Homework Assignment 2

Name: **Vikhyat Dhamija**
RU ID:194003013
Net ID: **vd283**

1. **Implement Shell sort which reverts to insertion sort. (Use the increment sequence 7, 3, 1). Create a plot for the total number of comparisons made in the sorting the data for both cases (insertion sort phase and shell sort phase).**

   Explain why Shell sort is more effective than Insertion sort in this case.

   Also, discuss results for the relative (physical wall clock) time taken when using (i) Shell sort that reverts to insertions sort, (ii) Shell sort all the way.

   Solution:

   a. Python Code of the implementation of shell sort which reverts to insertion sort has been done (Note ---Shell Sort is made a function with h i.e. the gap parameter and from the main function for this sequence 7 , 3 ,1 the shell sort(data,7), shell sort(data,3), Shell sort(data,1) are called to implement whole shell sort with this 7 ,3,1 sequence and as shell sort with h=1 is implicitly an insertion sort.

      In this question I have considered three cases:-

      1. Shell Sort all the way(7,3,1)
      2. Shell Sort that reverts to the insertion sort(Shell sort with h=7 and then insertion sort)
      3. Insertion Sort

b. Effectiveness of the Shell Sort over Insertion Sort

S.No	Data Set	Data Size	Shell Sort with =7,3,1 (Number of Comparisons)	Insertion Sort(Number of Comparisons)	Clock Time for Shell sort reverting to insertion sort(ms)	Clock Time for Insertion Sort (ms)
1	data0.1024	1024	3061	1023	1.519203186	0.495910645
2	data0.2048	2048	6133	2047	2.493858337	0.991106033
3	data0.4096	4096	12277	4095	4.463672638	1.487970352
4	data0.8192	8192	24565	8191	9.475708008	2.976417542
5	data0.16384	16384	49141	16383	20.80249786	5.956411362
6	data0.32768	32768	98293	32767	38.20157051	12.89176941
7	data1.1024	1024	46728	265553	13.39316368	12.40181923
8	data1.2048	2048	169042	1029278	42.7107811	273.9171982
9	data1.4096	4096	660619	4187890	170.1657772	1109.094858
10	data1.8192	8192	2576270	16936946	665.636301	4489.548683
11	data1.16384	16384	9950922	66657561	2608.959913	18660.079
12	data1.32768	32768	39442456	267966668	10984.87306	70818.33577

Note that it can be clearly seen that when Shell sort is performed in 7 ,3,1 sequence it is very effective in case of the shuffled data but its performance is poor in comparison to insertion sort in case of sorted list. As insertion check whole list in one time but shell sort check it in its various sequences even including that of 1 i.e. implicitly insertion sort.

Data Set	Data Size	Shell Sort after h=7,3 ,1 all the way(Comparisons)	Shell Sort that reverts to insertion sort 7 to 1(Number of Comparisons)	Insertion Sort(Number of Comparisons)	Clock Time (Shell Sort)	Clock Time(Shell Sort 7to 1)	Clock Time Insertion Sort
data0.1024	1024	3061	2040	1023	1.519203	0.965834	0.495911
data0.2048	2048	6133	4088	2047	2.493858	1.481771	0.991106
data0.4096	4096	12277	8184	4095	4.463673	2.97761	1.48797
data0.8192	8192	24565	16376	8191	9.475708	6.88982	2.976418

data0.16384	16384	49141	32760	16383	20.8025	11.92927	5.956411
data0.32768	32768	98293	65528	32767	38.20157	25.29526	12.89177
data1.1024	1024	46728	55205	265553	13.39316	14.8797	12.40182
data1.2048	2048	169042	194681	1029278	42.71078	50.5929	273.9172
data1.4096	4096	660619	748856	4187890	170.1658	194.9282	1109.095
data1.8192	8192	2576270	2817462	16936946	665.6363	738.9901	4489.549
data1.16384	16384	9950922	10660730	66657561	2608.96	2704.152	18660.08
data1.32768	32768	39442456	41470765	2.68E+08	10984.87	10812.85	70818.34

c. **Relative performance of Shell Sort all the way(7,3,1) and the shell sort that reverts to the insertion sort(7-> insertion)**

From above it can be easily seen that

1. Shell Sort all the way(7,3,1)
2. Shell Sort that reverts to the insertion sort(Shell sort with h=7 and then insertion sort)
3. Insertion Sort

1 , 2 , 3 shows a decreasing trend of number of comparisons and the time taken in the sorted lists and the increasing trend in the unsorted list.

In time taken the 1 and 2 has not that big difference but both have great difference in time and comparison operations with the insertion sort.

