Name	Siddiqui Hawaiza Sabeel
UID no.	2022701010
Experiment No.	1 B
SUBJECT	DAA

AIM:	Experiment on finding the running time of an algorithm.
Program 1	
PROBLEM STATEMENT :	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.
ALGORITHM/ THEORY:	**Step 1:** Start. **Step 2:** Include the required libraries stdio.h, stdlib.h, time.h, and limits.h. **Step 3:** Define two sorting functions as per problem statement selection_sort and insertion_sort. **Step 4:** In the main function, using file handling open the file for writing. **Step 5:** Generate 10000 blocks of 100000 random numbers each and store them in the file. **Step 6:** Close the file after writing. **Step 7:** Open the file for reading. **Step 8:** For each block of 100 elements, read the elements from the file into two arrays.

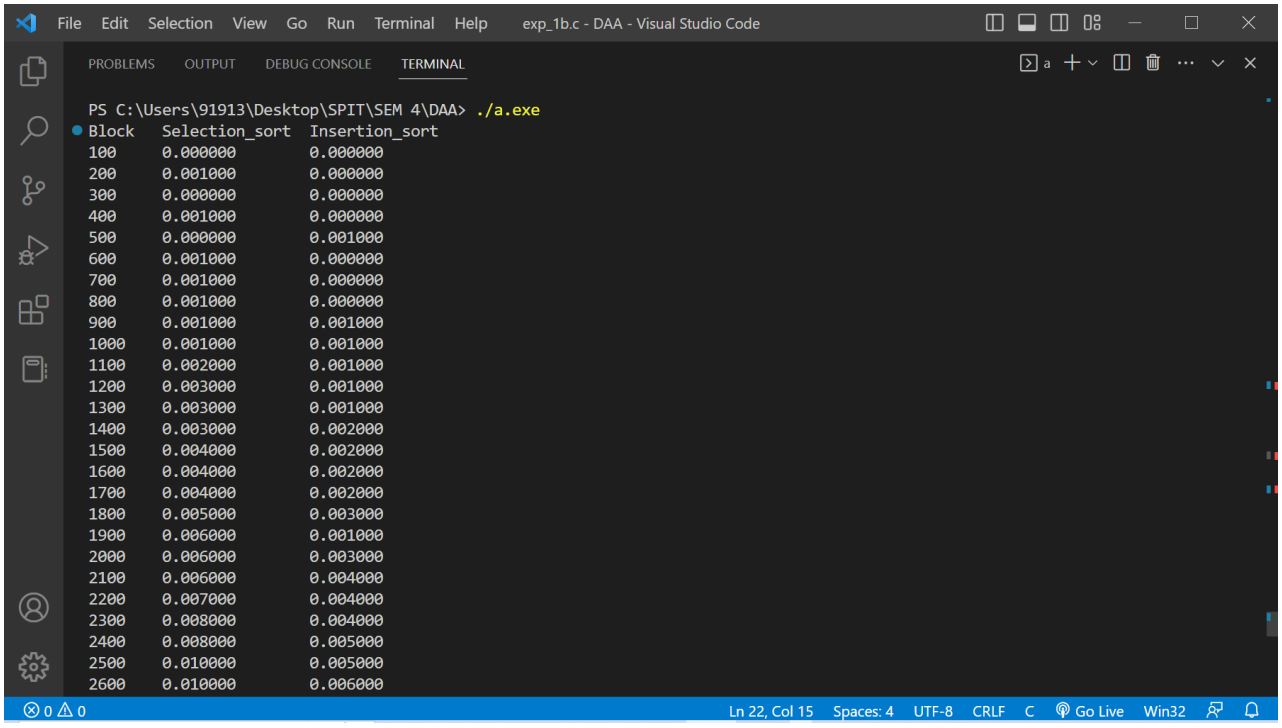
	Step 9: Sort the elements of array using the selection_sort function. **Step 10:** Use clock() to measure the time taken by the algorithm, and store the value inside a variable. **Step 11:** Sort the elements of array using the insertion_sort function. **Step 12:** Use clock() to measure the time taken by the algorithm, and store the value inside a variable. **Step 13:** Display the number of blocks and time taken by both of the algorithm to sort a specific blocks. **Step 14:** Repeat the process until it reaches 100000 blocks. **Step 15:** Close the file after reading. **Step 16:** Stop.

