CS 202 - Advance Data Structure & Algorithm

Assignment-2 Report

- Abhijeet Sharma (B15102)

- **Insertion Sort :-**

  Best Case O(n)

  Worst Case $O(n^2)$

  Average Case $O(n^2)$

  ❑ **Pseudo Code**

```
1. for   j = 2 to n
2.               key = A[j]
3.               // Insert A[j] into the sorted sequence A[1 … j - 1]
4.               i = j - 1
5.               while i > 0 and A[i] > key
6.                    A[i+1] = A[i]
7.                    i = i - 1
8.               A[i + 1] = key
```

- **Rank Sort :-**

  Best Case $O(n^2)$

  Worst Case $O(n^2)$

  Average Case $O(n^2)$

❏ **Pseudo Code**

```
1.   for  j = 1 to n
2.              R[j] = 1
// Rank the n elements in A into R
3.   for  j = 2 to n
4.              for  i = 1 to j - 1
5.                   if  A[i] <= A[j]
6.                        R[j] = R[j] + 1
7.                   else
8.                        R[i] = R[i] + 1
// Move to correct place in U[1 . . n]
9.   for  j = 1 to n
10.             U[R[j]] = A[j]
// Move the sorted entries into A
11.   for  j = 1 to n
12.             A[j] = U[j]
```

- **Bubble Sort :-**

  Best Case O(n)

  Worst Case $O(n^2)$

  Average Case $O(n^2)$

❏ **Pseudo Code**

```
1. j = n
2. while  j ≥ 2
                // Bubble up the smallest element to its correct
position
3.              for  i = 1 to j - 1
```

```
4.                    if  A[i] > A[i + 1]
5.                          temp = A[i]
6.                          A[i] = A[i + 1]
7.                          A[i + 1] = temp
8.              j = j - 1
```

- ## Selection Sort :-

  Best Case $O(n^2)$

  Worst Case $O(n^2)$

  Average Case $O(n^2)$

❑ **Pseudo Code**

```
1.   sorted = false
2.   j = n
3.   while  j  > 1 and sorted = false
4.              pos = 1
5.              sorted = true
                // Find the position of the largest element
6.              for  i = 2 to j
7.                   if  A[pos] <= A[i]
8.                        pos = i
9.                   else
10.                       sorted = false
                     // Move A[j] to the position of largest element
by           //swapping
11.             temp = A[pos]
12.             A[pos] = A[j]
13.             A[j] = temp
14.              j = j - 1
```

- ## Merge Sort :-

Best Case O(nlog(n))
Worst Case O(nlog(n))
Average Case O(nlog(n))

❑ **Pseudo Code**

```
1.    n1 = q - p + 1
2.    n2 = r - q
3.    A1[n1+1] = ∞
4.    A2[n2+1] = ∞
5.    i = 1
6.    j = 1
7.   for k = p to r
8.                if A1[i] ≤ A2[j]
9.                      A[k] = A1[i]
10.                     i = i + 1
11.               else
12.                     A[k] = A2[j]
13.                     j = j + 1
```

- ## Quick Sort :-

Best Case O(nlog(n))
Worst Case O(n^2)
Average Case O(nlog(n))

❑ **Pseudo Code**

```
1.    pivot = A[r]      // Pivot
2.    i = p - 1
3.    j = r + 1
4.    while  TRUE
5.                do
6.                      j = j - 1
7.          while A[j] > pivot
8.                do
```