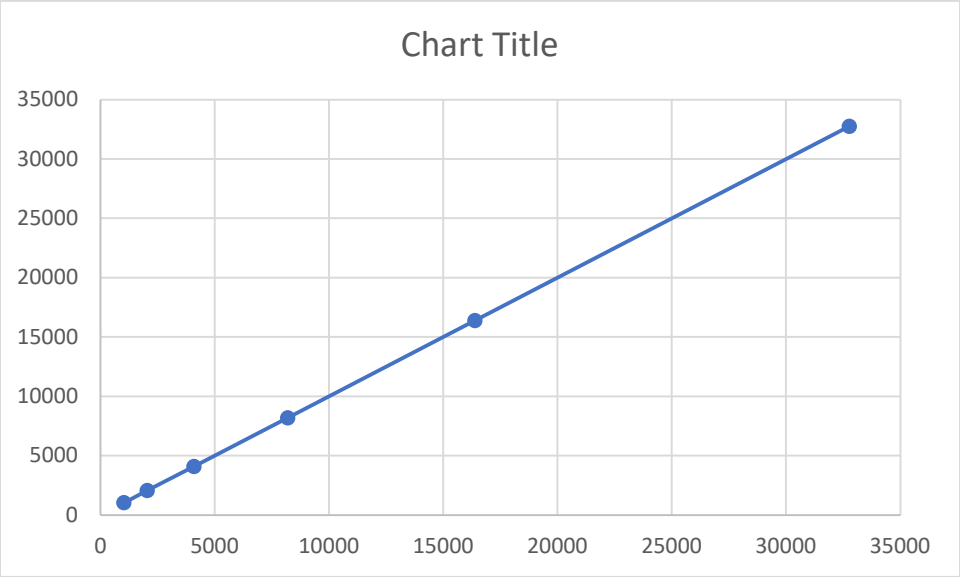
d. Total number of comparisons made in the sorting the data for both cases (insertion sort phase and shell sort phase)
The number of comparisons in the insertion sort phase and the shell sort phase in the shell sort implementation that reverts to insertion sort(7,1) are tabled below:

Data Set	Insertion Sort Phase	Shell Sort phase	Total
data0.1024	1023	1017	2040
data0.2048	2047	2041	4088
data0.4096	4095	4089	8184
data0.8192	8191	8185	16376
data0.16384	16383	16377	32760
data0.32768	32767	32761	65528
data1.1024	16595	38610	55205
data1.2048	46470	148211	194681
data1.4096	148065	600791	748856
data1.8192	392778	2424684	2817462
data1.16384	1127104	9533626	10660730