PROGRAM:	`#include<stdio.h>`
	`#include<stdlib.h>`
	`#include<time.h>`
	`#include<limits.h>`
	
	`void selection_sort(int arr[],int size) {`
	`  for(int i=0; i<size-1; i++) {`
	`  int min=i;`
	`  for(int j=i+1; j<size; j++)`
	`  if(arr[j] < arr[min])`
	`  min = j;`
	`  int temp = arr[min];`
	`  arr[min] = arr[i];`
	`  arr[i] = temp;`
	`  }`
	`}`
	
	`void insertion_sort(int arr[],int n) {`
	`  int i,key,j;`
	`  for(int i=1; i<n; i++) {`

```
        key = arr[i];
        j=i-1;
        while(j>=0 && arr[j]>key) {
            arr[j+1] = arr[j];
            j=j-1;
        }
        arr[j+1] = key;
    }
}

void main() {
    FILE *fp;
    fp  = fopen ("exp1b.txt", "w");
    srand((unsigned int) time(NULL));
    for(int block=0; block<1000; block++) {
        for(int i=0; i<100; i++) {
            int number = (int)(((float) rand() /
(float)(RAND_MAX))*100000);
            fprintf(fp,"%d ",number);
        }
        fputs("\n",fp);
    }
    fclose (fp);

   fp = fopen("exp1b.txt", "r");
   printf("Block\tSelection_sort\tInsertion_sort\n");
   for(int block=1; block<=100000; block++) {
    clock_t t,t1;

    int arr[(block+1)*100];
    int arr1[(block+1)*100];
    for(int i=0; i<(block+1)*100; i++){
        fscanf(fp, "%d", &arr[i]);
        arr1[i] = arr[i];
    }
    fseek(fp, 0, SEEK_SET);
    t = clock();
    selection_sort(arr,(block+1)*100);
    t = clock() - t;
    t1 = clock();
    insertion_sort(arr1,(block+1)*100);
    t1 = clock() - t1;
    double time_taken_selection_sort = ((double)t)/CLOCKS_PER_SEC;
```

```
    double time_taken_insertion_sort =
((double)t1)/CLOCKS_PER_SEC;
    printf("%d\t%f\t%f\n",(block*100),time_taken_selection_sort,ti
me_taken_insertion_sort);
   }

    fclose(fp);
}
```

RESULT:



File Edit Selection View Go Run Terminal Help exp_1b.c - DAA - Visual Studio Code

