Lab - 01

DIP : Design and Analysis of Algorithm

AIM : Find time and space complexity, Explore Insertion sort & Magic Square implementation algorithm

Name : Ambalia Harshit

Department : Computer Engineering (M.Tech sem II)

Roll no : MT001

Date : 01 Jan 2024

Measuring Time complexity :

- Different methods :

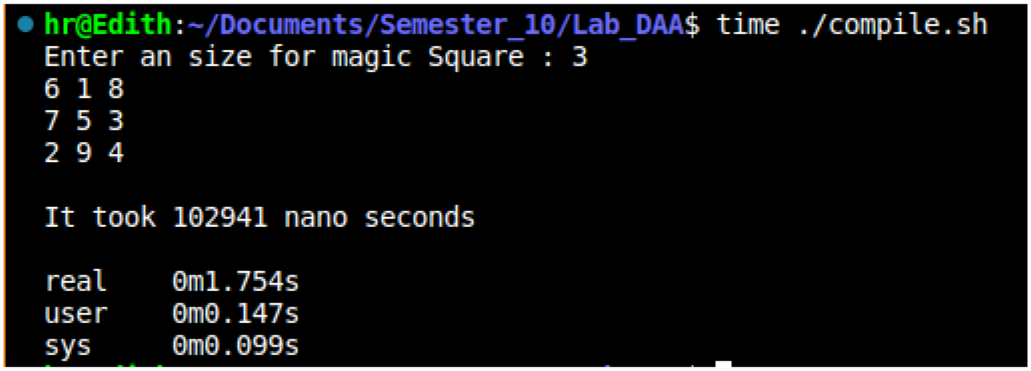
```
    1. time();
    2. clock();
    3. clock_gettime();
    4. getrusage()
    5. Using command line argument 'time';
    6. Using command line argument 'gprof';
```

- C system calls difference :

    1. time() gives in seconds
    2. clock() gives in milliseconds
    3. clock_gettime() gives in nanoseconds(more accurate)
        - but this function is not available in all the machines(POSIX-specific)
    4. getrusage() return CPU time in seconds with 6 point precision

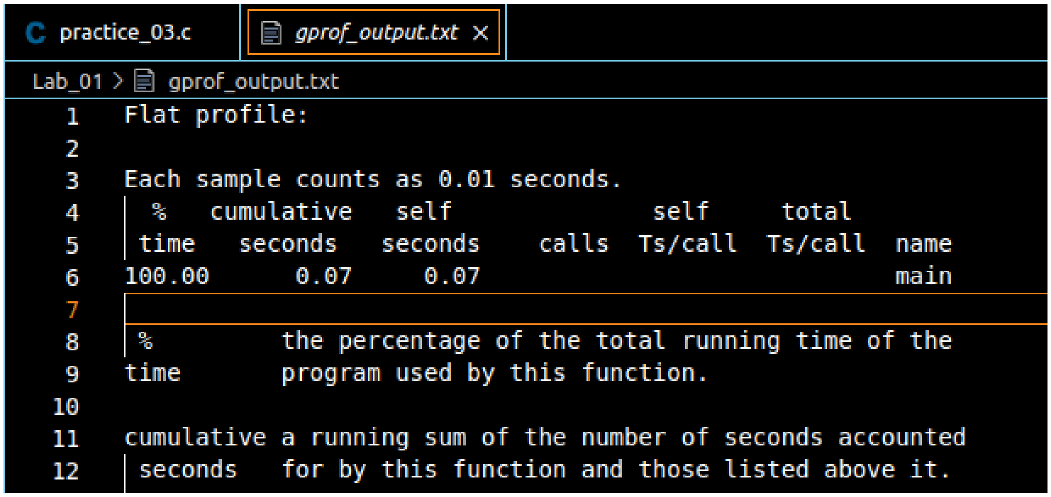
- Using command line commands :

