

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.

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 :-

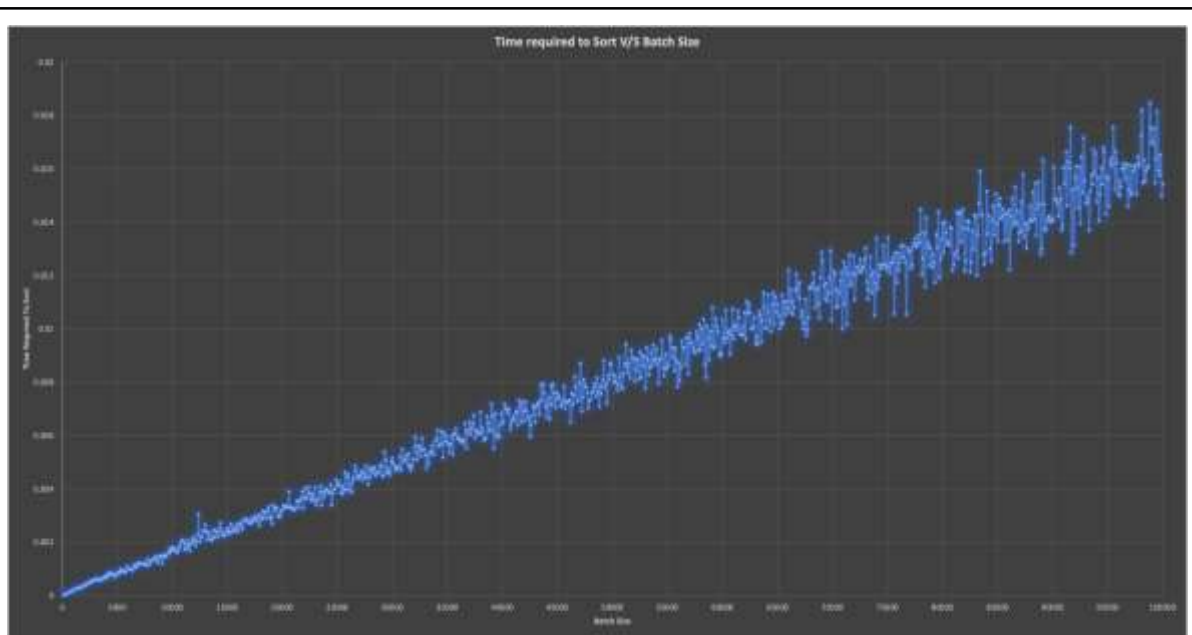
```
1804289283 846930886 1681692777 1714698915 1957747793 424238335 719983386 1640760492 596516649 1189661421 1025202363 1350490027 783368690
1102520059 2044897765 1967513926 1365180540 1540389436 304089172 1303455736 35005211 521568368 294702567 1726956429 336465782 861031530
278722862 233669123 2145174067 466703135 1101513929 1801979802 1315634022 835723058 1369133069 1125698167 1059961393 2089018456 628175011
1656478042 1131176229 1053377373 859484421 1914544919 608413784 756895937 1734575196 1973594324 145798325 2038666370 1125666413 184803526
412776051 1424268980 1511759956 749241873 137806662 42599170 882506956 135497281 511702305 2084420525 1937477084 1827338327 572660336
1155126505 805750846 1632621729 1100661313 1433925857 1142616124 84353895 939815582 2001100545 1998898814 1548233367 610515434 1585590964
1374344043 760313750 147771087 356426808 945117276 1889947178 1780695788 7069393584 491705403 1918502661 752392754 1474612395 2053999932
1264005060 1411549676 1843993368 943947739 1984210012 855636226 1740698586 1469348094 1956297539 1036140795 463480570 2040851434 1975660378
317097467 1890266601 1376710097 927612902 1330573317 603570492 1687926652 660260756 959097301 485560280 402724286 593209441 1194953865
894429689 364228444 1947346619 221556440 270744729 1063958031 1633108117 2114738097 2007905771 1469834481 822890675 1450120709 791698927
631704567 498777856 1255179497 524872353 327254566 1572274965 269455306 1703964683 352406219 1600028624 160051528 2040332871 112805732
1120048929 378609503 515500019 1713258270 1573383368 1409859708 2077486713 1373226340 1435158149 200747796 289700723 1117142618 168002243
150122846 439493451 990895921 1760243955 1231192379 1622597488 111537764 338888228 2147469641 438752350 191165193 268461500 2142757034
116087764 1889470124 155324814 8936987 1982275856 1275373743 387346491 350322227 841148365 1960709859 1760281934 771151432 1186432551
1244316437 971896228 1476153275 213975407 1139901474 1626276121 653468858 2150794395 1219036029 1894661237 1605908235 1350573790 76085818
1605894428 1799386145 1987231011 1875335028 1784639529 2103318776 1597322404 1939964443 2112255763 1432114613 1067854538 352118806
1782436840 1509002904 145344818 1395235128 532670688 1351797369 492067017 1504569917 680468996 706043324 498987743 154250470 1359512183
480298490 1398295499 1096689772 2088206725 601385644 1172755950 1544617905 243268139 1012502954 1273469786 2027907669 968338882 722308542
