

CSE 5311 Design and Analysis of Algorithms

Project - Implementing and Comparing the Different Algorithms

Team Members:

Vineet Venkata Kalluri – 1002063858

Sai Rohith Tulasi – 1001533204

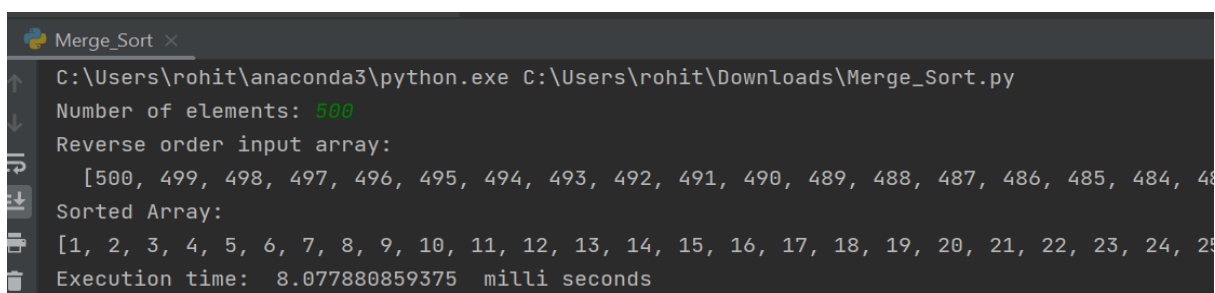
Overall Status

In this project we have implemented 'Merge Sort', 'Quick Sort', 'Insertions Sort', 'Heap Sort' and 'Radix Sort' algorithms and compared their running times based on the small (500), medium (10000) and large (100000) input lengths. To get better understanding, we have categorized the inputs into 3 categories, 1) Best case (when the input data is in sorted order), 2) Worst case (when the input data is reverse sorted order), 3) Random case (when the input data is in random order).

Merge Sort:

The Merge Sort algorithm is a sorting algorithm that is based on the Divide and Conquer paradigm. In this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner.

- For Small Input Length (500)
 - When the input data is in reverse sorted order (Worst Case):



```
Merge_Sort x
C:\Users\rohit\anaconda3\python.exe C:\Users\rohit\Downloads\Merge_Sort.py
Number of elements: 500
Reverse order input array:
[500, 499, 498, 497, 496, 495, 494, 493, 492, 491, 490, 489, 488, 487, 486, 485, 484, 483, 482, 481, 480, 479, 478, 477, 476, 475, 474, 473, 472, 471, 470, 469, 468, 467, 466, 465, 464, 463, 462, 461, 460, 459, 458, 457, 456, 455, 454, 453, 452, 451, 450, 449, 448, 447, 446, 445, 444, 443, 442, 441, 440, 439, 438, 437, 436, 435, 434, 433, 432, 431, 430, 429, 428, 427, 426, 425, 424, 423, 422, 421, 420, 419, 418, 417, 416, 415, 414, 413, 412, 411, 410, 409, 408, 407, 406, 405, 404, 403, 402, 401, 400, 399, 398, 397, 396, 395, 394, 393, 392, 391, 390, 389, 388, 387, 386, 385, 384, 383, 382, 381, 380, 379, 378, 377, 376, 375, 374, 373, 372, 371, 370, 369, 368, 367, 366, 365, 364, 363, 362, 361, 360, 359, 358, 357, 356, 355, 354, 353, 352, 351, 350, 349, 348, 347, 346, 345, 344, 343, 342, 341, 340, 339, 338, 337, 336, 335, 334, 333, 332, 331, 330, 329, 328, 327, 326, 325, 324, 323, 322, 321, 320, 319, 318, 317, 316, 315, 314, 313, 312, 311, 310, 309, 308, 307, 306, 305, 304, 303, 302, 301, 300, 299, 298, 297, 296, 295, 294, 293, 292, 291, 290, 289, 288, 287, 286, 285, 284, 283, 282, 281, 280, 279, 278, 277, 276, 275, 274, 273, 272, 271, 270, 269, 268, 267, 266, 265, 264, 263, 262, 261, 260, 259, 258, 257, 256, 255, 254, 253, 252, 251, 250, 249, 248, 247, 246, 245, 244, 243, 242, 241, 240, 239, 238, 237, 236, 235, 234, 233, 232, 231, 230, 229, 228, 227, 226, 225, 224, 223, 222, 221, 220, 219, 218, 217, 216, 215, 214, 213, 212, 211, 210, 209, 208, 207, 206, 205, 204, 203, 202, 201, 200, 199, 198, 197, 196, 195, 194, 193, 192, 191, 190, 189, 188, 187, 186, 185, 184, 183, 182, 181, 180, 179, 178, 177, 176, 175, 174, 173, 172, 171, 170, 169, 168, 167, 166, 165, 164, 163, 162, 161, 160, 159, 158, 157, 156, 155, 154, 153, 152, 151, 150, 149, 148, 147, 146, 145, 144, 143, 142, 141, 140, 139, 138, 137, 136, 135, 134, 133, 132, 131, 130, 129, 128, 127, 126, 125, 124, 123, 122, 121, 120, 119, 118, 117, 116, 115, 114, 113, 112, 111, 110, 109, 108, 107, 106, 105, 104, 103, 102, 101, 100, 99, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84, 83, 82, 81, 80, 79, 78, 77, 76, 75, 74, 73, 72, 71, 70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 60, 59, 58, 57, 56, 55, 54, 53, 52, 51, 50, 49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Sorted Array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 500]
Execution time: 8.077880859375 milli seconds
```