    - Command : time
        - Ie. $ time ./practice_02
        - Returns 3 values :
            - Real time : Total elapsed time from start to finish, including any pauses or delays experienced by the program.
            - USer time : Represents the time the CPU spent actively executing instructions from your program's code itself.
            - Sys time : Reflects the time the CPU spent executing code on behalf of your program, within the operating system kernel.



```
hr@Edith:~/Documents/Semester_10/Lab_DAA$ time ./compile.sh
Enter an size for magic Square : 3
6 1 8
7 5 3
2 9 4

It took 102941 nano seconds

real    0m1.754s
user    0m0.147s
sys     0m0.099s
```

    - Command : gprof
        - gcc -pg -o ./practice_04 ./practice_04.c
        - ./practice_04
        - gprof practice_04 > gprof_output.txt



```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
100.00      0.07     0.07                             main

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.
```

- Observations :

- If we run the same code multiple times it’ll give a different time all the time…
    - Because, multitasking, background process, Cache, etc parameters play a role.

- If the same code is run using time commands in CLI and using any of 2 given C functions, CLI time is always given a higher value than using C commands… because…
    - Shell operations are used in this
    - Process creation(ie fork())
    - Launching the program(fetching/loading the file)
    - System calls to retrieve timing information(returns the fork())

Measuring Space complexity :

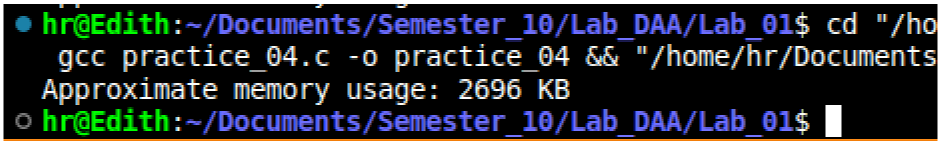
- Different methods :

```
1. getrusage()
```

- Code :

```
#include <stdio.h>
#include <sys/resource.h>

int main() {
   struct rusage usage;
   getrusage(RUSAGE_SELF, &usage);

   long maxrss = usage.ru_maxrss;
   printf("Approximate memory usage: %ld KB\n", maxrss);
   return 0;
}
```

- getrusage() : Retrieves detailed resource usage statistics for a process
    - maximum resident set size



```
● hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_01$ cd "/ho
  gcc practice_04.c -o practice_04 && "/home/hr/Documents
 Approximate memory usage: 2696 KB
○ hr@Edith:~/Documents/Semester_10/Lab_DAA/Lab_01$ 
```

Exercise :

1. Exercise 01 : Insertion sort

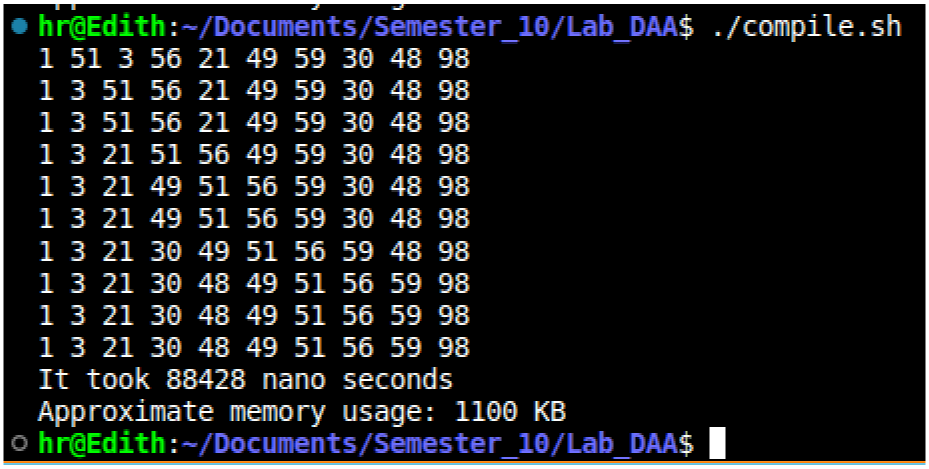
   - Algorithm :

```
INSERTION SORT (A)
        for j = 2 to A.length
        key = A[j]
        // Insert A[j] into the sorted sequence A[1.. j - 1]
        i = j - 1
        while i > 0 and A[i] > key
                A[i + 1] = A[i]
                ii = i -1
        A[i + 1] = key
```

