DSA Lab
Name: Siddhi Parekh
Reg No.: 221071047
Batch: C
SY Comps

Experiment No.: 8

AIM:
The aim of this experiment is to implement and execute the Insertion Sort and Merge Sort algorithms and subsequently analyze their time complexity.

THEORY:
Insertion Sort:
Algorithm Overview:
Insertion Sort is a simple sorting algorithm that builds the final sorted array one element at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort. However, insertion sort provides several advantages in its simplicity and ease of implementation.

Time Complexity Analysis:
Worst-Case Time Complexity (O(n^2)): The worst-case time complexity occurs when the array is in reverse order. In each iteration, the algorithm compares and shifts elements, leading to quadratic time complexity.

Best-Case Time Complexity (O(n)): The best-case time complexity occurs when the array is already sorted. In this scenario, insertion sort makes minimal comparisons, resulting in linear time complexity.

Average-Case Time Complexity (O(n^2)): On average, insertion sort exhibits quadratic time complexity, making it less suitable for large datasets.

Merge Sort:

Algorithm Overview:

Merge Sort is a divide-and-conquer algorithm that recursively divides the input array into two halves, sorts each half, and then merges the sorted halves. It ensures stability and guarantees a time complexity of O(n log n).

Time Complexity Analysis:

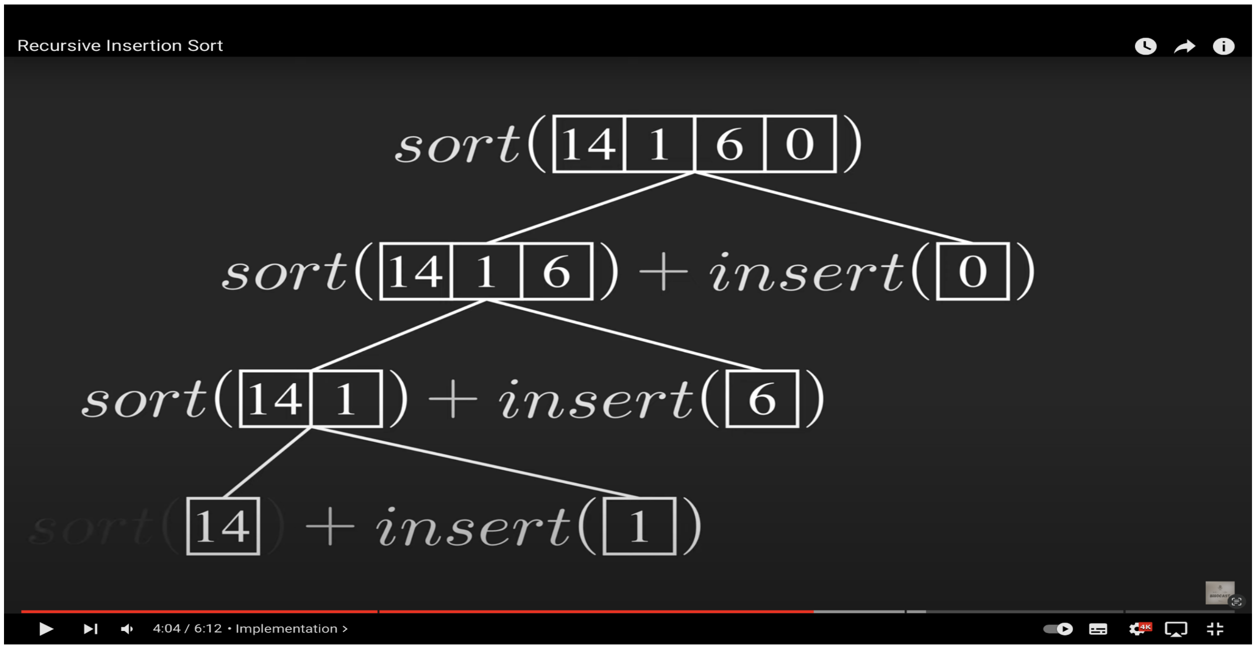
Worst-Case Time Complexity (O(n log n)): The worst-case time complexity of merge sort is O(n log n), making it efficient for large datasets. The algorithm consistently divides the array into halves until individual elements, and then merges them in a sorted manner.

Best-Case Time Complexity (O(n log n)): The best-case time complexity is also O(n log n). Unlike other algorithms, merge sort's performance is consistent across different scenarios.

