NAME:	**Sanskar Kamble**
UID:	**2021300054**
BATCH:	**C**
EXPERIMENT NO:	**02**

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

Details : The understanding of running time of algorithms is explored by implementing two basic sorting algorithms namely Insertion and Selection sorts. These algorithms work as follows.

Insertion sort: It works similar to the sorting of playing cards in hands. It is assumed that the first card is already sorted in the card game, and then we select an unsorted card. If the selected unsorted card is greater than the first card, it will be placed at the right side; otherwise, it will be placed at the left side. Similarly, all unsorted cards are taken and put in their exact place.

Selection sort: It first finds the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array. In this algorithm, the array is divided into two parts, first is the sorted part, and another one is the unsorted part. Initially, the sorted part of the array is empty, and the unsorted part is the given array. Sorted part is placed at the left, while the unsorted part is placed at the right. In selection sort, the first smallest element is selected from the unsorted array and placed at the first position. After that second smallest element is selected and placed in the second position. The process continues until the array is entirely sorted.

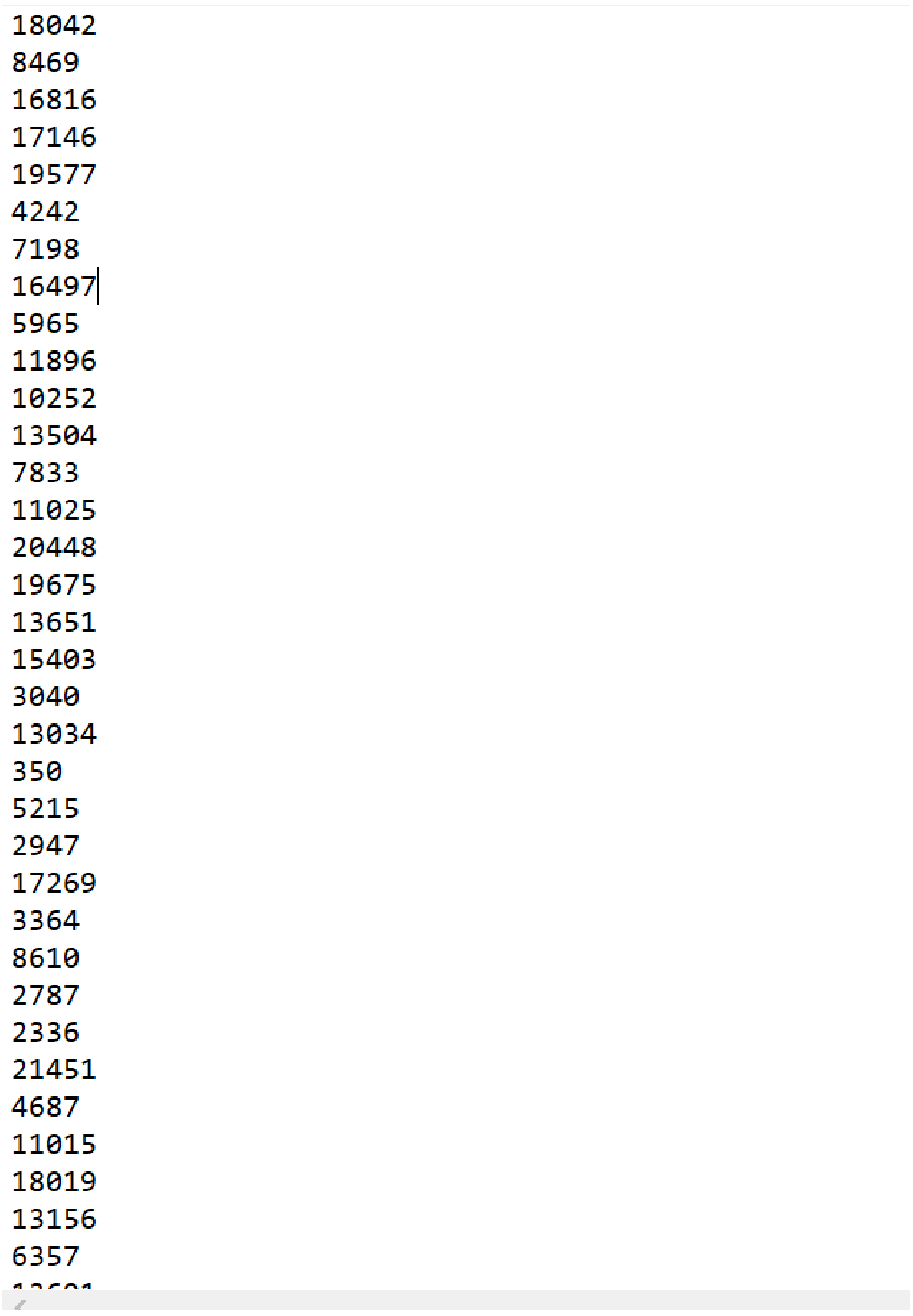
Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
const int limit = 100000;
const int block = 100;
void insertion_sort (FILE *f)
{
        FILE *fp;
        fp = fopen("insertion_sort.txt", "w");
    fprintf(fp,"Block Size\tTime Taken\n");
        int size = 0;
        for (int times = 0; times<limit/block; times++)
        {
                size+=block;
                int arr [size];
                for (int i = 0; i<size; ++i)
                        fscanf(f,"%d",&arr[i]);
                // now our array is ready, we perform insertion sort
                clock_t t;
                t = clock();
                int i, key, j;
                for (i = 1; i<size; 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;
        }
        t = clock()-t;
        double time_taken = ((double)t)/CLOCKS_PER_SEC;
        // storing the result in a file
        fprintf(fp,"%d\t%lf\n",size,time_taken);
        }
        fclose(fp);
}

