DIGITAL ANALYSIS AND ALGORITHM

EXPERIMENT - 01

NAME :- MANTHAN AYALWAR

UID :- 2021700003

BATCH :- D1

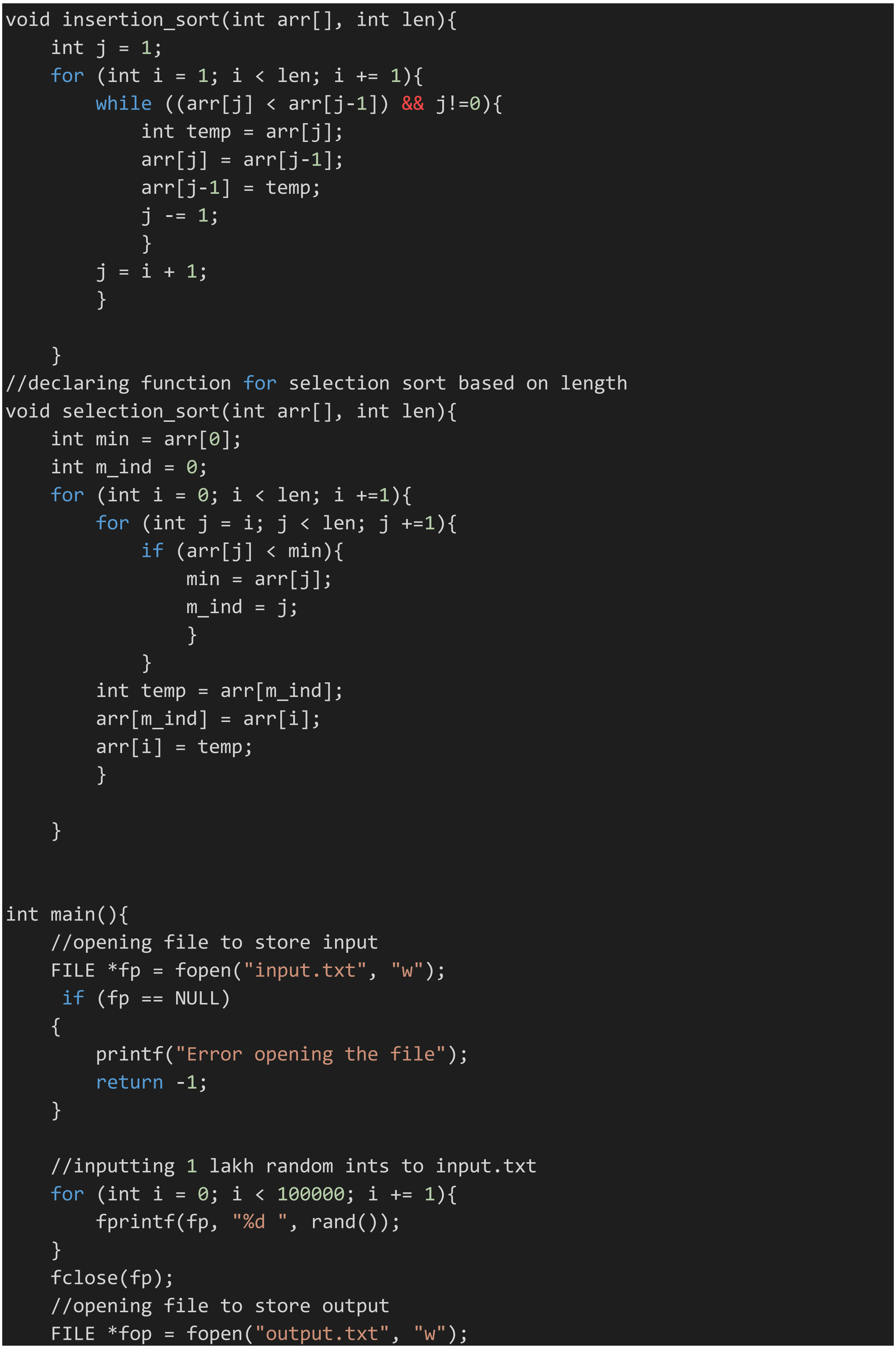
PART – 1B

Aim :- Experiment on finding the running time of an algorithm.