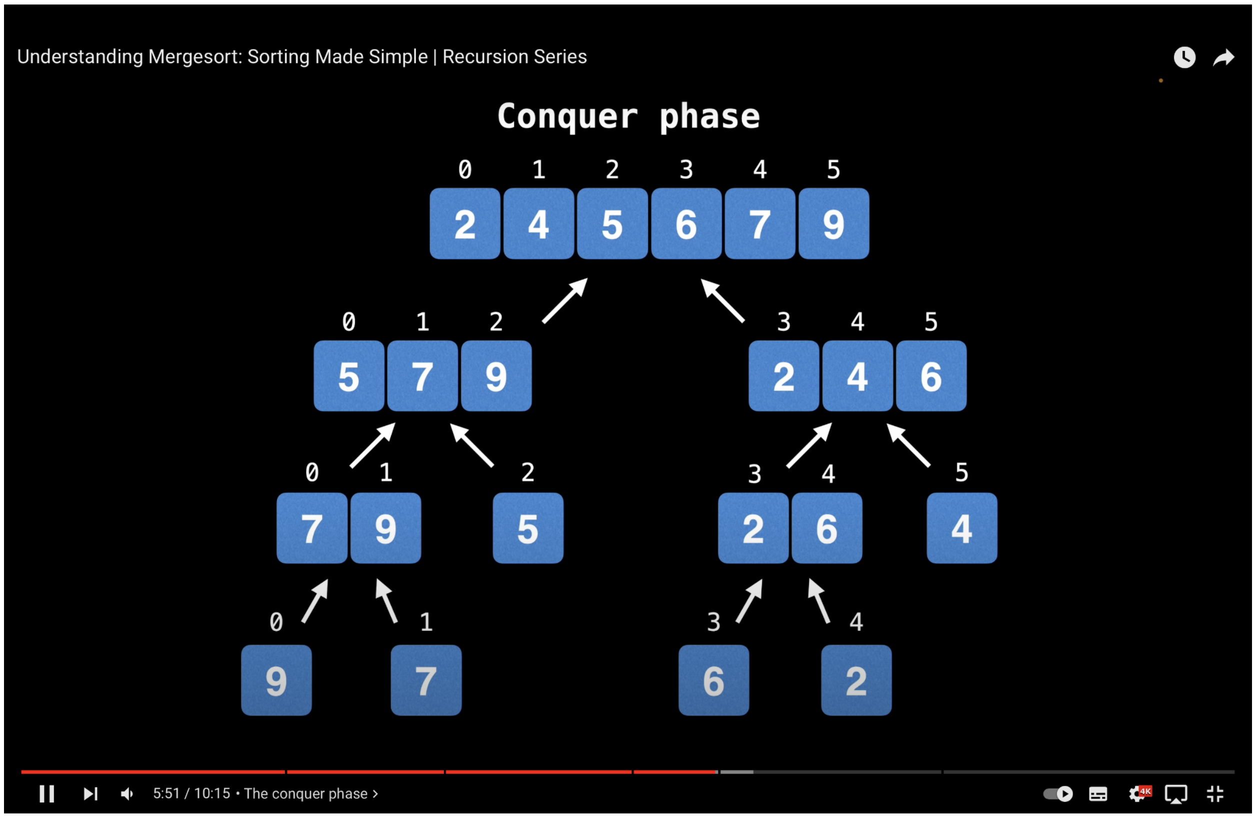
Average-Case Time Complexity (O(n log n)): Merge sort maintains its O(n log n) time complexity on average, making it a reliable choice for sorting.

EXAMPLE:

Insertion Sort:


Recursive Insertion Sort
$sort(\boxed{14}\boxed{1}\boxed{6}\boxed{0})$
$sort(\boxed{14}\boxed{1}\boxed{6}) + insert(\boxed{0})$
$sort(\boxed{14}\boxed{1}) + insert(\boxed{6})$
$sort(\boxed{14}) + insert(\boxed{1})$
4:04 / 6:12 • Implementation ›

Merge Sort:


Understanding Mergesort: Sorting Made Simple | Recursion Series
Conquer phase
0 1 2 3 4 5
2 4 5 6 7 9
0 1 2
5 7 9
3 4 5
2 4 6
0 1
7 9
2
5
3 4
2 6
5
4
0
9
1
7
3
6
4
2
5:51 / 10:15 • The conquer phase ›

ALGORITHM:

```
Step 0: Insert_sort(array)
            int j;
            int i;
            int key;
            while(j<length of array)
                key = array(j)
                i=j-1
                if(i>=0 && array(i)>key)
                   array(i+1)=array(i)
                   i=i-1;
           array(i+1)=key

Step 1: merge_sort(A,p,r)
           if p<r
           q=[(p+r)/2]
           Merge_sort(A,p,q)
           Merge_sort(A,q+1,r)
           Merge(A,p,q,r)

Step 2: merge(A,p,q,r)
           n1 = q-p+1
           n2 = r-q
           Let L(1,....n1+1] and R[1……n2+1] be new arrays

           for(i = 1 to n)
                L[i] = A[p+i - 1]

Step 3: printarray(array)
            print(array)

Step 4: Open file(data.txt)
            Put start and end time when the functions are called in the int main.
            cout the time that is taken by merge and insertion sort in the file.
```

Step 5: To plot graph to see the time complexities between merge and insertion sort, we use matplotlib in python.

```
import matplotlib.pyplot as plt
plt.plot(num,time_ins,'b',label="Insertion-sort")
plt.plot(num,time_merge,'r',label="Merge-sort")
plt.legend()
```

CONCLUSION:

In conclusion, while both Insertion Sort and Merge Sort aim to sort arrays, their time complexity characteristics differ significantly. Insertion Sort, with its quadratic time complexity, is suitable for small datasets, while Merge Sort, with its consistent O(n log n) time complexity, excels in handling larger datasets. The empirical analysis of their time complexity through experimental execution will provide valuable insights into their practical performance characteristics.