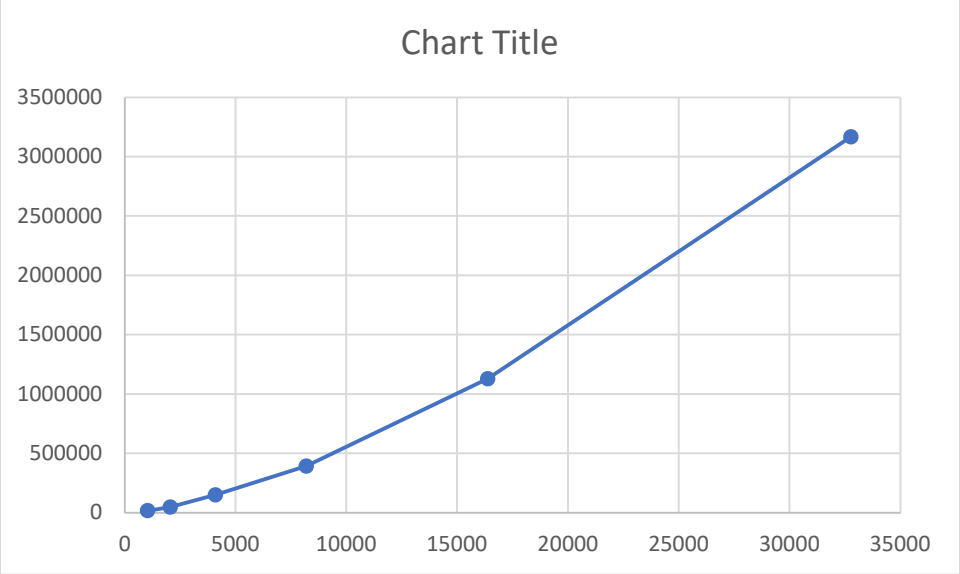
data1.32768	3167309	38303456	41470765

A. Plot of Insertion Sort Phase comparisons

1. Data0 (Sorted Datasets)

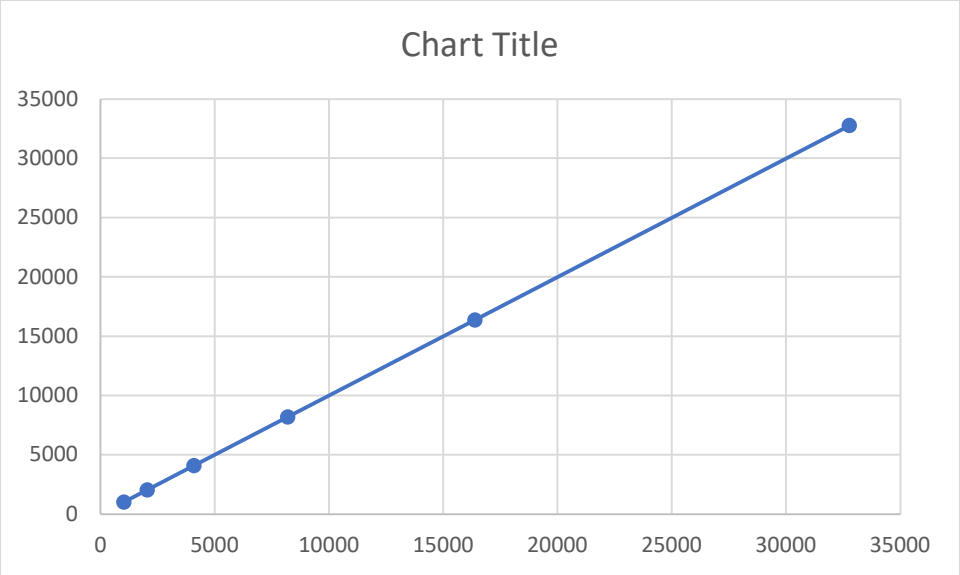


2. Data1 (Unsorted Datasets)

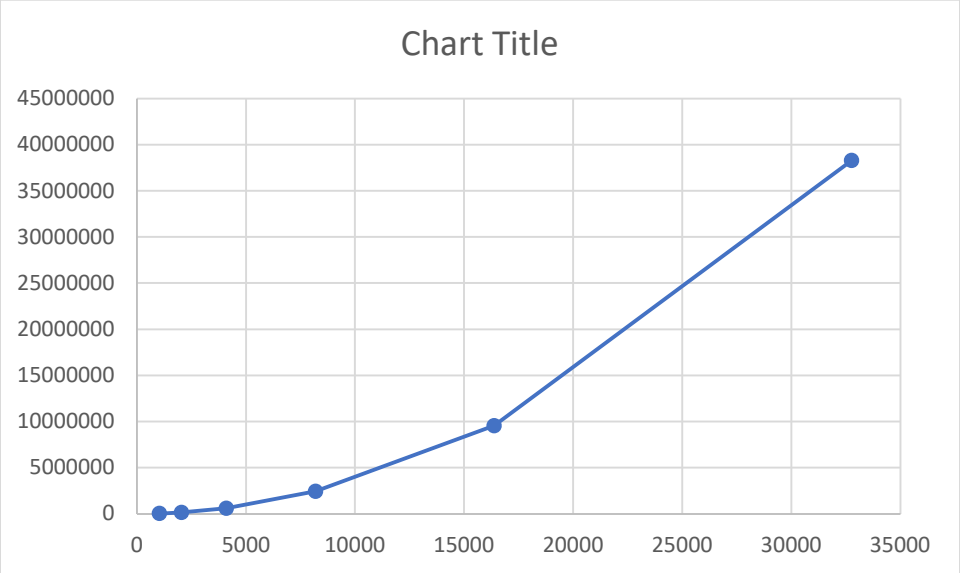


B. Plot of Shell Sort Phase comparisons

1. Data0 (Sorted Datasets)



2. Data1(Unsorted Datasets)

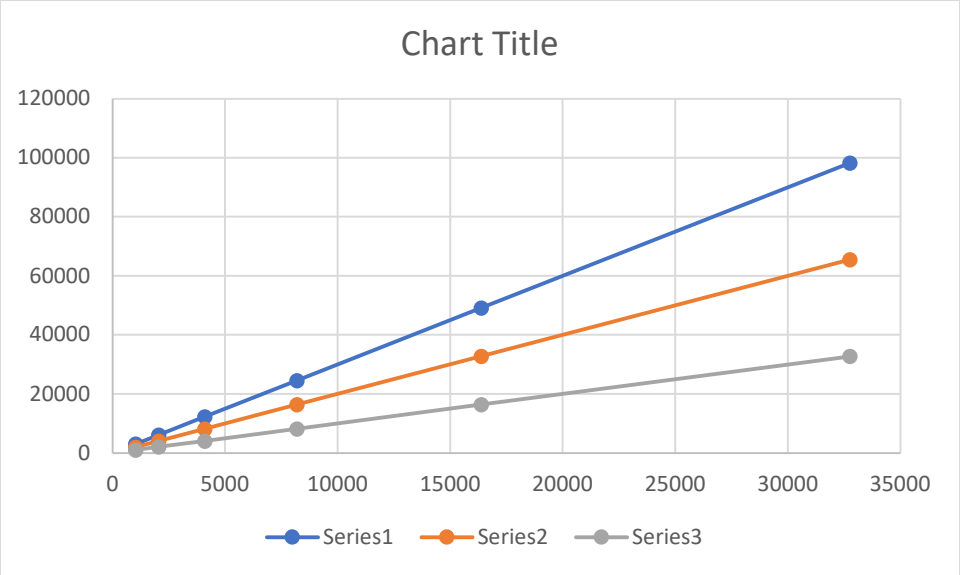


Please note that I have compared all variants 1,2,3 (as mentioned above) in the tabled data before. Further here is the graphical representation for better clarity.