Definition & Assumptions – For this experiment, you need to implement two sorting algorithms namely .Insertion and Selection sort methods. Compare these algorithms based on time and space complexity. Time required to sorting algorithms can be performed using high_resolution_clock::now() under namespace std::chrono. You have to generate1,00,000 integer numbers using C/C++ Rand function and save them in a text file. Both the sorting algorithms uses these 1,00,000 integer numbers as input as follows. Each sorting algorithm sorts a block of 100 integers numbers with array indexes numbers A[0..99], A[0..199], A[0..299],..., A[0..99999]. You need to use high_resolution_clock::now() function to find the time required for 100, 200, 300.... 100000 integer numbers. Finally, compare two algorithms namely Insertion and Selection by plotting the time required to sort 100000 integers using LibreOffice Calc/MS Excel. The x-axis of 2-D plot represents the block no. of 1000 blocks. The y-axis of 2-D plot representsthe tunning time to sort 1000 blocks of 100,200,300,...,100000 integer numbers. Note – You have to use C/C++ file processing functions for reading and writing randomly generated 100000 integer numbers.

Code :-

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
//declaring function for insertion sort based on length
```



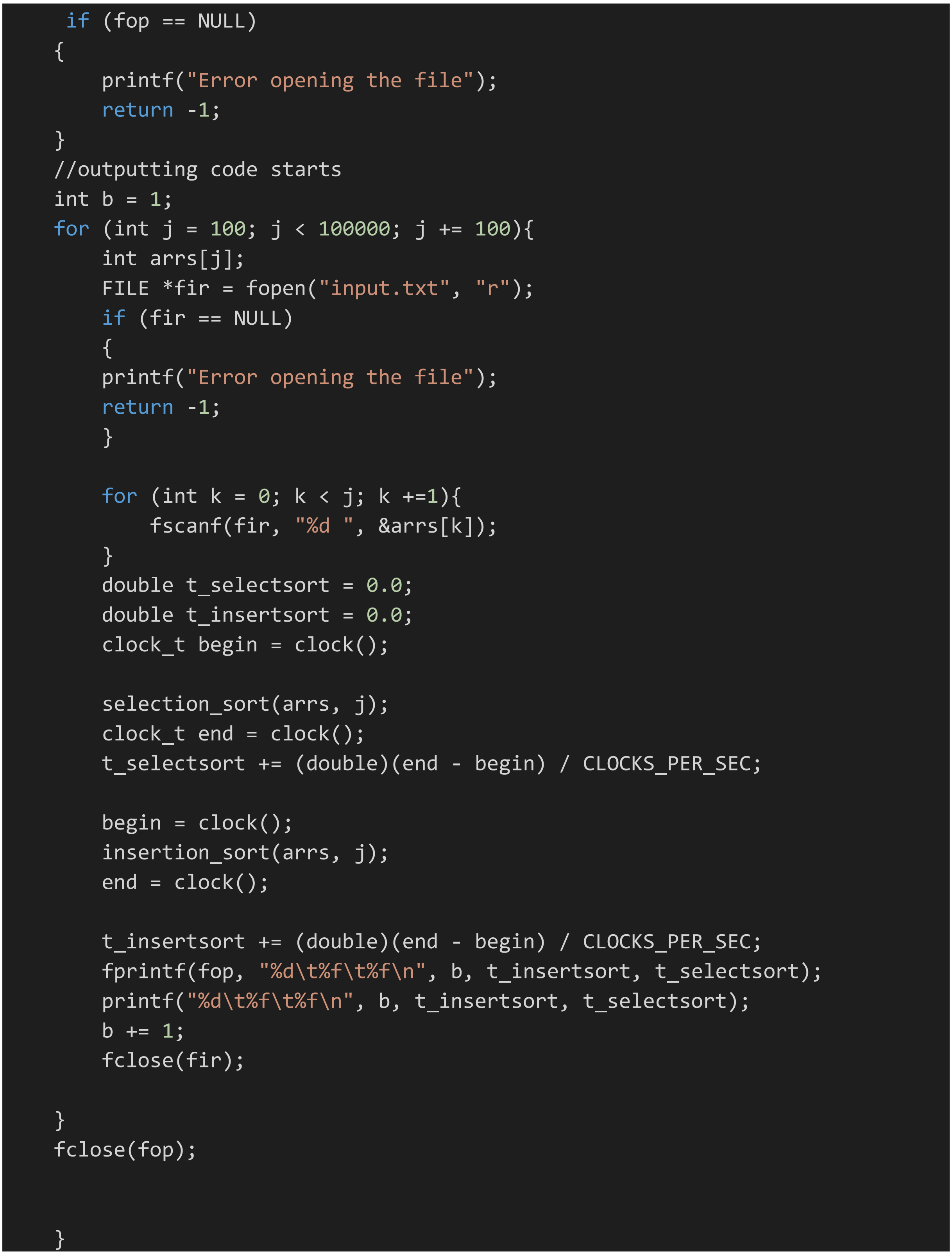
```
void insertion_sort(int arr[], int len){
    int j = 1;
    for (int i = 1; i < len; i += 1){
        while ((arr[j] < arr[j-1]) && j!=0){
            int temp = arr[j];
            arr[j] = arr[j-1];
            arr[j-1] = temp;
            j -= 1;
            }
        j = i + 1;
        }

