

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 generate 1,00,000 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 represents the tuning 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.

Theory :-

Insertion sort is a simple sorting algorithm that works similar to 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.

Algorithm:

To sort an array of size n in ascending order:

- Iterate from arr[1] to arr[n] over the array.
- Compare the current element (key) to its predecessor.
- If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

The [selection sort](#) algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from the unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

- The subarray is already sorted.
- The remaining subarray is unsorted.

In every iteration of the selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

Code :-

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

double populate(int a[], int b[], int n) {
    clock_t start, end;
    double cpu_time_used;
    start = clock();
    for(int i = 0; i < n; i++)
    {
        int r = rand();
        a[i] = b[i] = r;
    }
    end = clock();
    FILE *fp = fopen("./random10.txt", "w+");
    if(!fp) {
        printf("Error opening file\n");
        return -1;
    }
    for(int i = 0; i < n; i++) {
        fprintf(fp, "%d\n", a[i]);
    }
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    return cpu_time_used;
}
```

```

void swap(int *x, int *y) {
    int t = *x;
    *x = *y;
    *y = t;
}

double selection(int a[], int n) {
    FILE *fp = fopen("./selection10.csv", "w+");
    // printf("File opened\n");
    double totalTime = 0;
    if(!fp) {
        printf("Error opening file\n");
        return -1;
    }
    fprintf(fp, "n, time\n");
    for (int i = 9999; i <= n; i+=10000)
    {
        // printf("%d\n", i);
        clock_t start, end;
        double cpu_time_used;
        start = clock();
        for(int j = 0; j < i; j++) {
            int min = j;
            for(int k = j+1; k <= i; k++) {
                if(a[k] < a[min]) {
                    min = k;
                }
            }
            swap(&a[j], &a[min]);
        }
        end = clock();
        cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
        totalTime += cpu_time_used;
        fprintf(fp, "%d, %f\n", i+1, cpu_time_used);
        printf("Sorted from 0 to %d in %.2fs\n", i, cpu_time_used);

        // for(int z = 0; z < i; z++) {
        //     printf("%d\n", a[z]);
        // }
        // getchar();
    }
    fclose(fp);
    fp = fopen("./selection10.txt", "w+");
    for(int i = 0; i < n; i++) {
        fprintf(fp, "%d\n", a[i]);
    }
    fclose(fp);
    return totalTime;
}

```

```

}

double insertion(int a[], int n) {
    FILE *fp = fopen("./insertion10.csv", "w+");
    // printf("File opened\n");
    double totalTime = 0;
    if(!fp) {
        printf("Error opening file\n");
        return -1;
    }
    fprintf(fp, "n, time\n");
    //insertion sort
    //first sort from 0 to 100 the 0 to 200 and so on upto n
    for (int i = 9999; i <= n; i+=10000)
    {
        // printf("%d\n", i);
        clock_t start, end;
        double cpu_time_used;
        start = clock();
        for(int j = 1; j <= i; j++)
        {
            int k = j;
            // printf("%d\n", i);
            while(k > 0 && a[k] < a[k-1])
            {
                swap(&a[k], &a[k-1]);
                k--;
            }
        }
        end = clock();
        cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
        totalTime += cpu_time_used;
        fprintf(fp, "%d, %f\n", i+1, cpu_time_used);

        printf("Sorted from 0 to %d in %.2fs\n", i, cpu_time_used);

        // for(int z = 0; z < i; z++) {
        //     printf("%d\n", a[z]);
        // }
        // getchar();
    }
    fclose(fp);
    fp = fopen("./insertion10.txt", "w+");
    for(int i = 0; i < n; i++) {
        fprintf(fp, "%d\n", a[i]);
    }
    fclose(fp);
    return totalTime;
}

```