A. **Data1(Shuffled Data)**

Series3- Insertion Sort , Series 2:- Shell Sort (h=7-> insertion) , Series 1:- (Shell Sort(7,3,1)

B. **Data0(Sorted Data)**



Series3- Insertion Sort , Series 2:- Shell Sort (h=7-> insertion) , Series 1:- (Shell Sort(7,3,1)

2. **The Kendall Tau distance is a variant of the "number of inversions". It is defined as the number of pairs that are in different order in two permutations. Write an efficient program that computes the Kendall Tau distance in less than quadratic time on average. Plot your results and discuss. [Use the dataset linked after Q4. Note: data0.* for convenience is an ordered set of numbers (in powers of two). data1.* are shuffled data sets of sizes (as given by "*").]**

Solution:

As mentioned , **Kendall Tau distance is defined as the number of pairs that are in different order in two permutations.**

It is a variant of **number of inversions.**

As mentioned , Datasets(Permutations) being used are the **data0.* datasets** as one permutation , which are ordered ones and the corresponding **data1.* datasets** as the other permutation which are the shuffled ones.

Then , the number of inversions(i.e. when A[i] > A[j] for i less than j) are being calculated using the divide and conquer method as used in the merge sort.

Logic of the algorithm

Same as the merge sort , first we divide the list or permutation in half in every stage till it reaches 1 and then they are merged in the reverse direction of division.

When the merging is being done then certain numbers are shifted from right moves ahead of certain left elements , by doing each number comparison between the left subarray and the right sub array(left and right are the sub arrays to be merged) so as to make one sorted merged or combined list from two left and right sub array.

Now if the element from right moves ahead certain number of elements in the left list then that means that , that number from the right has inverted pairs in comparison to sorted list , equal to the number of elements in the left sub list from which it moves ahead.

So by capturing such instances in the merge sort processes we counted the number of inverted pairs with respect to the ordered list as provided , which is also equal to **Kendall Tau Distance.**

Plot and Analysis

S.No	Data Set 1	Data Set 2	Data Size	Kendall Tau distance using Brute force Quadratic Algorithm(A)	Number of Comparisons using Merge Sort based method for calculating Kendall Tau distance(B)	Comparisons in A (O(n^2))	Comparisons in B(O(n Log n)
1	data0.1024	data1.1024	1024	264541	264541	523776	10240
2	data0.2048	data1.2048	2048	1027236	1027236	2096128	22528
3	data0.4096	data1.4096	4096	4183804	4183804	8386560	49152
4	data0.8192	data1.8192	8192	16928767	16928767	33550336	106496
5	data0.16384	data1.16384	16384	66641183	66641183	134209536	229376
6	data0.32768	data1.32768	32768	267933908	267933908	536854528	491520