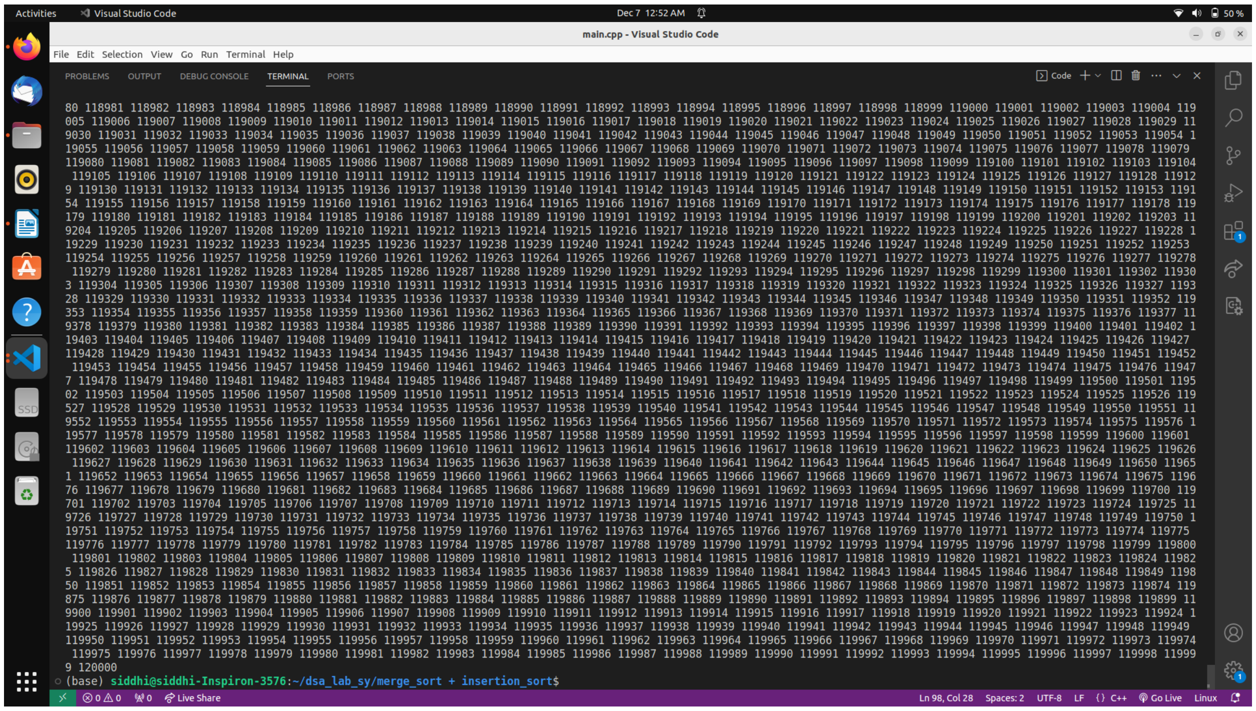
CODE:

```
#include <bits/stdc++.h>

using namespace std;

void insert_sort(int *p, int n)
{
        int i;
        //int j=0;
        int key;
        for(int j=1; j<n;j++)
        {
        key=p[j];
        i=j-1;

        while(i>=0 && p[i]>key)
        {
```