```

}

int main()
{
    int n = 100000;
    int a[n],b[n];
    double timeToPopulate = populate(a, b, n);
    printf("Time taken to populate: %f\nSorting...\n", timeToPopulate);
    //first sort from 0 to 100 the 0 to 200 and so on upto n
    double timeToSortI = insertion(a, n);
    double timeToSortS = selection(b, n);
    printf("Array sorted by insertion sort in %.2f\n", timeToSortI);
    printf("Array sorted by selection sort in %.2f\n", timeToSortS);
    printf("Total time taken to sort: %f\n", timeToSortI + timeToSortS);
    return 0;
}

```

Input :-

```

1804289283 846930886 1681692777 1714636915 1057747793 424238335 719885386 1649760492 596516649 1189641421 1025202362 1350490027 783368690
1102520059 2044897765 1967913926 1365180540 1540389426 304089172 1303455736 35005211 521595366 294702567 172656429 336465782 861031530
278722862 233669123 2145174067 466703135 1101519829 1801979802 1315634022 835723058 1369133069 1125998167 1059961393 1089018456 628175011
1850478942 1131176229 1653377373 859484421 1914544919 608413784 756898537 1734575196 1973594324 145788325 2038664370 1125566413 184803526
412776091 1424268880 1911759956 749241873 137806862 42999170 882906996 135497281 511702305 2084428525 1937477084 1827336327 572660336
1155126905 805750846 1632621729 1100661313 1433925957 1141616124 84353895 939819582 2001100545 1998899814 1548233367 410515434 1585990364
1374344043 780313750 1477171087 356426808 945117276 1889947178 1790695788 706393584 491705403 1918502651 752392754 1474612399 2053999932
1264095060 1411546676 1843993368 943947739 1984210012 855636226 1740828586 1469348294 1956227538 1056140795 463480570 2040851434 1975960378
317097487 1992068601 2376110097 927812902 1330573317 803570492 1887926652 660260756 959967301 485560280 402724286 593209441 1194953885
894429689 364228444 1947346519 221556440 270744729 1063958031 1633108117 2114738097 2007905771 1469834481 822890675 1610120709 791698927
631704567 498777856 1255179497 524872353 327254566 1672275965 269455306 1703966689 352406219 1600028624 160051528 2040332871 112805732
1120048829 37609503 515530019 1713258270 157383368 1409559708 2077486713 1373226940 1635159149 200747796 289700723 1117142618 168002243
150122846 438453451 980892921 1760243955 1231192379 1622597488 111537764 388888228 2147469841 438752350 191165193 268461500 2162757034
116087764 1889470124 155324814 8936987 1982275856 1275373743 387346491 350322227 841148265 1960709859 1760281936 771151432 1186432551
1244315437 971899228 1476153275 213975407 1139901474 1626276121 653468858 2130794395 1239036029 1884661237 1605908235 1350573793 76065818
1605894428 1799386145 1987231011 1875335028 1784639529 2103318776 1597322404 1939964443 2112255763 1432114613 1067854538 382118806
1782436840 1900002904 185344818 1395235128 532670688 1351797396 492067917 1504569947 680468996 706043324 496887743 158259470 1359512183