    }
//declaring function for selection sort based on length
void selection_sort(int arr[], int len){
    int min = arr[0];
    int m_ind = 0;
    for (int i = 0; i < len; i +=1){
        for (int j = i; j < len; j +=1){
            if (arr[j] < min){
                min = arr[j];
                m_ind = j;
                }
            }
        int temp = arr[m_ind];
        arr[m_ind] = arr[i];
        arr[i] = temp;
        }

    }


int main(){
    //opening file to store input
    FILE *fp = fopen("input.txt", "w");
     if (fp == NULL)
    {
        printf("Error opening the file");
        return -1;
    }

    //inputting 1 lakh random ints to input.txt
    for (int i = 0; i < 100000; i += 1){
        fprintf(fp, "%d ", rand());
    }
    fclose(fp);
    //opening file to store output
    FILE *fop = fopen("output.txt", "w");
```



```
     if (fop == NULL)
    {
        printf("Error opening the file");
        return -1;
    }
    //outputting code starts
    int b = 1;
    for (int j = 100; j < 100000; j += 100){
        int arrs[j];
        FILE *fir = fopen("input.txt", "r");
        if (fir == NULL)
        {
        printf("Error opening the file");
        return -1;
        }

        for (int k = 0; k < j; k +=1){
            fscanf(fir, "%d ", &arrs[k]);
        }
        double t_selectsort = 0.0;
        double t_insertsort = 0.0;
        clock_t begin = clock();

        selection_sort(arrs, j);
        clock_t end = clock();
        t_selectsort += (double)(end - begin) / CLOCKS_PER_SEC;

        begin = clock();
        insertion_sort(arrs, j);
        end = clock();

        t_insertsort += (double)(end - begin) / CLOCKS_PER_SEC;
        fprintf(fop, "%d\t%f\t%f\n", b, t_insertsort, t_selectsort);
        printf("%d\t%f\t%f\n", b, t_insertsort, t_selectsort);
        b += 1;
        fclose(fir);

    }
    fclose(fop);