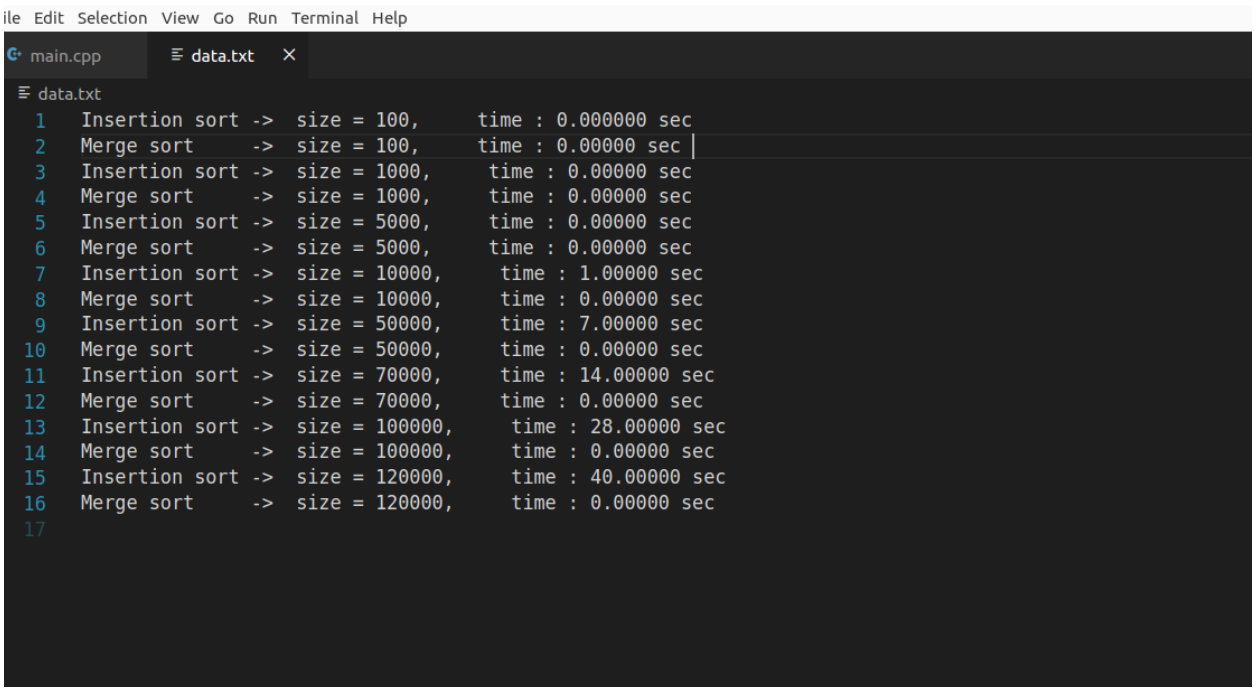
```
        p[i+1]=p[i];
        i=i-1;
        }

        p[i+1]=key;
        }
}

void merge(int *arr, int p, int q, int r) {

  int n1 = q - p + 1;
  int n2 = r - q;

  int L[n1], M[n2];

  for (int i = 0; i < n1; i++)
        L[i] = arr[p + i];
  for (int j = 0; j < n2; j++)
        M[j] = arr[q + 1 + j];


  int i, j, k;
  i = 0;
  j = 0;
  k = p;


  while (i < n1 && j < n2) {
        if (L[i] <= M[j]) {
        arr[k] = L[i];
        i++;
        } else {
        arr[k] = M[j];
        j++;
        }
        k++;
  }
  while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
  }

  while (j < n2) {
        arr[k] = M[j];
        j++;
        k++;
  }
}


void mergeSort(int *arr, int l, int r) {
  if (l < r) {

        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);


        merge(arr, l, m, r);
  }
}


void printArray(int *arr, int size) {
  for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
  cout << endl;
}
```

```
int main()
{
        ofstream file;
        file.open("data.txt");

        int input[8] = {100,1000, 5000, 10000, 50000, 70000, 100000, 120000};

        for(int i=0; i<8; i++){


        time_t start , end, start1, end1;

        // int n;
        // cout<<"Enter the size of array: ";
        // cin>>n;
        int arr[input[i]];
        int arr1[input[i]];
        //cout<<"Enter the elements"<<endl;


        for(int k =input[i]; k>0; k--)
        {
        arr[input[i]-k]=k;
        }

        for(int j =input[i]; j>0; j--)
        {
        arr1[input[i]-j]=j;
        }

        cout<<"Using insertion sort:"<<endl;
        time(&start);
        ios_base::sync_with_stdio(false);

        insert_sort(arr,input[i]);
        time(&end);
        printArray(arr, input[i]);

        double time_taken = double(end - start);
        file << "Insertion sort ->  size = "<<input[i]<<",       time : " << fixed
        << time_taken << setprecision(5);
        file << " sec " << endl;

        cout<<"Using Merge sort:"<<endl;

        time(&start1);
        ios_base::sync_with_stdio(false);

        mergeSort(arr1, 0, input[i] - 1);
        time(&end1);

        printArray(arr1, input[i]);

        double time_taken1 = double(end1 - start1);
        file << "Merge sort          ->  size = "<<input[i]<<",  time : " << fixed
        << time_taken1 << setprecision(5);
        file << " sec " << endl;
        // file<<input[i];

        }





        return 0;
}
```