- When the input data is in sorted order (Best case):

```
Sorted order input array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
Sorted Array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
Execution time: 7.09765625 milli seconds
```

- When the input data is in random order (Average Case):

```
Merge_Sort x
Random input array:
[89257 72174 95817 44567 31020 84435 33868 80233 44296 53923 33202 43596
35542 70704 11072 32032 51931 72085 41085 85621 51880 21419 95084 60967
8171 15906 20027 86909 16640 56644 91439 40391 93396 55418 16416 19695
12421 51931 19776 30258 14120 30720 60335 39561 94643 65006 30442 78217
27384 91538 99223 31227 75638 36426 26298 22378 33135 71421 53723 18879
82026 82117 82217 82530 83146 83452 83826 84164 84435 84506 84681 84945
85621 85706 85724 85834 85988 86262 86397 86526 86740 86909 86915 87390
87445 87495 87516 87539 87721 87796 88152 88823 89076 89257 89359 89372
89757 89825 90044 90132 90213 90720 90943 91007 91052 91439 91538 91697
91830 92224 92443 92460 92660 92847 92903 93267 93396 94266 94559 94643
95084 95381 95587 95817 96182 96355 96970 97011 97122 97684 97926 98378
98452 98935 99223 99297 99539 99548 99585 99778]
Execution time: 5.005615234375 milli seconds
```

➤ For Medium Input Length (10000)

- When the input data is in reverse sorted order (Worst Case):

```
Merge_Sort x
C:\Users\rohit\anaconda3\python.exe C:\Users\rohit\Download
Number of elements: 10000
Reverse order input array:
[10000, 9999, 9998, 9997, 9996, 9995, 9994, 9993, 9992, 9
Sorted Array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
Execution time: 101.484375 milli seconds
```

- When the input data is in sorted order (Best case):

```

Merge_Sort x
11, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22
Execution time: 101.484375 milli seconds
Sorted order input array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22]
Sorted Array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22]
Execution time: 101.365478515625 milli seconds

```

- When the input data is in random order (Average Case):

```

Random input array:
[89257 72174 95817 ... 97578 96670 8661]
Sorted Array:
[ 13 39 57 ... 99963 99968 99994]
Execution time: 91.507080078125 milli seconds

```

➤ For Large Input Length (100000)

```

Insertion_Sort Merge_Sort x
C:\Users\rohit\anaconda3\python.exe C:\Users\rohit\Downloads\Merge_Sort.py
Number of elements: 100000
Sorted Array:
Execution time of worst case: 1090.680419921875 milli seconds
Sorted Array:
Execution time of best case: 1085.745849609375 milli seconds
Sorted Array:
Execution time of random: 1019.4033203125 milli seconds

```

Average Time of Execution

- For small input (500), the average time of execution is 1.50 milli seconds.
- For medium input (10000), the average time of execution is 45.16 milli seconds.
- For large input (100000), the average time of execution is 573.31 milli seconds.