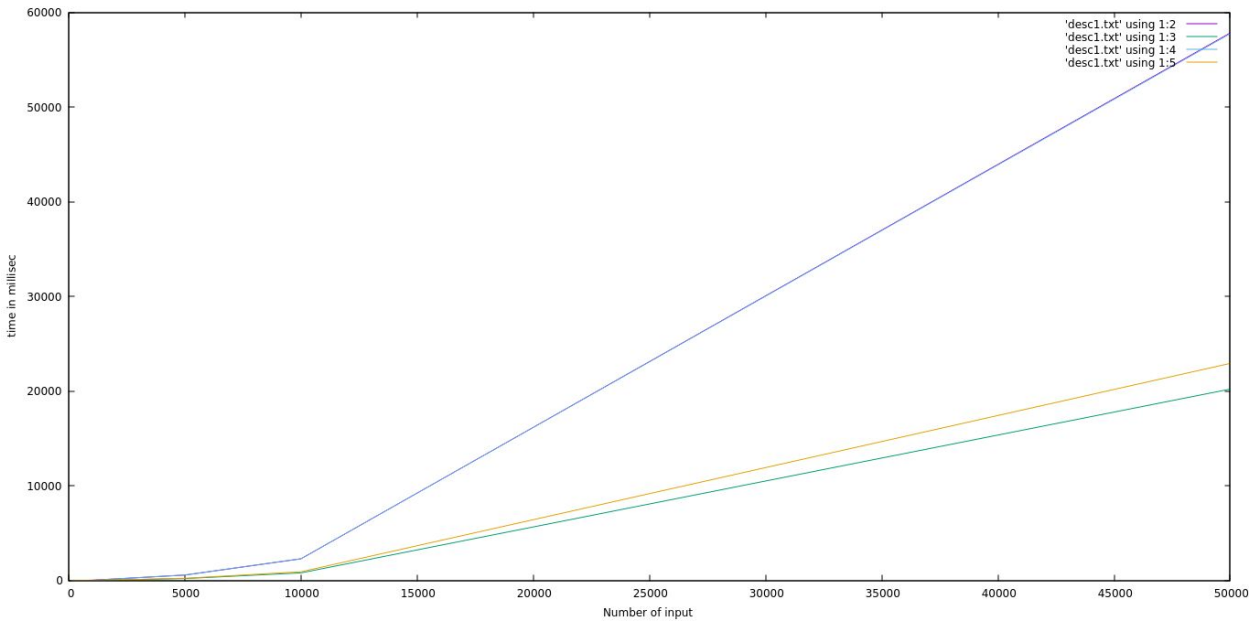
```
9.                  i = i + 1
10.             while A[i] < pivot
11.             if j  >  i
12.                  exchange A[i] with A[j]
13.             else if  j = i
14.                       return  j - 1
15                   else
16                        return  j
```

Graph1:-Graph is plotted between descending input i.e 100,500,1000,5000,10000,50000 for these numbers (x axis) and Time Required by insertion sort,selection sort bubble and rank .

- Output

Input Size	Insertion	Selection	Bubble	Rank
100	0.386	0.127	0.342	0.147
500	8.32	2.743	6.242	2.282
1000	26.517	7.836	22.573	9.244
5000	572.337	200.453	578.077	232.907
10000	2288.84	803.83	2301.46	914.869
50000	57827.5	20205.1	57744.2	22921.9

- Here we can see that after 10000 (as input increases), time complexity follows as
    - bubble >Rank >Insertion >Selection .