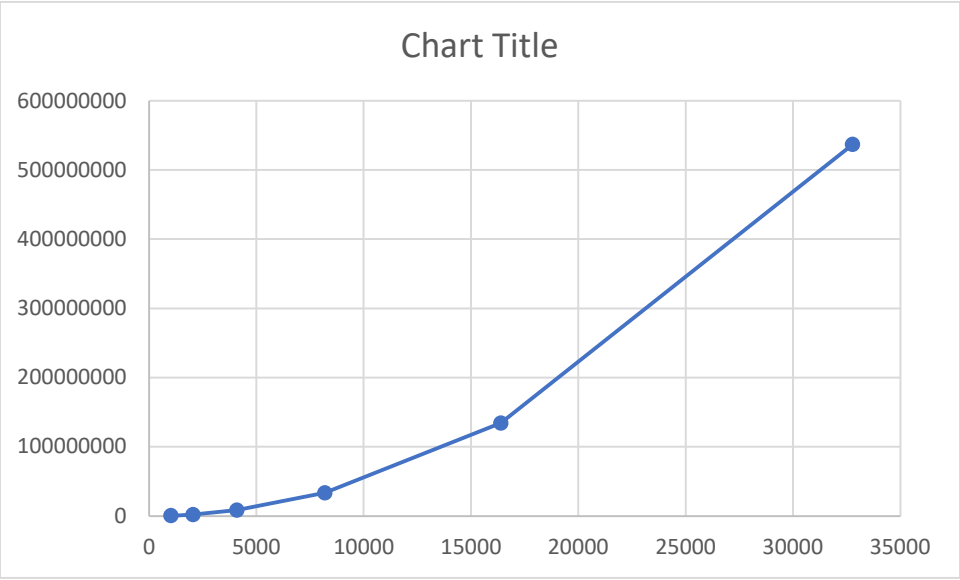
I have compared two methods here:-

a. Quadratic method – which is n^2 algorithm using two for loops and forming all the pairs and then comparing the pairs in each data set whether they are inverted or not.
b. Merge Sort method as explained:-

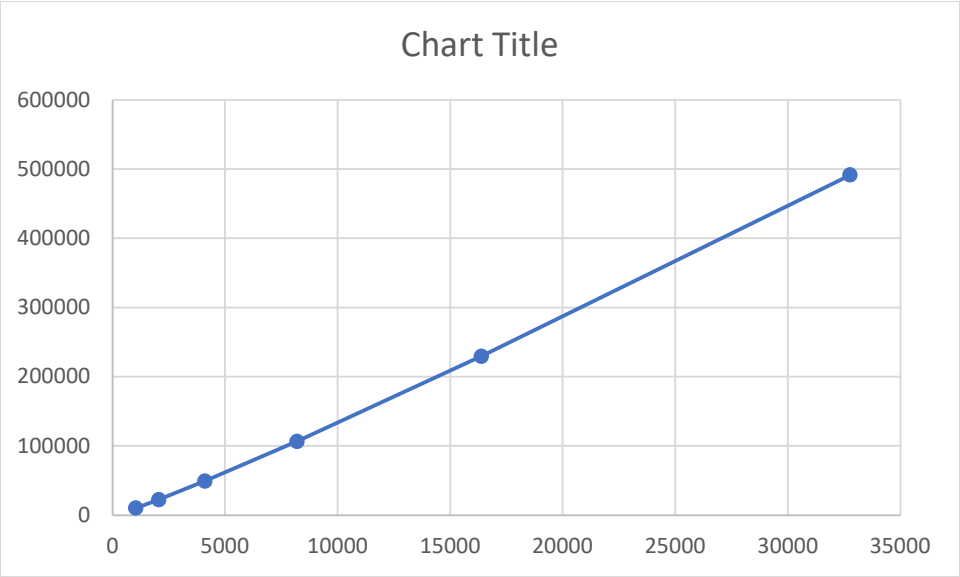
For comparison , I have used the complexity counter in order to measure the number of comparisons(cost model) in both the algorithms for calculating the Kendall Tau distance and as it can be clearly seen and obvious from the algorithm structure of both that for **b** algorithm number of comparisons will be N Log N as is clearly predicted from the data and in case of **a** algorithm the number of comparisons are quadratic.

Plots are as follows:

A.



B.



3. **Implement the two versions of Merge Sort that we discussed in class. Create a table or a plot for the total number of comparisons to sort the data (using data set here) for both cases.**

   Discuss (i) relative number of operations, (ii) relative (physical wall clock) time taken.

   Solution:

   Two methods of Merge Sort as discussed in the class are :-

   a. **Top Down Merge Sort Method based on Recursion**
   b. **Bottom Up method based on iteration**

   As mentioned in the question , first we have to table the data of the total number of comparisons for sorting the data and then we have to compare these operations and the time taken by the algorithms.

   Data gathered as follows:-

   A. **Top Down Approach Using recursion**

S.No	Data Set	Data Size	Number of Comparisons	Time Taken(ms)
1	data0.1024	1024	10240	3.968238831
2	data0.2048	2048	22528	10.41769981

3	data0.4096	4096	49152	22.3197937
4	data0.8192	8192	106496	45.21536827
5	data0.16384	16384	229376	93.26434135
6	data0.32768	32768	491520	193.9358711
7	data1.1024	1024	10240	4.958868027
8	data1.2048	2048	22528	10.91170311
9	data1.4096	4096	49152	23.27418327
10	data1.8192	8192	106496	47.11842537
11	data1.16384	16384	229376	103.1804085
12	data1.32768	32768	491520	221.2209702

B. Bottom Up Approach using iteration

S.No	Data Set	Data Size	Number of Comparisons	Time Taken(ms)
1	data0.1024	1024	10240	4.014492035
2	data0.2048	2048	22528	9.423971176
3	data0.4096	4096	49152	20.83158493
4	data0.8192	8192	106496	47.13249207
5	data0.16384	16384	229376	91.76111221
6	data0.32768	32768	491520	194.4320202
7	data1.1024	1024	10240	4.925727844
8	data1.2048	2048	22528	11.40809059
9	data1.4096	4096	49152	22.86314964
10	data1.8192	8192	106496	47.21832275
11	data1.16384	16384	229376	101.7985344
12	data1.32768	32768	491520	210.3040218

(i) Relative number of operations

As it can be seen from the above data that the number of comparisons are the same for both the approaches as the concept of divide and merge is the same just the difference is the direction i.e. from big to small and then merging or small merging to become big.
This is being clearly reflected from the number of comparisons data for both algorithms.