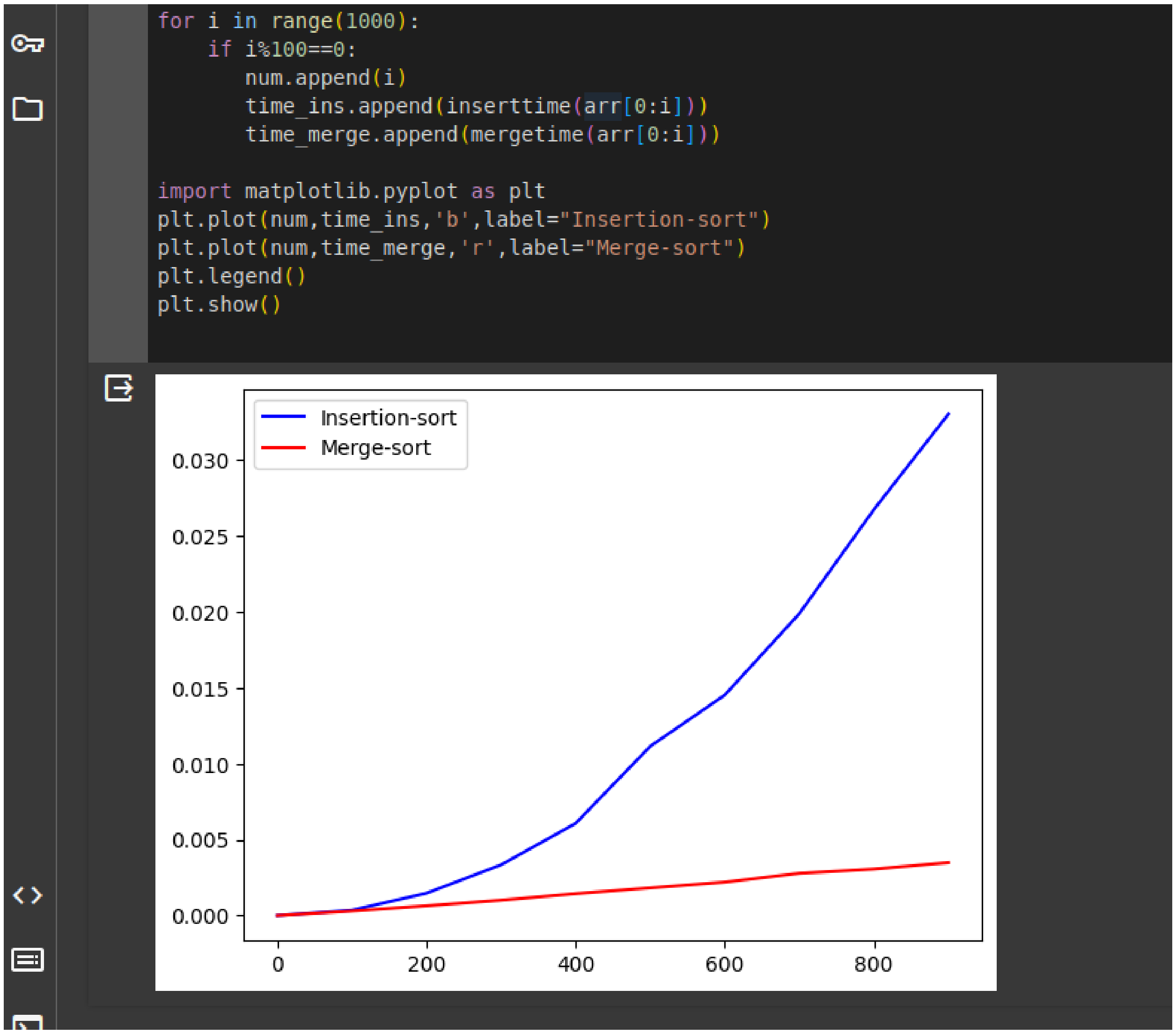
OUTPUT:



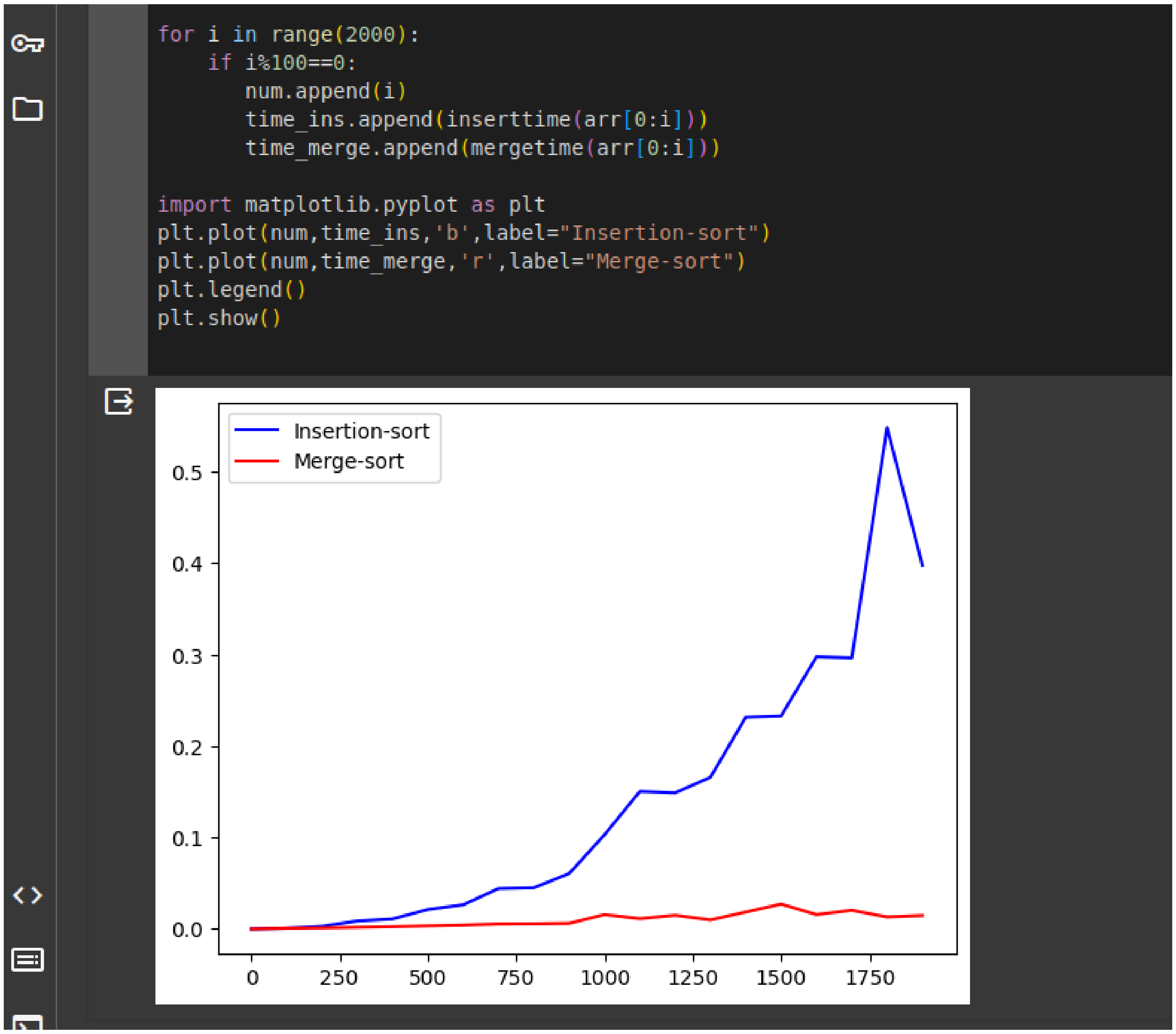
```
80 118981 118982 118983 118984 118985 118986 118987 118988 118989 118990 118991 118992 118993 118994 118995 118996 118997 118998 118999 119000 119001 119002 119003 119004 119
005 119006 119007 119008 119009 119010 119011 119012 119013 119014 119015 119016 119017 119018 119019 119020 119021 119022 119023 119024 119025 119026 119027 119028 119029 11
9030 119031 119032 119033 119034 119035 119036 119037 119038 119039 119040 119041 119042 119043 119044 119045 119046 119047 119048 119049 119050 119051 119052 119053 119054 1
19055 119056 119057 119058 119059 119060 119061 119062 119063 119064 119065 119066 119067 119068 119069 119070 119071 119072 119073 119074 119075 119076 119077 119078 119079
119080 119081 119082 119083 119084 119085 119086 119087 119088 119089 119090 119091 119092 119093 119094 119095 119096 119097 119098 119099 119100 119101 119102 119103 119104
 119105 119106 119107 119108 119109 119110 119111 119112 119113 119114 119115 119116 119117 119118 119119 119120 119121 119122 119123 119124 119125 119126 119127 119128 11912
9 119130 119131 119132 119133 119134 119135 119136 119137 119138 119139 119140 119141 119142 119143 119144 119145 119146 119147 119148 119149 119150 119151 119152 119153 1191
54 119155 119156 119157 119158 119159 119160 119161 119162 119163 119164 119165 119166 119167 119168 119169 119170 119171 119172 119173 119174 119175 119176 119177 119178 119
179 119180 119181 119182 119183 119184 119185 119186 119187 119188 119189 119190 119191 119192 119193 119194 119195 119196 119197 119198 119199 119200 119201 119202 119203 11
9204 119205 119206 119207 119208 119209 119210 119211 119212 119213 119214 119215 119216 119217 119218 119219 119220 119221 119222 119223 119224 119225 119226 119227 119228 1
19229 119230 119231 119232 119233 119234 119235 119236 119237 119238 119239 119240 119241 119242 119243 119244 119245 119246 119247 119248 119249 119250 119251 119252 119253
119254 119255 119256 119257 119258 119259 119260 119261 119262 119263 119264 119265 119266 119267 119268 119269 119270 119271 119272 119273 119274 119275 119276 119277 119278
 119279 119280 119281 119282 119283 119284 119285 119286 119287 119288 119289 119290 119291 119292 119293 119294 119295 119296 119297 119298 119299 119300 119301 119302 11930
3 119304 119305 119306 119307 119308 119309 119310 119311 119312 119313 119314 119315 119316 119317 119318 119319 119320 119321 119322 119323 119324 119325 119326 119327 1193
28 119329 119330 119331 119332 119333 119334 119335 119336 119337 119338 119339 119340 119341 119342 119343 119344 119345 119346 119347 119348 119349 119350 119351 119352 119
353 119354 119355 119356 119357 119358 119359 119360 119361 119362 119363 119364 119365 119366 119367 119368 119369 119370 119371 119372 119373 119374 119375 119376 119377 11
9378 119379 119380 119381 119382 119383 119384 119385 119386 119387 119388 119389 119390 119391 119392 119393 119394 119395 119396 119397 119398 119399 119400 119401 119402 1
19403 119404 119405 119406 119407 119408 119409 119410 119411 119412 119413 119414 119415 119416 119417 119418 119419 119420 119421 119422 119423 119424 119425 119426 119427
119428 119429 119430 119431 119432 119433 119434 119435 119436 119437 119438 119439 119440 119441 119442 119443 119444 119445 119446 119447 119448 119449 119450 119451 119452
 119453 119454 119455 119456 119457 119458 119459 119460 119461 119462 119463 119464 119465 119466 119467 119468 119469 119470 119471 119472 119473 119474 119475 119476 11947
7 119478 119479 119480 119481 119482 119483 119484 119485 119486 119487 119488 119489 119490 119491 119492 119493 119494 119495 119496 119497 119498 119499 119500 119501 1195
02 119503 119504 119505 119506 119507 119508 119509 119510 119511 119512 119513 119514 119515 119516 119517 119518 119519 119520 119521 119522 119523 119524 119525 119526 119
527 119528 119529 119530 119531 119532 119533 119534 119535 119536 119537 119538 119539 119540 119541 119542 119543 119544 119545 119546 119547 119548 119549 119550 119551 11
9552 119553 119554 119555 119556 119557 119558 119559 119560 119561 119562 119563 119564 119565 119566 119567 119568 119569 119570 119571 119572 119573 119574 119575 119576 1
19577 119578 119579 119580 119581 119582 119583 119584 119585 119586 119587 119588 119589 119590 119591 119592 119593 119594 119595 119596 119597 119598 119599 119600 119601
119602 119603 119604 119605 119606 119607 119608 119609 119610 119611 119612 119613 119614 119615 119616 119617 119618 119619 119620 119621 119622 119623 119624 119625 119626
 119627 119628 119629 119630 119631 119632 119633 119634 119635 119636 119637 119638 119639 119640 119641 119642 119643 119644 119645 119646 119647 119648 119649 119650 11965
1 119652 119653 119654 119655 119656 119657 119658 119659 119660 119661 119662 119663 119664 119665 119666 119667 119668 119669 119670 119671 119672 119673 119674 119675 1196
76 119677 119678 119679 119680 119681 119682 119683 119684 119685 119686 119687 119688 119689 119690 119691 119692 119693 119694 119695 119696 119697 119698 119699 119700 119
701 119702 119703 119704 119705 119706 119707 119708 119709 119710 119711 119712 119713 119714 119715 119716 119717 119718 119719 119720 119721 119722 119723 119724 119725 11
9726 119727 119728 119729 119730 119731 119732 119733 119734 119735 119736 119737 119738 119739 119740 119741 119742 119743 119744 119745 119746 119747 119748 119749 119750 1
19751 119752 119753 119754 119755 119756 119757 119758 119759 119760 119761 119762 119763 119764 119765 119766 119767 119768 119769 119770 119771 119772 119773 119774 119775
119776 119777 119778 119779 119780 119781 119782 119783 119784 119785 119786 119787 119788 119789 119790 119791 119792 119793 119794 119795 119796 119797 119798 119799 119800
 119801 119802 119803 119804 119805 119806 119807 119808 119809 119810 119811 119812 119813 119814 119815 119816 119817 119818 119819 119820 119821 119822 119823 119824 11982
5 119826 119827 119828 119829 119830 119831 119832 119833 119834 119835 119836 119837 119838 119839 119840 119841 119842 119843 119844 119845 119846 119847 119848 119849 1198
50 119851 119852 119853 119854 119855 119856 119857 119858 119859 119860 119861 119862 119863 119864 119865 119866 119867 119868 119869 119870 119871 119872 119873 119874 119
875 119876 119877 119878 119879 119880 119881 119882 119883 119884 119885 119886 119887 119888 119889 119890 119891 119892 119893 119894 119895 119896 119897 119898 119899 11
9900 119901 119902 119903 119904 119905 119906 119907 119908 119909 119910 119911 119912 119913 119914 119915 119916 119917 119918 119919 119920 119921 119922 119923 119924 1
19925 119926 119927 119928 119929 119930 119931 119932 119933 119934 119935 119936 119937 119938 119939 119940 119941 119942 119943 119944 119945 119946 119947 119948 119949
119950 119951 119952 119953 119954 119955 119956 119957 119958 119959 119960 119961 119962 119963 119964 119965 119966 119967 119968 119969 119970 119971 119972 119973 119974
 119975 119976 119977 119978 119979 119980 119981 119982 119983 119984 119985 119986 119987 119988 119989 119990 119991 119992 119993 119994 119995 119996 119997 119998 11999
9 120000
(base) siddhi@siddhi-Inspiron-3576:~/dsa_lab_sy/merge_sort + insertion_sort$
```