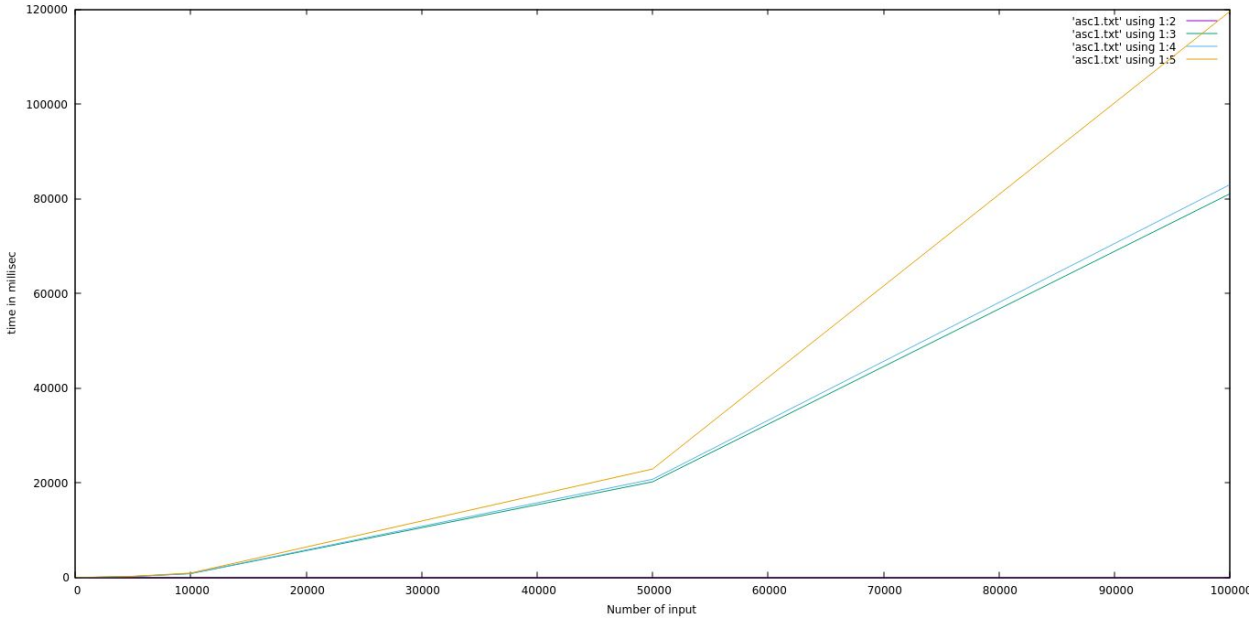


Graph2:-Graph is plotted between ascending input i.e 100,500,1000,5000,10000,50000 for these numbers (x axis) and Time Required by insertion sort,selection sort bubble and rank .

- **Output :-**

Input Size	Insertion	Selection	Bubble	Rank
100	0.005	0.111	0.109	0.13
500	0.019	4.631	3.114	2.614
1000	0.023	9.875	9.172	9.251
5000	0.109	203.279	218.486	241.842
10000	0.179	802.569	827.963	919.367
50000	1.058	20212.6	20751.9	22916.9
100000	1.83	81063.2	82994.4	119592

- **Conclusion** :-Here we can see that after 50000 (as input increases), time complexity follows
    - Rank>bubble>Selection >Insertion .



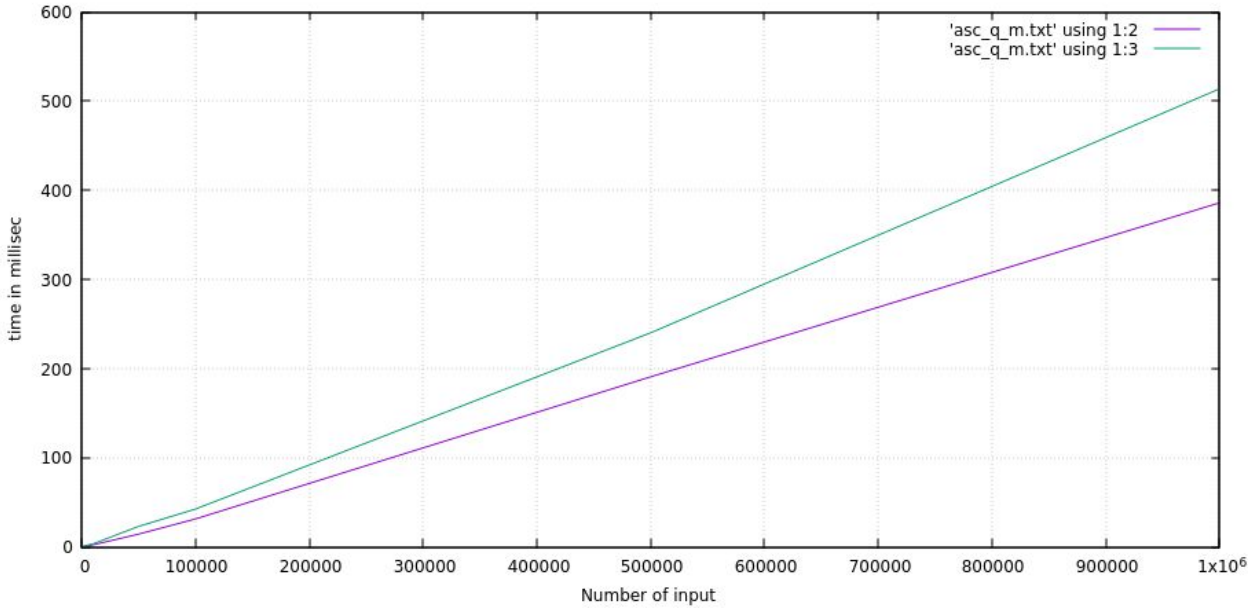
Graph3:-Graph is plotted between ascending input i.e 100,500,1000,5000,10000,50000 for these numbers (x axis) and Time Required by Merge and Quick .