void selection_sort (FILE *f) {
        FILE *fp;
        fp = fopen("selection_sort.txt", "w");
        fprintf(fp,"Block Size\tTime Taken\n");
        int size = 0;
        for (int times = 0; times<limit/block; times++) {
                 size+=block;
        int arr [size];
        for (int i = 0; i<size; ++i)
                fscanf(f,"%d",&arr[i]);
        // now our array is ready, we perform selection sort
        clock_t t;
        t = clock();
        int i, j, min;
    for (i = 0; i<size-1; i++) {
        min= i;
        for (j = i+1; j<size; j++) {
            if (arr[j]<arr[min])
            min= j;
         }
  if(min!=i) {
        int temp = arr[min];
        arr[min] = arr[i];
        arr[i] = temp;
}
    }
t = clock()-t;
double time_taken = ((double)t)/CLOCKS_PER_SEC;
// storing the result in a file
fprintf(fp,"%d\t%lf\n",size,time_taken);
```

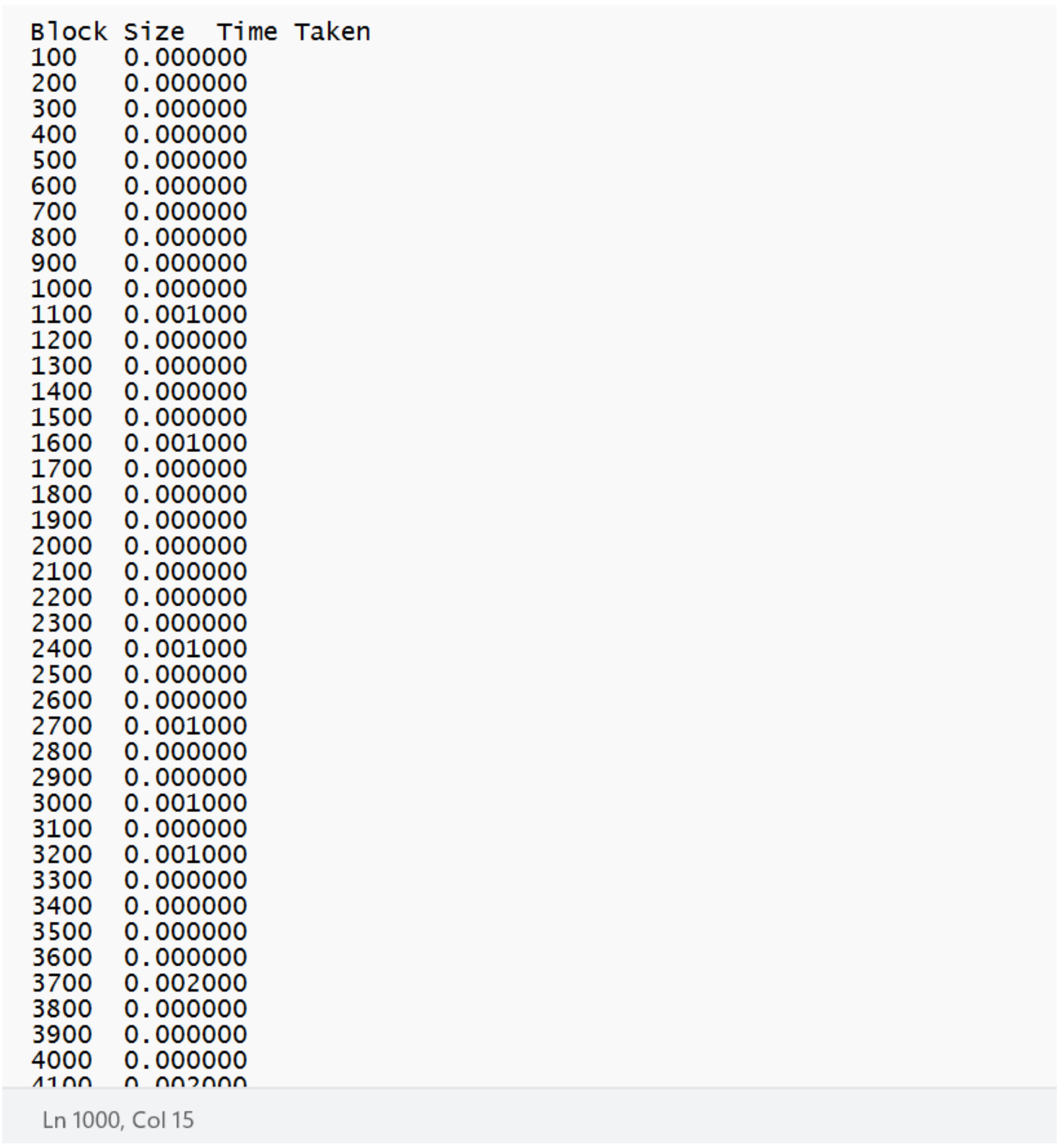
```
	}
	fclose(fp);
}