{Please note that the counter is placed at the while loop of merge operation as For example N/2 and N/2 subsets are merged then the loop moves N times to pick each

element and put it into the auxiliary array. So this counter better reflects the cost of the algorithm.

In bottom up approach , merge is called from the loops and in the top down through recursion so as the results are same that means the merge is called the same time for both.}

(ii) Relative (physical wall clock) time taken.

S.No	Data Set	Time Taken(ms)(A)	Time Taken(ms)(B)
1	data0.1024	3.968238831	4.014492035
2	data0.2048	10.41769981	9.423971176
3	data0.4096	22.3197937	20.83158493
4	data0.8192	45.21536827	47.13249207
5	data0.16384	93.26434135	91.76111221
6	data0.32768	193.9358711	194.4320202
7	data1.1024	4.958868027	4.925727844
8	data1.2048	10.91170311	11.40809059
9	data1.4096	23.27418327	22.86314964
10	data1.8192	47.11842537	47.21832275
11	data1.16384	103.1804085	101.7985344
12	data1.32768	221.2209702	210.3040218

From the Data it appears that the running time of two algorithms is almost similar i.e. the time taken is almost similar
(Note that the time taken is in milli seconds so not much difference is noticeable)

But it needs to be kept in mind that difference appear in case of big datasets like in my case of 32768 items where it is noticeable.

From the theoretical studies about the comparison of these two algorithms , it is said that that in many cases recursion is faster than iteration because of caching improved performances. So the top down merge sort can be faster than bottom up merge sort because of caching reasons. But in my case opposite is being seen .
This may be due to the fact that these caching etc. depends on machine's hardware and operating platforms. So this may be the reason for such results . But even after that , the execution times of both algorithms are almost similar.

(I had reiterated many times results are very similar so it is difficult to comment that which one is faster but the whole processes are same just the way is different i.e. in A recursion is used and in the B iteration is used)

4. **Create a data set of 8192 entries which has in the following order:**
   1024 repeats of 1,
   2048 repeats of 11,
   4096 repeats of 111 and
   1024 repeats of 1111.

   Write a sort algorithm that you think will sort this set "most" effectively. Explain why you think so.

   Solution:

   As the list mentioned here is sorted with having only few keys which are repeated :

   Now let us evaluate in terms of algorithm we have studied :

   For sorted array/list :-

   1. ~~Selection Sort~~
      As the complexity will be n^2 even in case of the sorted array

   2. Bubble Sort
      It can be O(n) in this case by putting the condition that if in first loop no inversions take place which means that array is already sorted.

   3. Insertion Sort
      It will be O(N)

   4. Merge Sort
      It will be N log N as this is its average , best and worst O(N)

      Note that when optimized , the merge sort can be really effective as when it is going to merge it will check the last element of the left sub array(a) and the first element of the right sub array(b) and when b is greater than a then that means the two lists are in order so need of merging , so no comparisons.

   5. Quick sort

It becomes O(N^2) in case of sorted and duplicates as first element is taken out and the partition of the rest is made and this goes on.

We have seen the 3 -way Quick Sort (Dutch Flag Algorithm) – that would have been the good choice if the list would not have been ordered.
If we consider this algorithm with list in sorted order as mentioned ,
First time it will scan the whole list i.e. N comparisons to make one partition that of 1 if key of 1 is chosen and then the rest groups of particular key and so .
It is just a quick sort but which takes all duplicate elements of key or pivot together and make the three groups one of less than key , other greater than key and other equal to key . So it is an optimized quick sort which would have been the best choice when we do not know that the list is ordered and just know that it contains duplicates and then with reshuffling or so and then using this algorithm we would have got the best result and thus would have been our choice.

From the above description it seems that insertion sort and optimized Merge sort and optimized bubble sort works well for sorted ordered list.

I have made the code and ran insertion sort , merge sort , optimized merge sort and quick sort(quick sort could not run due to maximum recursion depth reached in python but we know it will take quadratic time) and got the following results:-

S.No.	Sorting Algorithm	Complexity counter(of comparisons)	Time of Execution(ms)
1	Insertion Sort	8191	2.975463867
2	Merge Sort	106496	45.63188553
3	Optimized Merge Sort	0	3.502368927
4	Quick Sort	NA	NA

From the time factor it seems that insertion sort is performing better but optimized merge sort is also very good with just recursions and no merging which can be even better considering the theoretical aspect that caching can improve the recursion performance.