- **Output :-**

Input Size	Quick	Merge
100	0.027	0.045
500	0.215	0.322
1000	0.265	0.706
5000	1.192	2.035
10000	2.538	3.59
50000	14.495	23.161
100000	31.451	42.59
500000	190.956	240.243
1000000	386.163	513.95

- **Conclusion** :-time complexity follows

Quick < Merge .



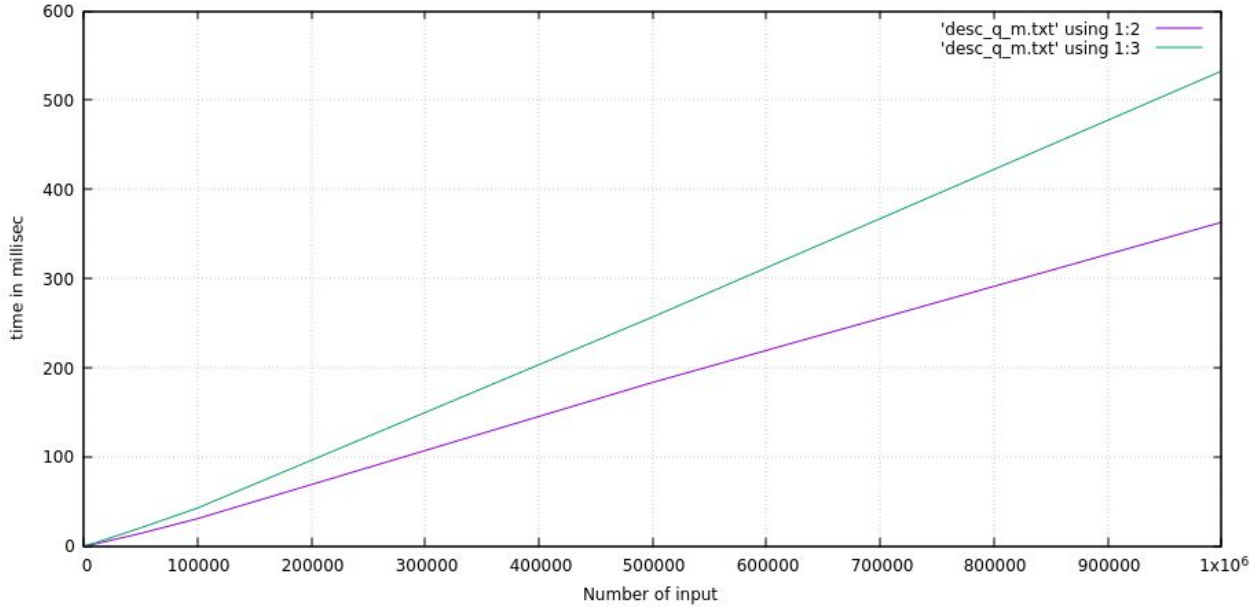
Graph 4:-Graph is plotted between descending input i.e 100,500,1000,5000,10000,50000 for these numbers (x axis) and Time Required by Merge and Quick .