1820388464 93310137 6939507 740759555 1285206936 178376348 502278611 1450573622 1037127828 1034949289 654867343 1529195746 392035588
1395354340 87755422 889023311 1494613810 1447267605 1365321801 745425661 396473730 1308044878 1346811305 1568239320 705178736 1590079444
436248626 1977648522 1470503465 1402586708 552473416 1143408282 188213298 559412924 1884167637 1473442062 201305624 238962600 776532036
1238433432 1273512899 1431419379 620148550 1665947468 619290071 707900973 407487131 2113903881 7684930 1776808933 711845894 404158660
937370165 2038657199 1973387981 1642548899 1501252966 260152959 1472713773 624272813 1662739668 2025187190 1967681095 1850952926 437116466
1704365084 117691340 638422090 1943327684 1953443376 1876855542 1069755936 1237379107 349517445 588219756 1856669170 1057418418 895706887
1823089412 1065103348 625032172 397451659 1469262009 1562402316 298628210 1295166342 1057467587 1798976206 1555319301 382687713 476667372
1070575321 260401255 296864819 774044599 697517721 2001229904 1959955939 1335939811 1797073940 1756915667 1065311705 719346228 846811127
1414825130 1370746584 559996658 324767320 155789234 231602422 1389672269 780821396 619054081 711645630 195740084 317675292 2008811973
1253207672 370073850 1414647625 1435905395 1046741222 337739299 1859306640 1343606042 111178358 440340713 1197352298 912556190 1782280524
846942590 524688209 700108581 1566288819 1371499336 2114937732 726371155 1927459594 292218004 882160379 11614746 1682085273 1662861776
430646890 246247255 1859721860 1546348142 105575579 964445884 2118421593 1520223205 452867621 1017675567 1857962504 201690613 213801961
82262754 64803132 1411154259 1737518944 282828202 110613202 114723506 982936784 1676902021 148822842 950390688 255789528 1266235189
1242606974 1137949908 1277949958 777210498 653448036 1808518808 1023457753 364686248 1309383303 1129033333 1329132133 1280321648 501772890
1781999754 150517567 212251746 1983690368 164319529 1834514500 484238046 1775473788 624549797 767066249 1866086990 739273303 175003033
1415503240 78012497 552910253 1471294892 1344247886 1795519125 861761152 474613996 425245975 1315209188 235649157 1448703729 1679895436
1455032460 430233414 961543921 678770460 932026304 496060028 829388027 1144627850 232266749 1192707556 31308902 816504794 820697697
835856699 1989571043 595301038 1395132002 1186090428 1574806401 1473144500 1739000681 1498617647 689906598 1387034159 12895191 1144522235
1812882134 1328104339 1380173692 1113502215 860516127 777205054 154755639 1722060849 1455559564 328298285 70636429 196495343 1472576339
402903177 1329202900 1563885238 1219407971 2416948 12260289 655495367 561717988 1407392298 1641585795 389040743 733053144 1473102829
```