480298490 1396295499 1096689772 2086206725 601385644 1172755590 1544617505 243268139 1012502954 1272469786 2027907669 968338082 722308542
1820388464 933110197 6939507 740759355 1285228806 1789376348 502278611 1450573622 1037127828 1034949289 654887343 1629195746 392035588
1335354340 87755422 889023311 1494613810 1447267605 1365921801 745425681 396473730 1308044878 1346811305 1785229320 705178736 1590079444
436248626 1977648522 14705031465 1402586708 552473416 1143408282 188213258 559412924 1884167637 1473442062 201305624 238962600 776532036
1238433432 1273911899 1431419379 620143550 1665947466 619290071 707900973 407487131 2113903881 7684930 1776808933 711845854 404158660
937370163 2058657199 1973387981 1842548899 1501252996 260152959 1472713773 824272813 1662739668 2025187190 1967681095 1850952926 437118466
1784365084 1176911340 638422090 1943327684 1953443376 1876855542 1069755936 1237379107 349517445 588218756 1856669179 1057418418 995706887
1823089412 1065103348 625032172 387451659 1469262009 1562402336 298625210 1295166342 1057487587 1798876206 1555319301 382697713 476667372
1070575321 260401255 296864819 774044599 697517721 2001229904 1950955939 1335839811 1797073940 1756915667 1065311705 719346228 846811127
1414823150 1307545884 555996658 324763320 155789234 231602422 1389867269 780621396 619054081 711645630 195740084 517675292 2008011973
1253207872 570073850 1414647625 1835905395 10467461222 337739299 1859306640 1343606042 1111783598 440340713 1197352298 915256190 1782280524
846942350 524688205 700108581 1566288819 1371499336 2114937732 728371155 1927459594 292218004 882160379 11614768 1682085273 1662981778
630688890 246247255 1898721860 1546348142 105575879 964445884 2118421593 1520223205 452867621 1017679567 1857962504 201690613 213801961
822262754 648031326 1411154259 1737518944 222828202 110613202 114723506 982936784 1676902021 1488222842 950390868 255789528 1266235189
1242608872 1137949908 1277849958 777210498 653448036 1908518808 1023457753 364686248 1309383303 1129033333 1329132133 1280321648 501772890
1781999754 150517567 212251746 1983690368 164319529 1034514500 484238046 1775473788 624549797 767066249 1866086990 739273303 1750003033
1415505369 78012497 552910253 1471294892 1344247886 1795519125 661781152 474613998 425245975 1315209188 235649157 1448703729 1679895436
1545032480 430233414 861543921 477870460 932026304 496060028 829388027 1146278050 232266748 1192707556 71308962 816504799 820697697
855896999 1583571043 555581038 1395132002 1196090428 1574806403 1473144500 1739000681 1498817647 689908538 1387036155 12895151 1144522533
1812282134 1328104339 1380171692 1113502215 960516127 777270504 1542755629 1722660049 1455590964 328298285 70436429 136455343 1472576335
402903177 1328202900 1503885228 1218407971 2414949 12260289 655455367 561717888 1407392282 1841585795 389040743 73305144 1433102825