int main () {

	// generating 1,00,000 integers and storing them in a file
	FILE *fp1;
	fp1= fopen("Randomnum1.txt","w");
	for (int i=0;i<limit;i++)
		fprintf(fp1,"%d\n",rand()/100000);

	// insertion sort
	insertion_sort(fp1);

	// selection sort
	selection_sort(fp1);

	fclose(fp1);

	return 0;
}
```

1,00,000 Randomly Generated numbers:



```
18042
8469
16816
17146
19577
4242
7198
16497
5965
11896
10252
13504
7833
11025
20448
19675
13651
15403
3040
13034
350
5215
2947
17269
3364
8610
2787
2336
21451
4687
11015
18019
13156
6357
```

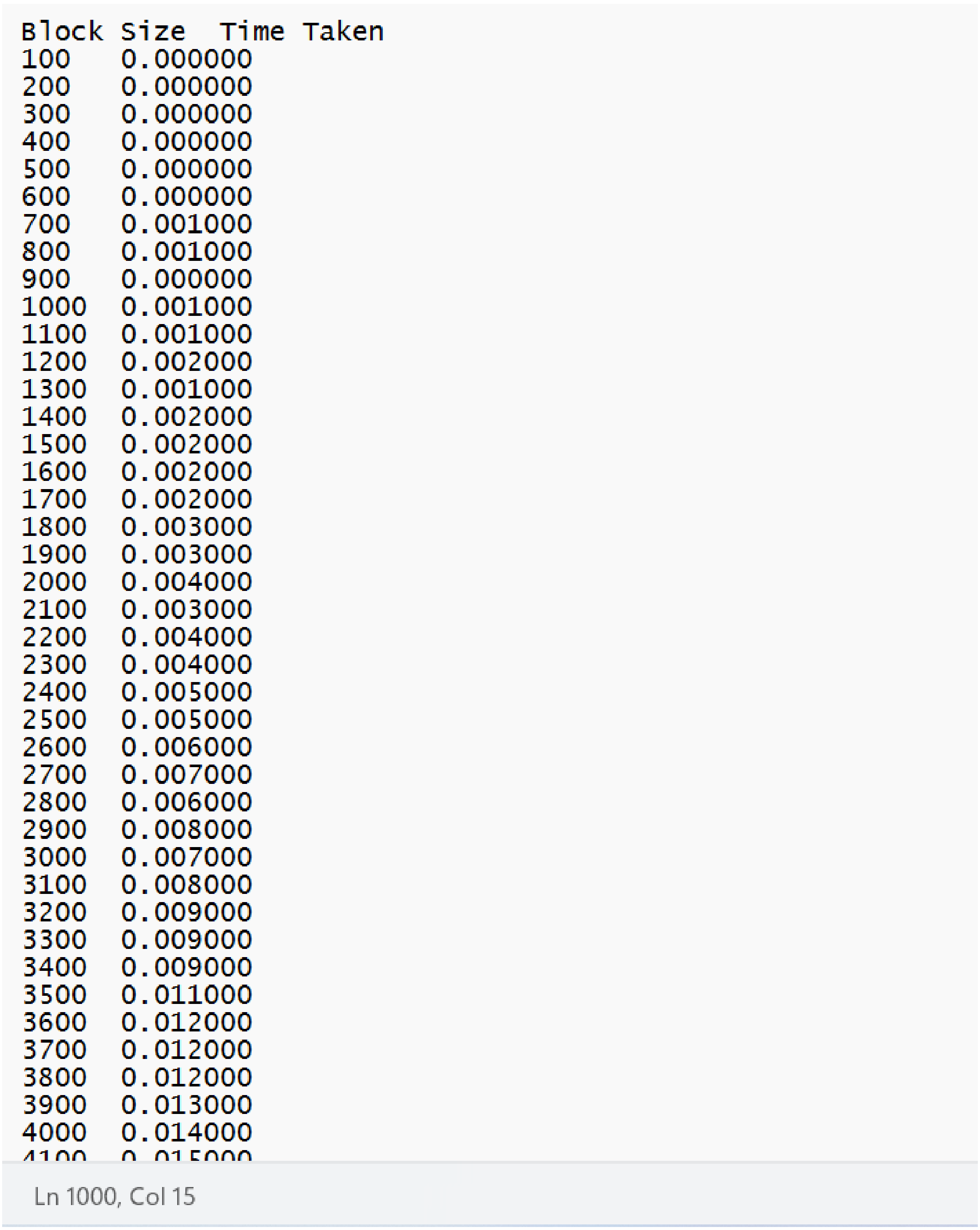
Insertion sort Time complexity:



```
Block Size  Time Taken
100   0.000000
200   0.000000
300   0.000000
400   0.000000
500   0.000000
600   0.000000
700   0.000000
800   0.000000
900   0.000000
1000  0.000000
1100  0.001000
1200  0.000000
1300  0.000000
1400  0.000000
1500  0.000000
1600  0.001000
1700  0.000000
1800  0.000000
1900  0.000000
2000  0.000000
2100  0.000000
2200  0.000000
2300  0.000000
2400  0.001000
2500  0.000000
2600  0.000000
2700  0.001000
2800  0.000000
2900  0.000000
3000  0.001000
3100  0.000000
3200  0.001000
3300  0.000000
3400  0.000000
3500  0.000000
3600  0.000000
3700  0.002000
3800  0.000000
3900  0.000000
4000  0.000000
4100  0.002000