Quick Sort:

Quicksort is a Divide and Conquer algorithm. It picks an element as a pivot and partitions the given array around the picked pivot. There are many different versions of quicksort that pick pivot in different ways.

- For Small Input Length (100)

```
Quick_sort x
C:\Users\rohit\anaconda3\python.exe C:\Users\rohit\Downloads\Quick_sort.py
Number of elements: 100
Sorted Array:
Execution time of worst case: 1.000732421875 milli seconds
Sorted Array:
Execution time of best case: 0.0 milli seconds
Sorted Array:
Execution time of random: 0.0 milli seconds

Process finished with exit code 0
```

- For Medium Input Length (500)

```
Quick_sort x
C:\Users\rohit\anaconda3\python.exe C:\Users\rohit\Downloads\Quick_sort.py
Number of elements: 500
Sorted Array:
Execution time of worst case: 7.155517578125 milli seconds
Sorted Array:
Execution time of best case: 9.6162109375 milli seconds
Sorted Array:
Execution time of random: 1.00341796875 milli seconds

Process finished with exit code 0
```

- For Large Input Length (998)

```
Quick_sort x
C:\Users\rohit\anaconda3\python.exe C:\Users\rohit\Downloads\Quick_sort.py
Number of elements: 998
Sorted Array:
Execution time of worst case: 30.607666015625 milli seconds
Sorted Array:
Execution time of best case: 37.487548828125 milli seconds
Sorted Array:
Execution time of random: 2.499267578125 milli seconds

Process finished with exit code 0
```

Average Time of Execution

- For small input (100), the average time of execution is 1.00 milli seconds.
- For medium input (500), the average time of execution is 5.92 milli seconds.
- For large input (998), the average time of execution is milli 23.53 seconds.

Insertion Sort:

Insertion sort is a simple sorting algorithm that works like the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

➤ For Small Input Length (500)

- When the input data is in reverse sorted order (Worst Case):

```
Insertion_Sort x
C:\Users\rohit\anaconda3\python.exe C:\Users\rohit\Downloads\Insertio
Number of elements: 500
Reverse order input array:
[500, 499, 498, 497, 496, 495, 494, 493, 492, 491, 490, 489, 488, 4
Sorted Array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2
Execution time: 14.9990234375 milli seconds
```

- When the input data is in sorted order (Best case):

```
Sorted order input array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
Sorted Array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1
Execution time: 0.0 milli seconds
```

- When the input data is in random order (Average Case):

```
Insertion_Sort x
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
Execution time: 0.0 milli seconds
Random input array:
[89257 72174 95817 44567 31020 84435 33868 80233 44296 53923 33202 43596
35542 70704 11072 32032 51931 72085 41085 85621 51880 21419 95084 60967
8171 15906 20027 86909 16640 56644 91439 40391 93396 55418 16416 19695
12421 51931 19776 30258 14120 30720 60335 39561 94643 65006 30442 78217
78378 63731 1870 48400 3037 33777 41727 73387 27323 30431 47374 67780
36569 35600 68948 42588 50202 90213 84681 63781 21345 27612 58245 33549
9240 80254 97926 59610 25435 20631 87796 35356]
Sorted Array:
[93, 171, 191, 719, 723, 1364, 1473, 1504, 1890, 2256, 2302, 2683, 2904, 325
Execution time: 5.08349609375 milli seconds
```

➤ For Medium Input Length (10000)

- When the input data is in reverse sorted order (Worst Case):

```
Insertion_Sort x
C:\Users\rohit\anaconda3\python.exe C:\Users\rohit\Downloads\In
Number of elements: 10000
Reverse order input array:
[10000, 9999, 9998, 9997, 9996, 9995, 9994, 9993, 9992, 9991,
Sorted Array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
Execution time: 4407.685791015625 milli seconds
```


- When the input data is in sorted order (Best case):

```
Sorted order input array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1
Sorted Array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
Execution time: 6.09033203125 milli seconds
```