- **Output :-**

Input Size	Quick	Merge
100	0.035	0.062
500	0.197	0.325
1000	0.29	0.403
5000	1.143	1.829
10000	2.377	3.611
50000	14.227	20.387
100000	30.617	42.551
500000	183.53	256.926
1000000	363.119	532.839

- **Conclusion** :Time complexity follows

Quick < Merge



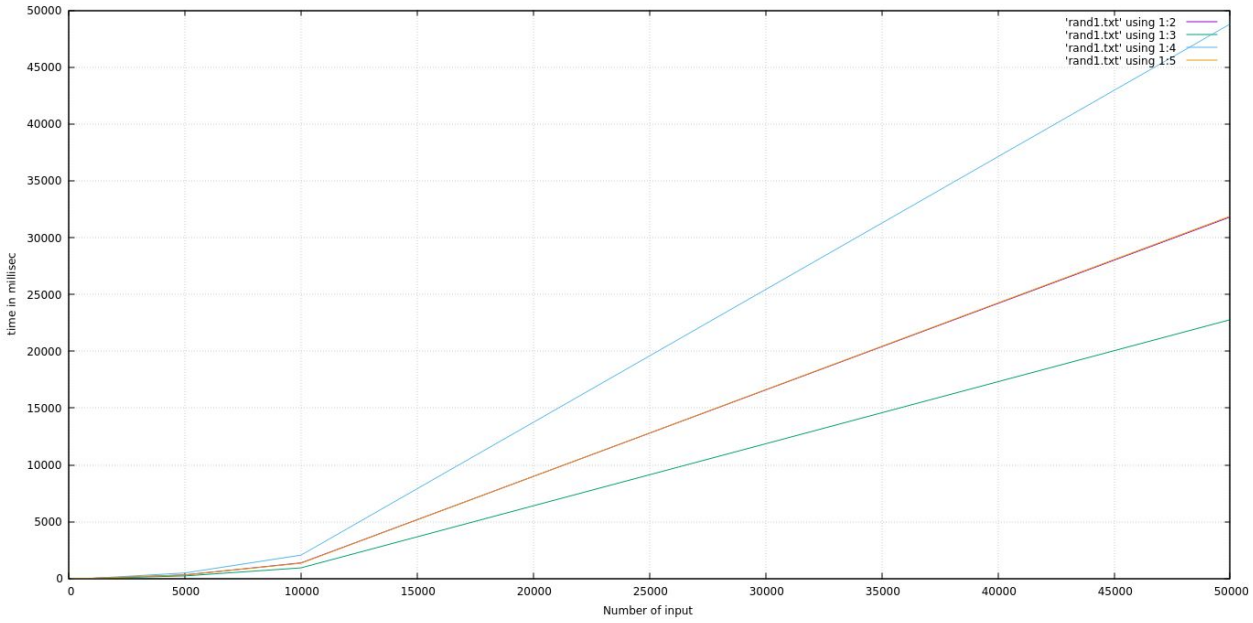
Graph 5:-Graph is plotted between Random input i.e 100,500,1000,5000,10000,50000 for these numbers (x axis) and Time Required by insertion sort,selection sort bubble and rank .

- **Output :-**

Input Size	Insertion	Selection	Bubble	Rank
100	0.195	0.138	0.263	0.178
500	6.151	2.522	4.79	3.079
1000	13.403	9.134	18.604	12.777
5000	315.681	223.896	483.48	317.624
10000	1367.21	934.728	2051.34	1346.99
50000	31812.1	22776.7	48797.4	31884.7

Conclusion :-Here we can see that after 10000 (as input increases), time complexity follows

bubble >Rank >Insertion >Selection .