   - Code :

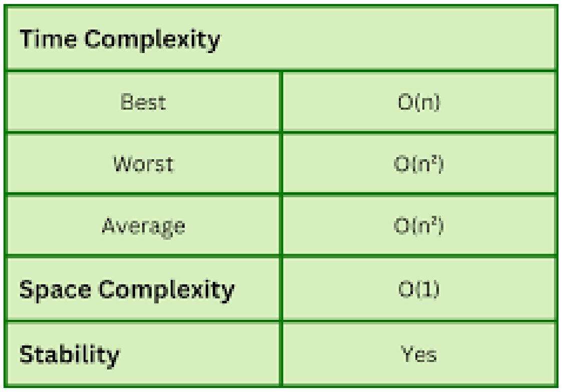
```
// Insertion sort

#include <stdio.h>
#include <time.h>
#include <sys/resource.h>

#include "/home/hr/Documents/Semester_10/Lab_DAA/HRA.h"

// void print_array(int arr[], int size) {
//     for (int i=0;i<size;i++) {
//         printf("%d ", arr[i]);
//     }
//     printf("\n");
// }

int main() {
   int length = 10;
   struct timespec start, end;

   int intList[] = {51, 1, 3, 56, 21, 49, 59, 30, 48, 98, 22};

   struct rusage usage;
   getrusage(RUSAGE_SELF, &usage);
   /* Best case - Minimum Swapping*/
   // int intList[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69,
70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100};
   // Average time taken : 2002740.5(10 test cases)

   /* Worst case - Maximum swapping */
   // int intList[] = {100, 99, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89,
88, 87, 86, 85, 84, 83, 82, 81, 80, 79, 78, 77, 76, 75, 74, 73, 72, 71,
70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 60, 59, 58, 57, 56, 55, 54, 53,
52, 51, 50, 49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35,
34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17,
16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
   // Average time taken : 2312648.5(10 test cases)

   /* Average case - Average swapping */
   // int intList[] = {54, 23, 12, 87, 45, 67, 91, 32, 78, 60, 41, 19, 89,
36, 75, 28, 53, 65, 95, 14, 83, 9, 70, 47, 98, 10, 68, 55, 27, 71, 94, 49,
6, 80, 42, 73, 88, 20, 64, 34, 3, 99, 31, 59, 81, 37, 22, 50, 72, 15, 93,
85, 26, 8, 62, 58, 33, 16, 44, 76, 13, 38, 21, 2, 56, 86, 46, 96, 77, 5,
18, 67, 66, 52, 11, 7, 61, 30, 1, 79, 25, 40, 43, 97, 63, 84, 92, 48, 4,
29, 24, 17, 35, 82, 51, 69, 57, 74, 39, 90, 59};
   // Average time taken : 2126839.8(10 test cases)

   clock_gettime(CLOCK_MONOTONIC, &start);

   /* ---------------------- METHOD 01 - Simple direct method
---------------------- */
   // for (int i=1;i<length;i++) {
   //     int current = intList[i];
   //     int j = i-1;

   //     while(intList[j]>current && j>=0) {
   //         intList[j+1] = intList[j];
   //         j--;
   //     }
   //     print_array(intList, length);
   //     intList[j+1] = current;
   // }
   /* ---------------------- METHOD 02 - Calls predefined function
---------------------- */
   /* This function will sort the passed integer array, while printing
array for each stage */
   insertionSort(intList, length);
   /*
---------------------------------------------------------------------------
--------- */

   clock_gettime(CLOCK_MONOTONIC, &end);


   long long elapsed_ns = (end.tv_sec - start.tv_sec) * 1000000000 +
(end.tv_nsec - start.tv_nsec);
   for (int i=0;i<length;i++) {
       printf("%d ", intList[i]);
   }
   printf("\nIt took %lld nano seconds\n", elapsed_ns);

   long maxrss = usage.ru_maxrss;
   printf("Approximate memory usage: %ld KB\n", maxrss);

   return 0;
}
```