    }
```

Input :-

```
1804289383 846930886 1681692777 1714636915 1957747793 424238335 719885386 1649760492 596516649 1189641421 1025202362 1350490027 783368690
1102520059 2044897763 1967513926 1365180540 1540383426 304089172 1303455736 35005211 521595368 294702567 1726956429 336465782 861021530
278722862 233665123 2145174067 468703135 1101513929 1801979802 1315634022 635723058 1369133069 1125898167 1059961393 2089018456 628175011
1656478042 1131176229 1653377373 859484421 1914544919 608413784 756898537 1734575198 1973594324 149798315 2038664370 1129566413 184803526
412776091 1424268980 1911759956 749241873 137806862 42999170 982906996 135497281 511702305 2084420925 1937477084 1827336327 572660336
1159126505 805750846 1632621729 1100661313 1433925857 1141616124 84353895 939819582 2001100545 1998898814 1548233367 610515434 1585990364
1374344043 760313750 1477171087 356426808 945117276 1889947178 1780695788 709393584 491705403 1918502651 752392754 1474612399 2053999932
1264095060 1411549676 1843993368 943947739 1984210012 855636226 1749698586 1469348094 1956297539 1036140795 463480570 2040651434 1975960378
317097467 1892066601 1376710097 927612902 1330573317 603570492 1687926652 660260756 959997301 485560280 402724286 593209441 1194953865
894429689 364228444 1947346619 221558440 270744729 1063958031 1633108117 2114738097 2007905771 1469834481 822890675 1610120709 791698927
631704567 498777856 1255179497 524872353 327254586 1572276965 269455306 1703964683 352406219 1600028624 160051528 2040332871 112805732
1120048829 378409503 515530019 1713258270 1573363368 1409959708 2077486715 1373226340 1631518149 200747796 289700723 1117142618 168002245
150122846 439493451 990892921 1760243555 1231192379 1622597488 111537764 338888228 2147469841 438792350 1911165193 269441500 2142757034
116087764 1869470124 155324914 8936987 1982275856 1275373743 387346491 350322227 841148365 1960709859 1760281936 771151432 1186452551
1244316437 971899228 1476153275 213975407 1139901474 1626276121 653468858 2130794395 1239036029 1884661237 1605908235 1350573793 76065818
1605894428 1789366143 1987231011 1875335928 1784639529 2103318776 1597322404 1939964443 2112255763 1432114613 1067854538 352118606
1782436840 1909002904 165344818 1395235128 532670688 1351797369 492067917 1504569917 680466996 706043324 496987743 159259470 1359512183
480298490 1398295499 1096689772 2086206725 601385644 1172755590 1544617505 243268139 1012502954 1272469786 2027907669 968338082 722308542
1820388464 933110197 6939507 740759355 1285228804 1789376348 502278611 1450573622 1037127828 1034949299 654887343 1529195746 392035568
1335354340 87755422 889023311 1494613810 1447267605 1369321801 745425661 396473730 1308044878 1346811305 1569229320 705178736 1590079444
434248626 1977648522 1470503465 1402586708 552473416 1143408282 188213258 559412924 1884167637 1473442062 201305624 238962600 776532036
1238433452 1273911899 1431419379 620145550 1665947468 619290071 707900973 407487131 2113903881 7684930 1776808933 711845894 404158660
937370163 2058657199 1973387981 1642548899 1501252996 260152959 1472713773 824272813 1662739668 2025187190 1967681095 1850952926 437116466
1704365084 1176911340 638422090 1943327684 1953443376 1876855542 1069755936 1237379107 349517445 588219756 1856669179 1057418418 995706887
1823089412 1065103348 625032172 387451659 1469262009 1562402336 298625210 1295166342 1057467587 1799878206 1555319301 382697713 476667372
1070575321 260401255 296864819 774044599 697517721 2001229904 1950955939 1335939811 1797073940 1756915667 1065311705 719346228 846811127
1414829150 1307565984 555996658 324763920 155789224 231602422 1389867269 780821396 619054081 711645630 195740084 917679292 2006811972
1253207672 570073850 1414647625 1635905385 1046741222 337739299 1896306640 1343606042 1111783898 446340713 1197352298 915256190 1782280524
846942590 524688209 700108581 1566288819 1371499336 2114937732 726371155 1927495994 292218004 882160379 11614769 1682085273 1662981776
630668850 246247255 1858721860 1548348142 105575579 964445884 2118421993 1520223205 452867621 1017679567 1857962504 201690613 213801961
822262754 648031326 1411154259 1737518944 282828202 110613202 114723506 982936784 1676902021 1486222842 950390868 255789528 1266235189
1242608872 1137949908 1277849958 777210498 653448036 1908518808 1023457753 364686248 1309383303 1129033333 1329132133 1280321648 501772890
1781999754 150517567 212251746 1983690368 364319529 1034514500 484238046 1775473788 624549797 767066249 1886086990 739273303 1750003033
1415505363 78012497 552910253 1671294892 1344247686 1795519125 661761152 474613996 425245975 1315209188 235649157 1448703729 1679895436
1545032460 430253414 861543921 677870460 932026304 496060028 828388027 1144278050 332266748 1192707556 31308902 816504794 820697697
655858699 1583571043 559301039 1395132002 1186090428 1974806403 1473144500 1739000681 1498617647 669908538 1387036159 12895151 1144522535
1812282134 1328104339 1380171692 1113502215 860516127 777720504 1543755629 1722060049 1455590964 328298285 70636429 136495343 1472576335
402903177 1329202900 1503885238 1219407971 2416949 12260289 655495367 561717988 1407392292 1841585795 389040743 733053144 1433102829
```

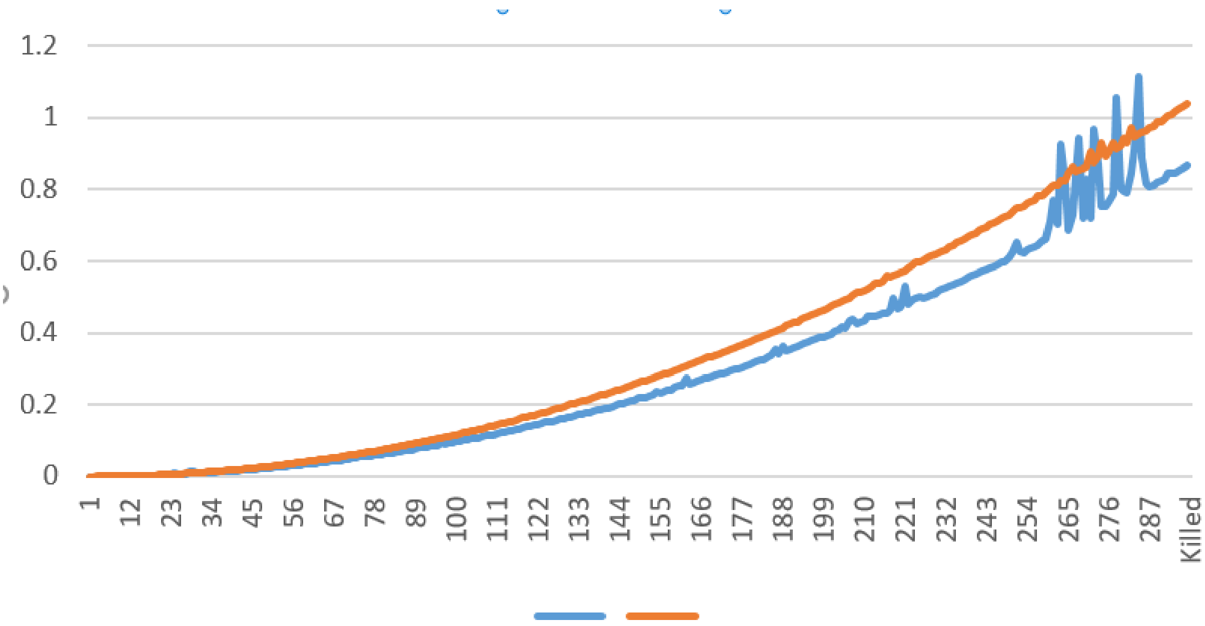
Ouput :-