5. **Implement Quicksort using median-of-three to determine the partition element. Compare the performance of Quicksort with the Merge sort implementation and dataset from Q3.**

Is there any noticeable difference when you use N=7 as the cut-off to insertion sort.

Experiment if there is any value of "cut-off to insertion" at which the performance inverts.

Solution:

(i) Implementation of Quicksort using median-of-three has been presented in terms of the python code.

(ii) Performance of Quicksort Vs Merge sort

Note that for comparison of the complexity we have taken the counter for the number of comparisons happening . As they are just the proxy to show us the algorithm complexity and then when we see the data we can say that using quick sort median of three , the performance achieved is O(N log N) which is also the case with merge sort.

Then we look at the time of execution performance :-

We can easily see that the Quick Sort is faster .

S.No	Data Set	Data Size	Quick sort(Normal) Complexity	Merge sort complexity	Merge Sort Time Taken(ms)	Quick sort Time Taken(ms)
1	data0.1024	1024	8204	10240	3.968238831	1.983642578
2	data0.2048	2048	18445	22528	10.41769981	5.454778671
3	data0.4096	4096	40974	49152	22.3197937	21.3830471
4	data0.8192	8192	90127	106496	45.21536827	39.2203331
5	data0.16384	16384	196624	229376	93.26434135	50.573349
6	data0.32768	32768	426001	491520	193.9358711	91.25757217
7	data1.1024	1024	9279	10240	4.958868027	3.008127213
8	data1.2048	2048	21509	22528	10.91170311	14.89281654
9	data1.4096	4096	47102	49152	23.27418327	28.81169319
10	data1.8192	8192	102622	106496	47.11842537	40.17186165
11	data1.16384	16384	232715	229376	103.1804085	64.43119049
12	data1.32768	32768	504160	491520	221.2209702	124.9902248

Reasons may be that –

a. No auxiliary array , No moving data from auxiliary to other and so on , which are there in the merge sort
b. Quick Sort is an in place sorting algorithm with no extra space complexity and related operation
c. Only condition for better performance of the quick sort is that the pivot should be close to middle and using the median of 3 method it shows that we are achieving good performance.

(iii) **Difference when we use N=7 as the cut-off to insertion sort.**

S.No	Data Set	Merge Sort Time Taken(ms)	Quick sort Time Taken(ms)	Quick sort(with insertion sort cut off) Time Taken(ms)
1	data0.1024	3.968238831	1.983642578	2.479553223
2	data0.2048	10.41769981	5.454778671	3.967046738
3	data0.4096	22.3197937	21.3830471	7.439613342
4	data0.8192	45.21536827	39.2203331	17.85802841
5	data0.16384	93.26434135	50.573349	39.1895771
6	data0.32768	193.9358711	91.25757217	75.93679428
7	data1.1024	4.958868027	3.008127213	3.424167633
8	data1.2048	10.91170311	14.89281654	6.005764008
9	data1.4096	23.27418327	28.81169319	13.89718056
10	data1.8192	47.11842537	40.17186165	29.78181839
11	data1.16384	103.1804085	64.43119049	56.54811859
12	data1.32768	221.2209702	124.9902248	122.5318909

From the above observed data , it can be seen that the performance of the quick sort has been improved with insertion sort cut off of 7.

(iv) Analysis by varying the cut off N for reverting to insertion sort in quick sort

Data Set	Quick sort Time Taken(ms)	Quick sort(with cutoff n=2 insertion sort) Time Taken	Quick sort(with cutoff n=4 insertion sort) Time	Quick sort(with cutoff n=6 insertion sort) Time	Quick sort(with insertion sort cut off=7) Time Taken(ms)	Quick sort(with cutoff n=8 insertion sort) Time	Quick sort(with cutoff n=10 insertion sort) Time
data0.1024	1.983642578	1.954317	1.487255	1.983166	2.479553	2.478123	2.498865
data0.2048	5.454778671	4.465818	4.46558	4.518986	3.967047	3.935575	3.502846
data0.4096	21.3830471	19.95826	8.431673	11.4069	7.439613	8.398533	7.931232
data0.8192	39.2203331	19.92369	19.37103	20.08557	17.85803	16.33215	16.66474
data1.1024	3.008127213	4.462957	2.48003	2.976418	3.424168	2.471685	3.470898
data1.2048	14.89281654	9.909391	6.947994	5.465508	6.005764	8.919001	10.96964
data1.4096	28.81169319	12.89439	23.31471	16.33549	13.89718	18.79239	12.88342
data1.8192	40.17186165	30.29299	30.2186	28.76639	29.78182	30.25627	33.56171

data1.16384	64.43119049	60.50849	58.57587	60.01711	56.54812	59.98254	59.51905
data1.32768	124.9902248	129.95	123.0078	130.445	122.5319	125.4785	126.9901