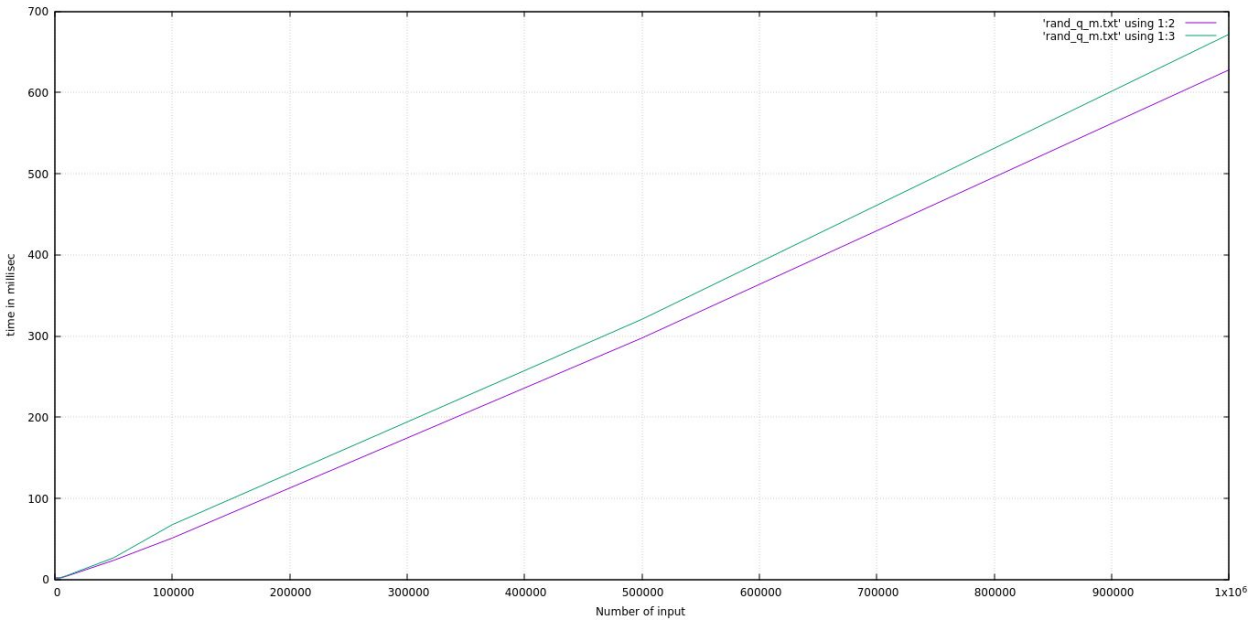
Graph 6:-Graph is plotted between Random input i.e 100,500,1000,5000,10000,50000 for these numbers (x axis) and Time Required by Merge and Quick .

- **Output :-**

Input Size	Merge	Quick
100	0.035	0.062
500	0.197	0.325
1000	0.29	0.403
5000	1.143	1.829
10000	2.377	3.611
50000	14.227	20.387
100000	30.617	42.551
500000	183.53	256.926
1000000	363.119	532.839

- **Conclusion** :-Time complexity follows

Merge>Quick



References:

1.Class Notes Uploaded On moodle ‘

2.http://quiz.geeksforgeeks.org/merge-sort/

3.http://quiz.geeksforgeeks.org/quick-sort/

4.https://en.wikipedia.org/wiki/Selection_sort

5.https://en.wikipedia.org/wiki/Insertion_sort

6.https://ds4beginners.wordpress.com/2006/09/22/rank-sort/

7.http://stackoverflow.com/questions/10792015/gnuplot-plotting-multiple-line-graphs