Lab Sheet 1: Measuring Running Time of an Algorithm

Aim:

To measure the execution time of different algorithms using the clock() function in Turbo C, and to study the effect

of algorithm complexity on running time for:

- **Simple loop (O(n))**
- **Nested loop ($O(n^2)$)**
- **Recursive function (Factorial)**

Procedure:

- **Open Turbo C and create a new C program.**
- **Include the required header files.**
- **Write the program to record the start time using clock().**
- **Execute the given loop or recursive function.**
- **Record the end time after execution.**
- **Calculate the execution time using CLOCKS_PER_SEC.**
- **Compile and run the program.**
- **Observe the time taken displayed on the screen.**

Program 1: Simple Loop

```
#include <stdio.h>
#include <time.h>
int main() {
int i, n = 1000000;
clock_t start, end;
double cpu_time;
start = clock();
for (i = 0; i < n; i++);
end = clock();
cpu_time = ((double)(end - start)) / CLOCKS_PER_SEC;
printf("Time taken: %f seconds\n", cpu_time);
return 0;
```

```
}
```

Program 2: Measuring Running Time of Nested Loops ($O(n^2)$)

```
#include <stdio.h>
#include <time.h>
int main() {
int i, j;
int n = 2000;
clock_t start, end;
double cpu_time;
start = clock();
for(i = 0; i < n; i++) {
for(j = 0; j < n; j++) {
// Empty loop body
}
}
end = clock();
cpu_time = ((double)(end - start)) / CLOCKS_PER_SEC;
printf("Time taken for O(n^2) loop: %f seconds\n", cpu_time);
return 0;
}
```

Program 3: Measuring Running Time of Recursive Function (Factorial)

```
#include <stdio.h>
#include <time.h>
long factorial(int n) {
if(n == 0)
return 1;
return n * factorial(n - 1);
}
int main() {
int n = 20;
clock_t start, end;
```

```
double cpu_time;

start = clock();

factorial(n);

end = clock();

cpu_time = ((double)(end - start)) / CLOCKS_PER_SEC;

printf("Time taken for recursive factorial: %f seconds\n", cpu_time);

return 0;

}
```

Result:

The execution time of different algorithms was measured successfully using Turbo C. It was observed that the running

time increases as the algorithm complexity increases from O(n) to $O(n^2)$.