 3.0 ms

Input: 3 size: 1000 3 iterations time - [7.0, 7.0, 7.0] Average Execution Time : 7.0 ms

Input: 4 size: 2000 3 iterations time - [16.0, 15.0, 15.0] Average Execution Time : 15.33 ms

Input: 5 size: 4000 3 iterations time - [35.0, 35.0, 35.0] Average Execution Time : 35.0 ms

Input: 6 size: 8000 3 iterations time - [78.0, 78.0, 77.0] Average Execution Time : 77.67 ms

Input: 7 size: 16000 3 iterations time - [174.0, 171.0, 169.0] Average Execution Time : 171.33 ms

Input: 8 size: 32000 3 iterations time - [376.0, 344.0, 369.0] Average Execution Time : 363.0 ms

4. Quick Sort -

Three Sorting Iterations for Array of 10 numbers:

Iteration 1 Sorted list = [16511, 29366, 37575, 53123, 61593, 75375, 90136, 91929, 94447, 97475]

Iteration 2 Sorted list = [2699, 21677, 23811, 23832, 25362, 35843, 70979, 84320, 94158, 96323]

Iteration 3 Sorted list = [8590, 16206, 25296, 25975, 28750, 53577, 60256, 74182, 84059, 94829]

Input: 1 size: 10 3 iterations time - [1.0, 0.0, 0.0] Average Execution Time : 0.33 ms

Input: 2 size: 500 3 iterations time - [51.0, 50.0, 50.0] Average Execution Time : 50.33 ms

Input: 3 size: 1000 3 iterations time - [205.0, 207.0, 206.0] Average Execution Time : 206.0 ms

Input: 4 size: 2000 3 iterations time - [865.0, 861.0, 853.0] Average Execution Time : 859.67 ms

Input: 5 size: 4000 3 iterations time - [3602.0, 3319.0, 3546.0] Average Execution Time : 3489.0 ms

Input: 6 size: 8000 3 iterations time - [14271.0, 13970.0, 14364.0] Average Execution Time : 14201.67 ms

Input: 7 size: 16000 3 iterations time - [57213.0, 56826.0, 60575.0] Average Execution Time : 58204.67 ms

Input: 8 size: 32000 3 iterations time - [1157374.0, 176538.0, 400304.0] Average Execution Time : 578072.0 ms

Thank You!!!

Process finished with exit code 0