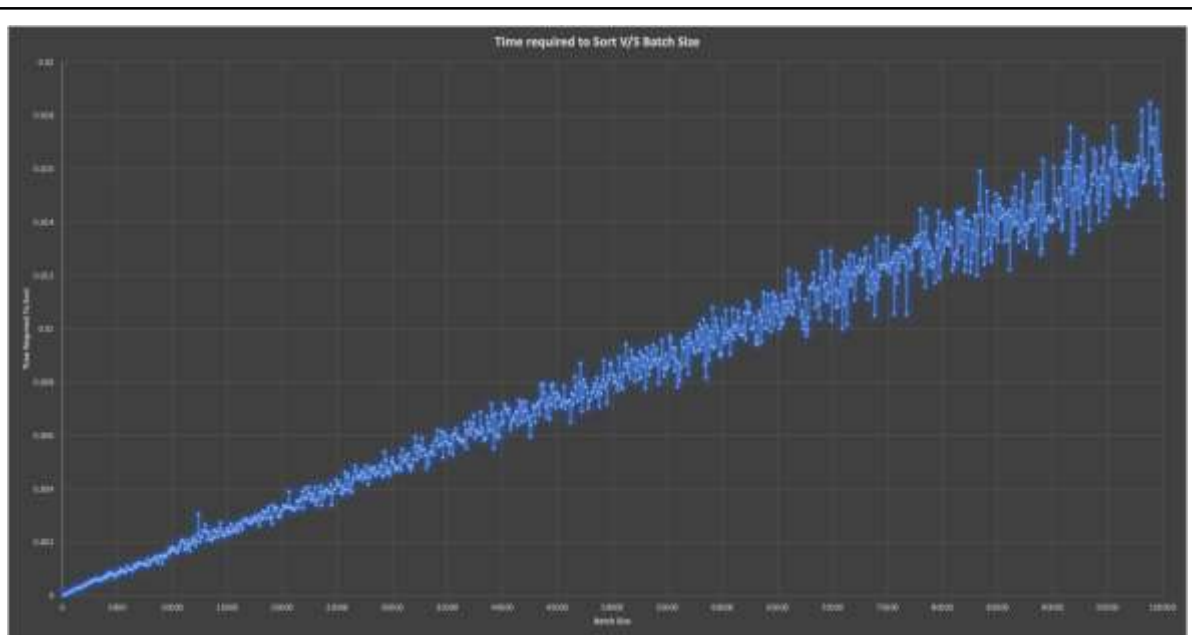
```

Ouput :-

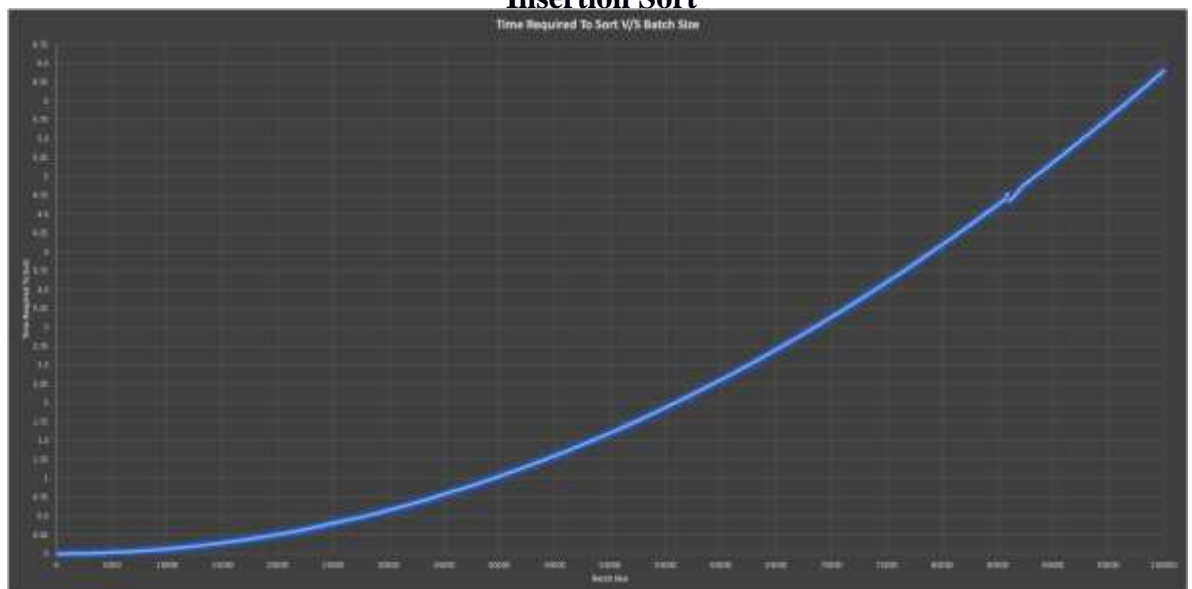
```
Sorted from 0 to 98299 in 6.17s
Sorted from 0 to 98399 in 6.18s
Sorted from 0 to 98499 in 6.19s
Sorted from 0 to 98599 in 6.21s
Sorted from 0 to 98699 in 6.22s
Sorted from 0 to 98799 in 6.23s
Sorted from 0 to 98899 in 6.24s
Sorted from 0 to 98999 in 6.26s
Sorted from 0 to 99099 in 6.27s
Sorted from 0 to 99199 in 6.28s
Sorted from 0 to 99299 in 6.29s
Sorted from 0 to 99399 in 6.30s
Sorted from 0 to 99499 in 6.39s
Sorted from 0 to 99599 in 6.37s
Sorted from 0 to 99699 in 6.40s
Sorted from 0 to 99799 in 6.55s
Sorted from 0 to 99899 in 6.42s
Sorted from 0 to 99999 in 6.43s
Array sorted by insertion sort in 8.22
Array sorted by selection sort in 2137.75
Total time taken to sort: 2145.973837
* Terminal will be reused by tasks, press any key to close it.
```

.....

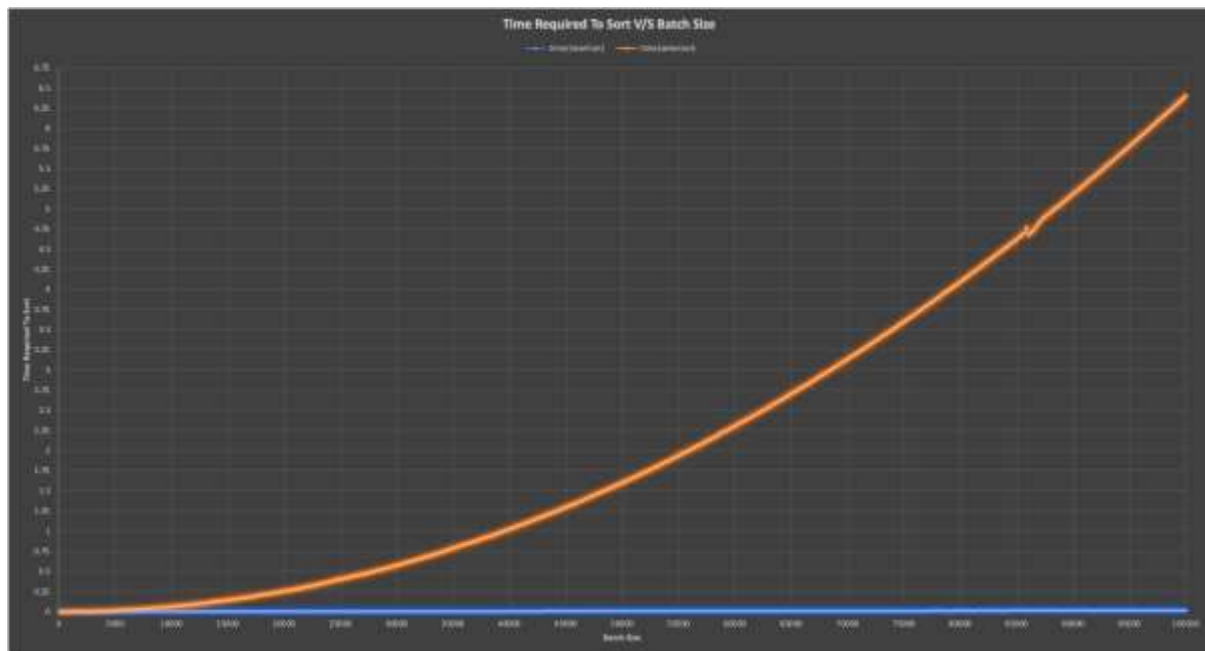
Graph :-



Insertion Sort



Selection Sort



Obsrvation :-

Factors affecting run time of sorting algorithm:-

1. The order of numbers to be sorted plays an important role in the running time.
2. CPU Utilization - If CPU is already utilized by some other processes then the running time of the algorithm will increase.
3. Recursion can cause a lot of overhead, if the structure of function permits, then use *tail recursion*, to avoid the stack overhead.
4. Choice of Language- C is faster when compared to Java because it is a low level language. Thus many network security applications are written in C, where speed matters.

Main memory usage- Higher main memory usage can lead to slower processing.

Conclusion :- Successfully understood calculation of running time of algorithms by implementing insertion and selection sort in C. Also observed the two sorting methods for 1,00,000 randomly generated numbers graphically. The array of random numbers used for sorting was the same for both sorting methods yet insertion sort on the same computer takes much less time when compared to selection sort. This can be confirmed from the graph as well as the run time observed