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
12100   0.255000        0.130000
12200   0.259000        0.144000
12300   0.258000        0.132000
12400   0.319000        0.181000
12500   0.325000        0.170000
12600   0.335000        0.225000
12700   0.451000        0.207000
12800   0.419000        0.347000
12900   0.607000        0.228000
13000   0.435000        0.225000
13100   0.667000        0.345000
13200   0.781000        0.364000
13300   0.833000        0.362000
13400   0.610000        0.266000
13500   0.480000        0.229000
13600   0.376000        0.238000
13700   0.346000        0.169000
13800   0.436000        0.172000
13900   0.328000        0.192000
14000   0.379000        0.237000
14100   0.470000        0.206000
14200   0.455000        0.207000
14300   0.432000        0.163000
14400   0.323000        0.167000
14500   0.368000        0.185000
14600   0.297000        0.161000
14700   0.317000        0.166000
14800   0.345000        0.214000
```

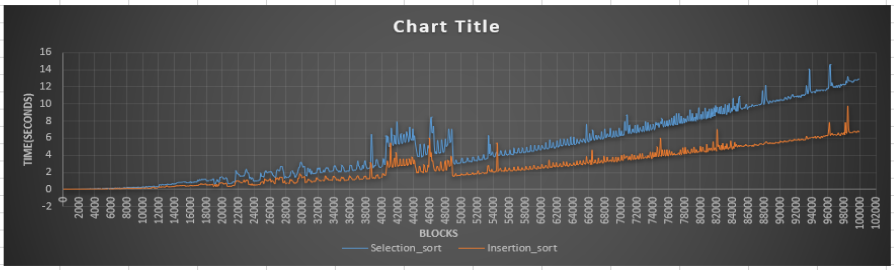
⊗ 0 ⚠ 0 Ln 22, Col 15 Spaces: 4 UTF-8 CRLF C Go Live Win32

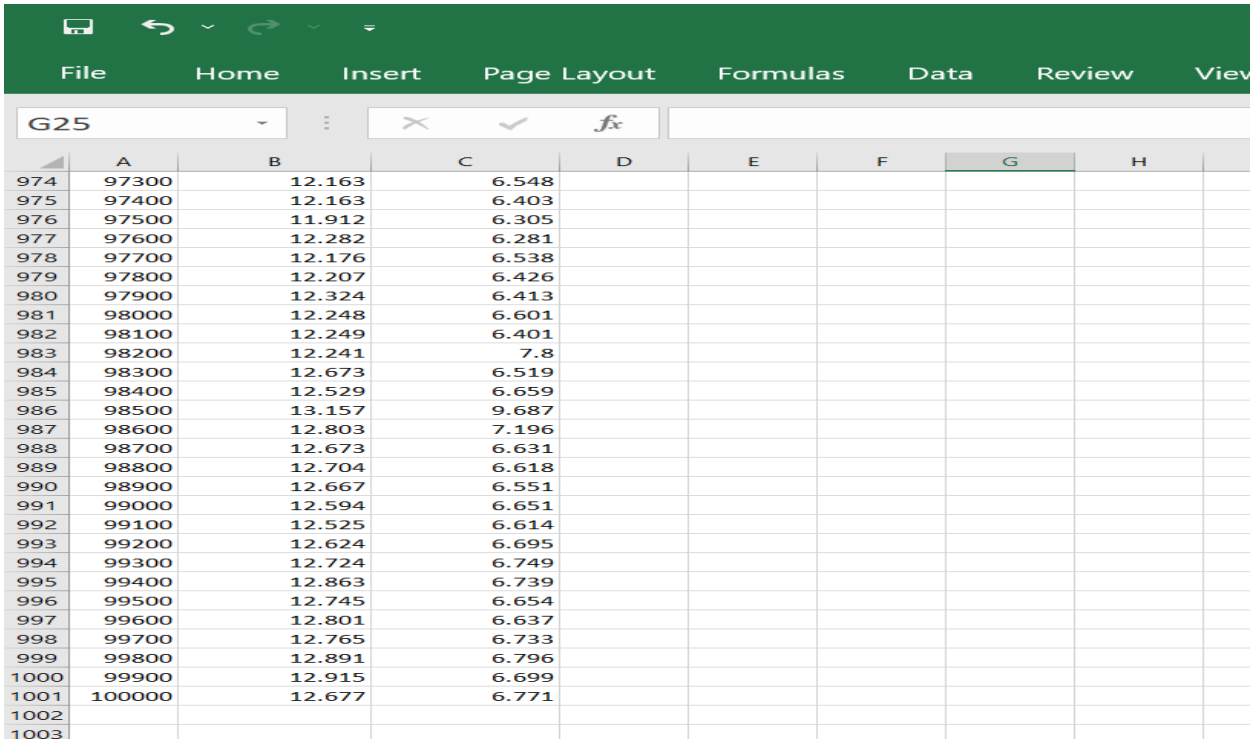
EXP1B - Excel

File Home Insert Page Layout Formulas Data Review View Help LOAD TEST TEAM

G25

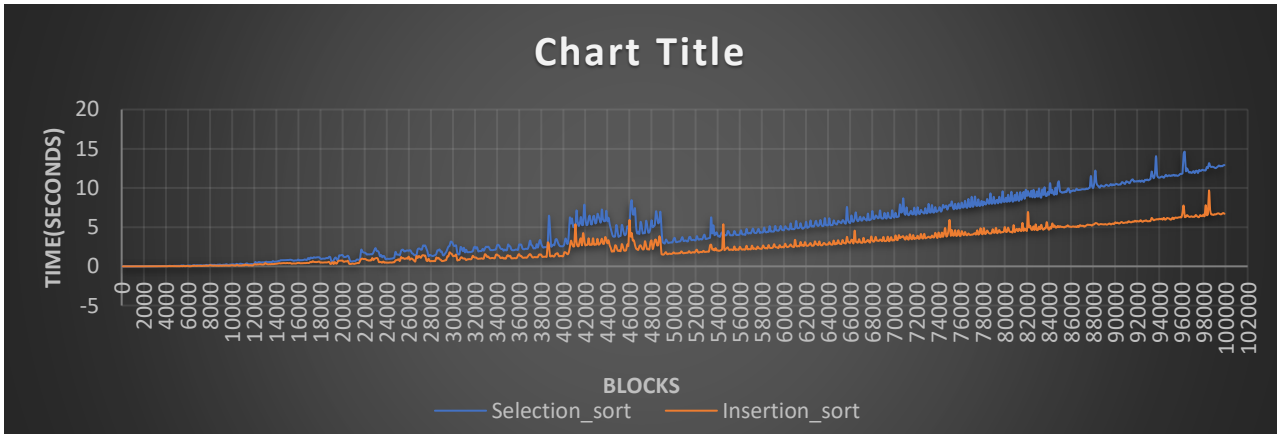
Block	Selection_sort	Insertion_sort
100	0	0
200	0	0
300	0.011	0
400	0.001	0
500	0.001	0
600	0.001	0
700	0.001	0.001
800	0.002	0
900	0.002	0.001
1000	0.002	0.002
1100	0.003	0.001
1200	0.003	0.002
1300	0.003	0.004
1400	0.005	0.003
1500	0.006	0.004
1600	0.006	0.004
1700	0.006	0.004
1800	0.006	0.004
1900	0.008	0.004
2000	0.01	0.004
2100	0.011	0.004
2200	0.011	0.006
2300	0.012	0.008
2400	0.014	0.008
2500	0.014	0.009
2600	0.015	0.009
2700	0.015	0.01
2800	0.017	0.009
2900	0.018	0.01
3000	0.02	0.012
3100	0.021	0.012