Ouput :-

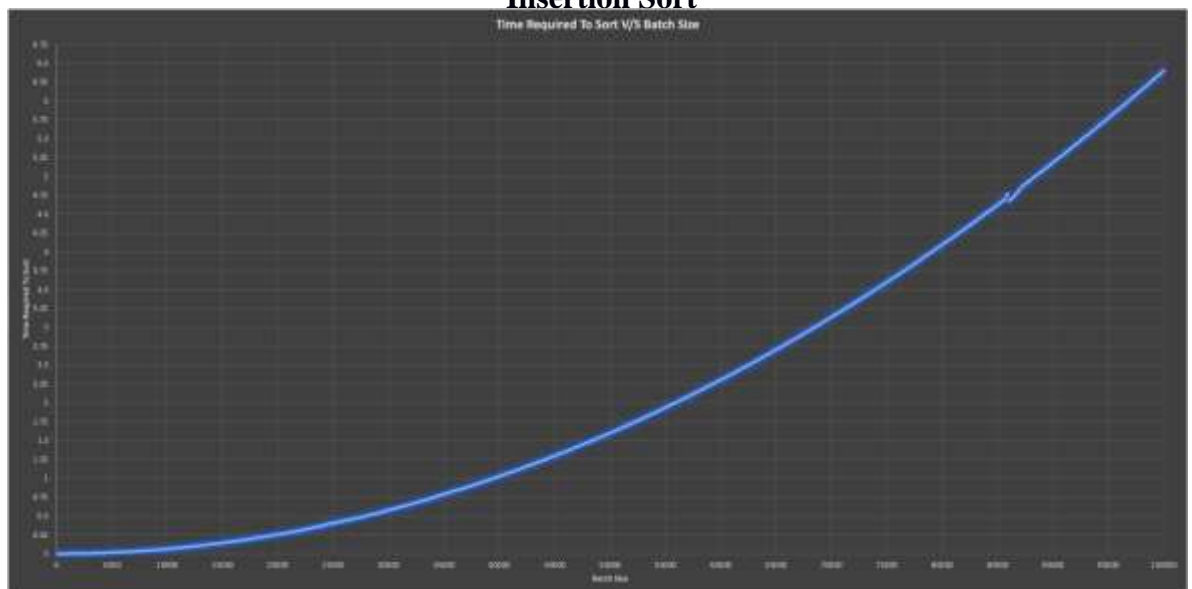
```
Sorted from 0 to 98299 in 6.17s
Sorted from 0 to 98399 in 6.18s
Sorted from 0 to 98499 in 6.19s
Sorted from 0 to 98599 in 6.21s
Sorted from 0 to 98699 in 6.22s
Sorted from 0 to 98799 in 6.23s
Sorted from 0 to 98899 in 6.24s
Sorted from 0 to 98999 in 6.26s
Sorted from 0 to 99099 in 6.27s
Sorted from 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.
```

.....

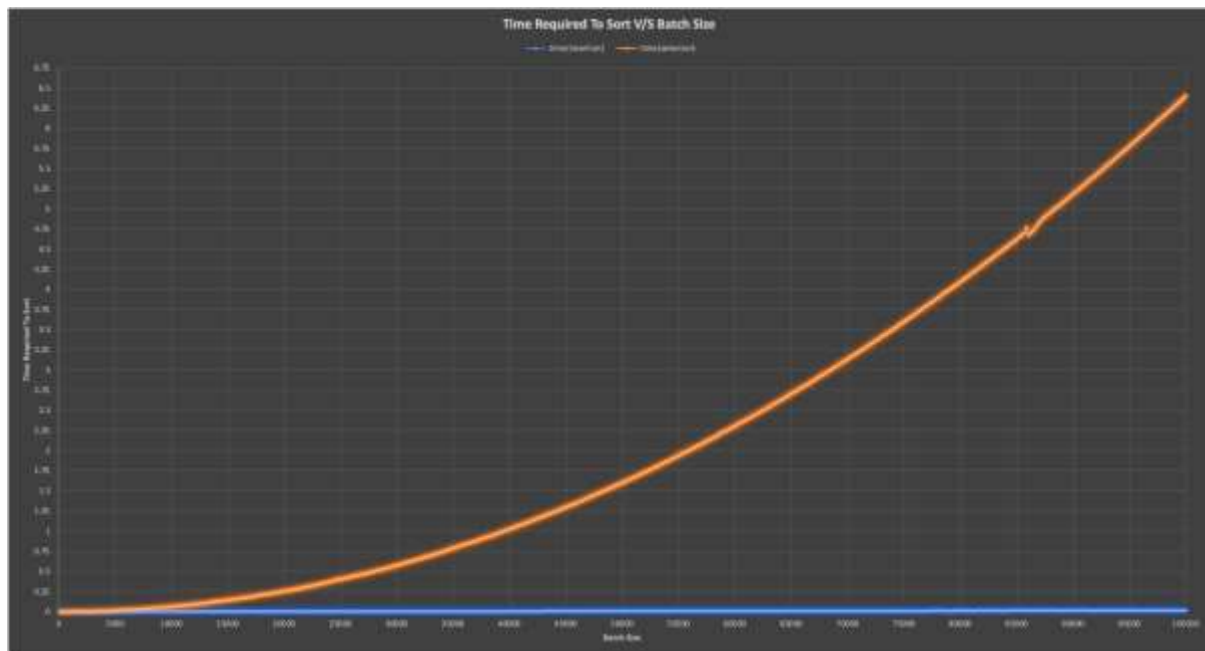
Graph :-



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