- When the input data is in random order (Average Case):

```
Random input array:
[89257 72174 95817 ... 97578 96670 8661]
Sorted Array:
[13, 39, 57, 66, 81, 86, 93, 109, 133, 135, 138, 140, 166,
Execution time: 2397.494140625 milli seconds
```

- For Large Input Length (100000)



```
Insertion_Sort Heap_Sort X
C:\Users\rohit\anaconda3\python.exe C:\Users\rohit\Downloads\Insertion_Sort.py
Number of elements: 100000
Execution time of worst case: 386534.6433105469 milli seconds
Execution time of best case: 11.1318359375 milli seconds
Execution time of random: 225981.61840820312 milli seconds

Process finished with exit code 0
```

Average Time of Execution

- For small input (500), the average time of execution is 6.66 milli seconds.
- For medium input (10000), the average time of execution is 2270 milli seconds.
- For large input (100000), the average time of execution is 204175 milli seconds.

Heap Sort:

Heap Sort is a comparison-based sorting technique based on Binary Heap data structure. It is like the selection sort where we first find the minimum element and place the minimum element at the beginning.

➤ For Small Input Length (500)

- When the input data is in reverse sorted order (Worst Case):

```
Insertion_Sort x Heap_Sort x
C:\Users\rohit\anaconda3\python.exe C:\Users\rohit\Downloads\Heap_Sort.py
Number of elements: 500
Reverse order input array:
[500, 499, 498, 497, 496, 495, 494, 493, 492, 491, 490, 489, 488, 487, 486, 485, 484, 483, 482, 481, 480,
Sorted Array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
Execution time: 1.16357421875 milli seconds
```

- When the input data is in sorted order (Best case):

```
Sorted order input array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
Sorted Array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
Execution time: 1.1328125 milli seconds
```

- When the input data is in random order (Average Case):

```
Insertion_Sort x Heap_Sort x
Random input array:
[89257 72174 95817 44567 31020 84435 33868 80233 44296 53923 33202 43596
35542 70704 11072 32032 51931 72085 41085 85621 51880 21419 95084 60967
8171 15906 20027 86909 16640 56644 91439 40391 93396 55418 16416 19695
12421 51931 19776 30258 14120 30720 60335 39561 94643 65006 30442 78217
27384 91538 99223 31227 75638 36426 26298 22378 33135 71421 53723 18879
35924 65633 45857 87539 23433 22093 7307 43817 30755 38205 34442 88823
83021 83700 83724 83804 83788 80202 80377 80320 80740 80707 80713 87370
87445 87495 87516 87539 87721 87796 88152 88823 89076 89257 89359 89372
89757 89825 90044 90132 90213 90720 90943 91007 91052 91439 91538 91697
91830 92224 92443 92460 92660 92847 92903 93267 93396 94266 94559 94643
95084 95381 95587 95817 96182 96355 96970 97011 97122 97684 97926 98378
98452 98935 99223 99297 99539 99548 99585 99778]
Execution time: 2.221923828125 milli seconds
```

➤ For Medium Input Length (10000)

- When the input data is in reverse sorted order (Worst Case):

```
Insertion_Sort x Heap_Sort x
C:\Users\rohit\anaconda3\python.exe C:\Users\rohit\Downloads\Heap_So
Number of elements: 10000
Reverse order input array:
[10000, 9999, 9998, 9997, 9996, 9995, 9994, 9993, 9992, 9991, 9990
Sorted Array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
Execution time: 31.0 milli seconds
```

- When the input data is in sorted order (Best case):

```
Sorted order input array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
Sorted Array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
Execution time: 34.60205078125 milli seconds
```

- When the input data is in random order (Average Case):

```
Random input array:
[89257 72174 95817 ... 97578 96670 8661]
Sorted Array:
[ 13 39 57 ... 99963 99968 99994]
Execution time: 69.91162109375 milli seconds
```

➤ For Large Input Length (100000)

```
Insertion_Sort  Heap_Sort ×
C:\Users\rohit\anaconda3\python.exe C:\Users\rohit\Downloads\Heap_Sort.py
Number of elements: 100000
Sorted Array:
Execution time of worst case: 393.897216796875 milli seconds
Sorted Array:
Execution time of best case: 438.639404296875 milli seconds
Random input array:
[89257 72174 95817 ... 6868 3935 65957]
Sorted Array:
[ 2 3 4 ... 99998 99998 99999]
Execution time of random: 887.4111328125 milli seconds
```

Average Time of Execution

- For small input (500), the average time of execution is 1.50 milli seconds.
- For medium input (10000), the average time of execution is 45.16 milli seconds.
- For large input (100000), the average time of execution is 573.31 milli seconds.