```
1       0.000010     0.000015
2       0.000061     0.000051
3       0.000089     0.000109
4       0.000162     0.000196
5       0.000258     0.000301
6       0.000368     0.000428
7       0.000501     0.000592
8       0.001030     0.000782
9       0.001265     0.000989
10      0.001358     0.001183
11      0.002210     0.001465
12      0.002643     0.001756
13      0.003096     0.002025
14      0.003618     0.002360
15      0.002995     0.002723
16      0.002606     0.003077
17      0.003179     0.003433
```

18	0.003397	0.003825
19	0.003947	0.004272
20	0.004100	0.004768
21	0.004451	0.005255
22	0.005205	0.005712
23	0.005581	0.006229
24	0.010399	0.006928
25	0.006627	0.007561

……..

Graph :-



Obsrvation :- In this graph we observed for small number of input size the time taken by selection sort and insertion sort is comparable . But as the size increases insertion sort outperform selection sort .The abrupt spike in graph of insertion graph is due to the poor memory management of machine running the program.

Conclusion :- From this experiment ,i learned the concept of insertion sort and selection sort. Firstly, we have generated 1,00,000 integer numbers using C/C++ Rand function and save them in a text file . And then by using these file ,each sorting algorithm sorts a block of 100 integers numbers with array indexes numbers