3200	0.023	0.011
3300	0.024	0.013
3400	0.027	0.017
3500	0.027	0.014
3600	0.03	0.016
3700	0.03	0.017





	A	B	C
974	97300	12.163	6.548
975	97400	12.163	6.403
976	97500	11.912	6.305
977	97600	12.282	6.281
978	97700	12.176	6.538
979	97800	12.207	6.426
980	97900	12.324	6.413
981	98000	12.248	6.601
982	98100	12.249	6.401
983	98200	12.241	7.8
984	98300	12.673	6.519
985	98400	12.529	6.659
986	98500	13.157	9.687
987	98600	12.803	7.196
988	98700	12.673	6.631
989	98800	12.704	6.618
990	98900	12.667	6.551
991	99000	12.594	6.651
992	99100	12.525	6.614
993	99200	12.624	6.695
994	99300	12.724	6.749
995	99400	12.863	6.739
996	99500	12.745	6.654
997	99600	12.801	6.637
998	99700	12.765	6.733
999	99800	12.891	6.796
1000	99900	12.915	6.699
1001	100000	12.677	6.771
1002			
1003			

GRAPH:



OBSERVATION:	• The above graph represents the amount of time (in seconds) required to sort blocks of integers using the Selection sort & Insertion sort algorithm. • In the above graph, X-axis represents no. of blocks and Y-axis represents Time. The maximum no. of blocks is 100000 on X-axis. • Maximum amount of time required to the sort 100000th block using selection sort is approx. 13.20 seconds and using insertion sort is 7.57 seconds.

	• As the no. of integers in blocks increases both the lines for selection sort and insertion sort grow exponentially and not linearly. • Sudden major spikes were most commonly observed between blocks 38000 to 50000 • The graph depicts that amount of time required to sort a block using selection sort increases quickly compared to insertion sort as no. of integers in the block increases.
CONCLUSION:	We found the running time of insertion sort and selection sort on each block and plotted a 2-D chart that shows the comparison of both algorithm's running time.