Analysis Points from the above tabled data

a. As it can be seen in the bigger size shuffled data like of 32768 and 16384 , the performance is the fastest for N=7 and then it decreases as the time taken increases. So it seems that N=7 is the insertion sort cut off in such cases
b. Further in case of sorted data , as obvious in the bigger data sets like here of 8192 , 4096 the performance seems to be increasing i.e. the time of execution seems to be decreasing with increasing N . May be it points to the fact that for sorted data is the worst case for quick sort (note that in our case we are taking median of three) and when more and more data is going to insertion sort for sorting the performance is increasing.

6. **Extra Points: View the following Data Set here. The column on the left is the original input of strings to be sorted or shuffled; the column on the extreme right are the string in sorted order; the other columns are the contents at some intermediate step during one of the 8 algorithms listed below.**
   Match up each algorithm under the corresponding column. Use each algorithm exactly once: (1) Knuth shuffle (2) Selection sort(3) Insertion sort (4) Merge sort(top-down)(5) Merge sort (bottom-up) (6) Quicksort (standard, no shuffle) (7) Quicksort (3-way, no shuffle) (8) Heapsort.

Solution:- Column 1 and Column 10 are input and output of the sorting Algorithm.

Column 2:-Merge Sort bottom up

Column 3:- Quick Sort

Column 4:- Knoth Shuffle

Column 5:- Merge Sort Top Down

Column 6:-Insertion Sort

Column 7:-Heap Sort

Column 8:-Selection Sort

Column 9:-Quick Sort with 3- way partition(Dutch National Flag algorithm) of Dijakstra

Explanation of the Process used

Column 8 – In selection sort the minimum is put in the first place and then the same process is repeated for the rest list . So Column 8 clearly showed this pattern and then when after seeing the swapping that happen in selection sort with paper and pencil , it came out to be selection.

Column 7 – When the array with 2i+1 and 2i+2 elements children of the node considered in forming the heap , then the heap was formed with paper and pencil , it clearly showed the max heap with wine on top . So Heap Sort.

Column 6:- Insertion sort has sorted list in the front and one by one elements are placed in the correct position and that was found in Column 6.

Column 2- Clearly 4 elements merged sorted groups were present so merge sort

Column 5:- Clearly 6 elements sorted groups were present indicating top down merge sort where 24-> 12->6->3->2,1 and then 6 groups sorted were found.

Column 3:- Quick Sort as navy chosen as the first key was placed in correct position and the partitioning was seen properly.

Column 9 :- 3 way merge sort as the first one was taken as key and then anything coming was put at the last and the smaller ones before the key . Such pattern was clearly observed.

Column 4 :- Knoth shuffle at the last

1……2……..3…….4……5……6……..7……8……9…….10

navy	coal	corn	blue	blue	blue	wine	bark	mist	bark
plum	jade	mist	gray	coal	coal	teal	blue	coal	blue
coal	navy	coal	rose	gray	corn	silk	cafe	jade	cafe
jade	plum	jade	mint	jade	gray	plum	coal	blue	coal
blue	blue	blue	lime	lime	jade	sage	corn	cafe	corn
pink	gray	cafe	navy	mint	lime	pink	dusk	herb	dusk
rose	pink	herb	jade	navy	mint	rose	gray	gray	gray
gray	rose	gray	teal	pink	navy	jade	herb	leaf	herb
teal	lime	leaf	coal	plum	pink	navy	jade	dusk	jade
ruby	mint	dusk	ruby	rose	plum	ruby	leaf	mint	leaf
mint	ruby	mint	plum	ruby	rose	pine	lime	lime	lime
lime	teal	lime	pink	teal	ruby	palm	mint	bark	mint
silk	bark	bark	silk	bark	silk	coal	silk	corn	mist
corn	corn	navy	corn	corn	teal	corn	plum	navy	navy
bark	silk	silk	bark	dusk	bark	bark	navy	wine	palm
wine	wine	wine	wine	leaf	wine	gray	wine	silk	pine
dusk	dusk	ruby	dusk	silk	dusk	dusk	pink	ruby	pink
leaf	herb	teal	leaf	wine	leaf	leaf	ruby	teal	plum
herb	leaf	rose	herb	cafe	herb	herb	rose	sage	rose
sage	sage	sage	sage	herb	sage	blue	sage	rose	ruby
cafe	cafe	pink	cafe	mist	cafe	cafe	teal	pink	sage
mist	mist	plum	mist	palm	mist	mist	mist	pine	silk
pine	palm	pine	pine	pine	pine	mint	pine	palm	teal
palm	pine	palm	palm	sage	palm	lime	palm	plum	wine