Radix sort

Radix Sort is to do digit by digit sort starting from least significant digit to most significant digit. Radix sort uses counting sort as a subroutine to sort.

➤ For Small Input Length (500)

- When the input data is in reverse sorted order (Worst Case):

```
Insertion_Sort x radix x
C:\Users\rohit\anaconda3\python.exe C:\Users\rohit\daa_project\radix.py
Number of elements: 500
Reverse order input array:
[500, 499, 498, 497, 496, 495, 494, 493, 492, 491, 490, 489, 488, 487, 486, 485, 484,
Sorted Array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
Execution time: 0.0 milli seconds
Sorted order input array:
```

- When the input data is in sorted order (Best case):

```
Sorted order input array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
Sorted Array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
Execution time: 0.0 milli seconds
```

- When the input data is in random order (Average Case):

```
85621 85706 85724 85834 85988 86262 86397 86526 86740 86909 86915 87390
87445 87495 87516 87539 87721 87796 88152 88823 89076 89257 89359 89372
89757 89825 90044 90132 90213 90720 90943 91007 91052 91439 91538 91697
91830 92224 92443 92460 92660 92847 92903 93267 93396 94266 94559 94643
95084 95381 95587 95817 96182 96355 96970 97011 97122 97684 97926 98378
98452 98935 99223 99297 99539 99548 99585 99778]
Execution time: 1.99755859375 milli seconds
```

➤ For Medium Input Length (10000)

- When the input data is in reverse sorted order (Worst Case):

```
Insertion_Sort x radix x
Number of elements: 10000
Reverse order input array:
[10000, 9999, 9998, 9997, 9996, 9995, 9994, 9993, 9992, 9991, 9990,
Sorted Array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
Execution time: 11.108642578125 milli seconds
```

- When the input data is in sorted order (Best case):

```
Sorted order input array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
Sorted Array:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
Execution time: 11.418701171875 milli seconds
```

- When the input data is in random order (Average Case):

```
Random input array:
[89257 72174 95817 ... 97578 96670 8661]
Sorted Array:
[ 13 39 57 ... 99963 99968 99994]
Execution time: 35.0 milli seconds
```

- For Large Input Length (100000)

```
Insertion_Sort  radix x
C:\Users\rohit\anaconda3\python.exe C:\Users\rohit\daa_project\radix.py
Number of elements: 100000
Sorted Array:
Execution time of worst case: 164.222900390625 milli seconds
Sorted Array:
Execution time of best case: 154.023681640625 milli seconds
Sorted Array:
Execution time of random: 415.1025390625 milli seconds
```

Average Time of Execution

- For small input (500), the average time of execution is 0.66 milli seconds.
- For medium input (10000), the average time of execution is 19.17 milli seconds.
- For large input (100000), the average time of execution is 244.44 milli seconds.

Conclusion

- When the input data is small (500) and,
- is in reverse order then Radix Sort is better.
 - is in sorted order then Insertion and Radix sort is better.
 - is in random order then Quick Sort is better.

- When the input data is medium (10000) and,
 - is in reverse order then Radix Sort is better.
 - is in sorted order then Insertion Sort is better.
 - is in random order then Quick Sort is better.

- When the input data is large (100000) and,
 - is in reverse order then Radix Sort is better.
 - is in sorted order then Insertion Sort is better.
 - is in random order then Quick Sort is better.

Division of Labor

Team Member: Vineet Venkata Kalluri

Searched and explored the Quick Sort, Insertion Sort and Heap Sort algorithms. Also, calculated the time taken for each of the algorithms in different data sizes.

Team Member: Sai Rohith

Searched and explored the Radix Sort and Merge Sort algorithms. Also, calculated the time taken for each of the algorithms in different data sizes.