```

Ln 1000, Col 15

Selection sort Time Complexity:



```
Block Size  Time Taken
100    0.000000
200    0.000000
300    0.000000
400    0.000000
500    0.000000
600    0.000000
700    0.001000
800    0.001000
900    0.000000
1000   0.001000
1100   0.001000
1200   0.002000
1300   0.001000
1400   0.002000
1500   0.002000
1600   0.002000
1700   0.002000
1800   0.003000
1900   0.003000
2000   0.004000
2100   0.003000
2200   0.004000
2300   0.004000
2400   0.005000
2500   0.005000
2600   0.006000
2700   0.007000
2800   0.006000
2900   0.008000
3000   0.007000
3100   0.008000
3200   0.009000
3300   0.009000
3400   0.009000
3500   0.011000
3600   0.012000
3700   0.012000
3800   0.012000
3900   0.013000
4000   0.014000
4100   0.015000
```

Ln 1000, Col 15

Conclusion: By performing the above experiment I was successfully able to observe that insertion sort is a much more efficient algorithm to sort numbers than selection sort as insertion sort 8 seconds to sort and took selection sort 50 minutes to sort the same amount of data. We also observe that even though the time complexity of selection sort is much greater than that of insertion sort, the space complexity for both is the same. Both insertion and selection sort have O(1) space complexity.