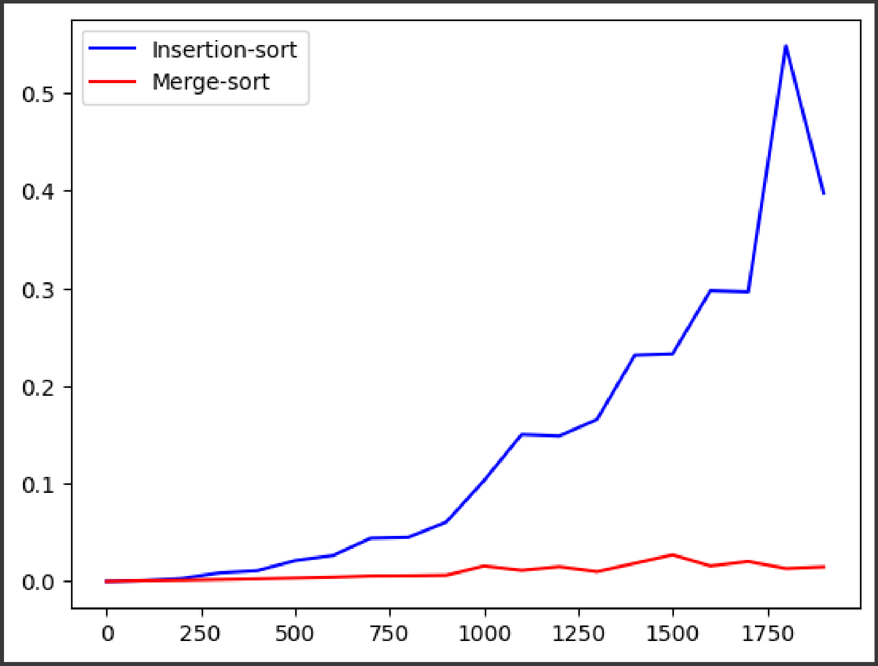
```
Insertion sort -> size = 100,     time : 0.000000 sec
Merge sort     -> size = 100,     time : 0.00000 sec
Insertion sort -> size = 1000,     time : 0.00000 sec
Merge sort     -> size = 1000,     time : 0.00000 sec
Insertion sort -> size = 5000,     time : 0.00000 sec
Merge sort     -> size = 5000,     time : 0.00000 sec
Insertion sort -> size = 10000,     time : 1.00000 sec
Merge sort     -> size = 10000,     time : 0.00000 sec
Insertion sort -> size = 50000,     time : 7.00000 sec
Merge sort     -> size = 50000,     time : 0.00000 sec
Insertion sort -> size = 70000,     time : 14.00000 sec
Merge sort     -> size = 70000,     time : 0.00000 sec
Insertion sort -> size = 100000,     time : 28.00000 sec
Merge sort     -> size = 100000,     time : 0.00000 sec
Insertion sort -> size = 120000,     time : 40.00000 sec
Merge sort     -> size = 120000,     time : 0.00000 sec
```

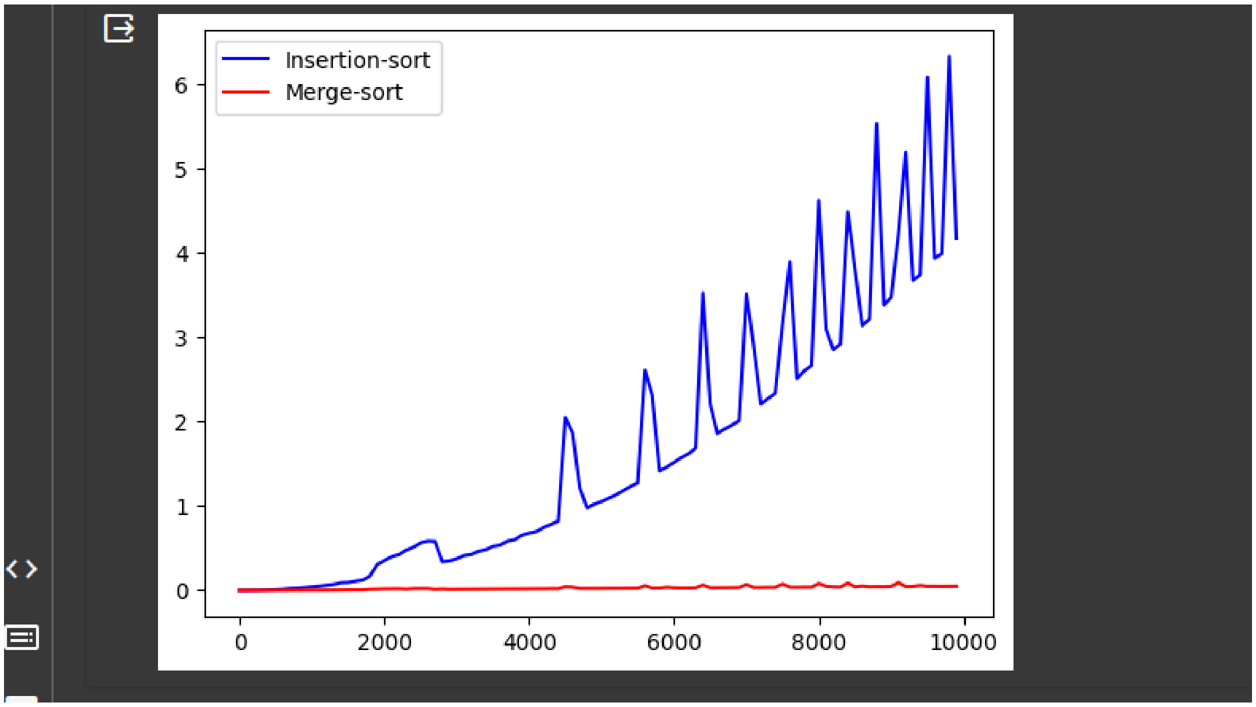


```
for i in range(1000):
    if i%100==0:
       num.append(i)
       time_ins.append(inserttime(arr[0:i]))
       time_merge.append(mergetime(arr[0:i]))

import matplotlib.pyplot as plt
plt.plot(num,time_ins,'b',label="Insertion-sort")
plt.plot(num,time_merge,'r',label="Merge-sort")
plt.legend()
plt.show()
```



```
for i in range(2000):
    if i%100==0:
       num.append(i)
       time_ins.append(inserttime(arr[0:i]))
       time_merge.append(mergetime(arr[0:i]))

import matplotlib.pyplot as plt
plt.plot(num,time_ins,'b',label="Insertion-sort")
plt.plot(num,time_merge,'r',label="Merge-sort")
plt.legend()
plt.show()
```




Insertion-sort
Merge-sort
0
1
2
3
4
5
6
0
2000
4000
6000
8000
10000