- Output screen-shots :



```
hr@Edith:~/Documents/Semester_10/Lab_DAA$ ./compile.sh
1 51 3 56 21 49 59 30 48 98
1 3 51 56 21 49 59 30 48 98
1 3 51 56 21 49 59 30 48 98
1 3 21 51 56 49 59 30 48 98
1 3 21 49 51 56 59 30 48 98
1 3 21 49 51 56 59 30 48 98
1 3 21 30 49 51 56 59 48 98
1 3 21 30 48 49 51 56 59 98
1 3 21 30 48 49 51 56 59 98
1 3 21 30 48 49 51 56 59 98
It took 88428 nano seconds
Approximate memory usage: 1100 KB
hr@Edith:~/Documents/Semester_10/Lab_DAA$
```

- Understandings :
  - Best case : Array is already sorted, So a minimum number of swapping will be performed, which results in faster execution.
    - For a given array in Code, Average time taken : 2002740.5 nanoseconds(10 test cases).
  - Worst Case : Array is reversely sorted, So maximum number of swapping will be performed, which results in timely execution compared to best case.
    - For a given array in Code, Average time taken : 2312648.5 nanoseconds(10 test cases).
  - So, we can see time taken by worst case is comparably more than best case.



Time Complexity	
Best	O(n)
Worst	$O(n^2)$
Average	$O(n^2)$
Space Complexity	O(1)
Stability	Yes

2. Exercise 02 : Magic Square

    - Algorithm :

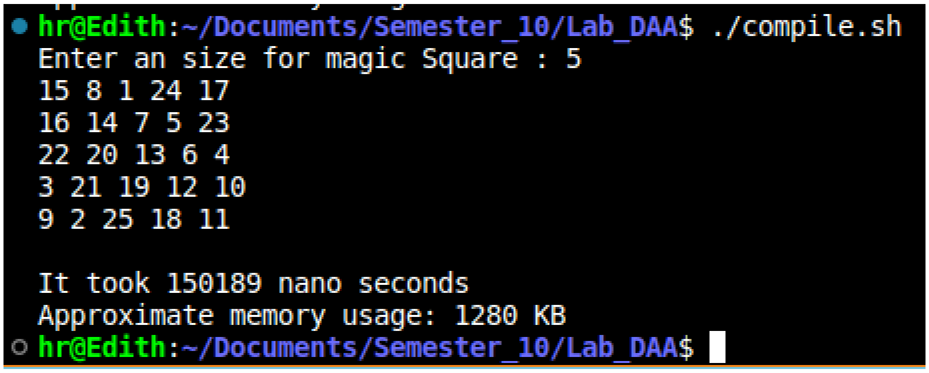
```
CreateSquare(mat, r, c)
    Input: The matrix.
    Output: Row and Column.

    Begin
    count := 1
    fill all elements in mat to 0
    range := r * c
    i := 0
    j := c/2
    mat[i, j] := count //center of top row
    while count < range, do
        increase count by 1
        if both i and j crosses the matrix range, then
            increase i by 1
        else if only i crosses the matrix range, then
            i := c – 1
            decrease j by 1
        else if only j crosses the matrix range, then
            j := c – 1
            decrease i by 1
        else if i and j are in the matrix and element in (i, j) ≠ 0, then
            increase i by 1
        Else
            decrease i and j by 1
        mat[i, j] := count
    Done
    display the matrix mat
End
```

- Code :

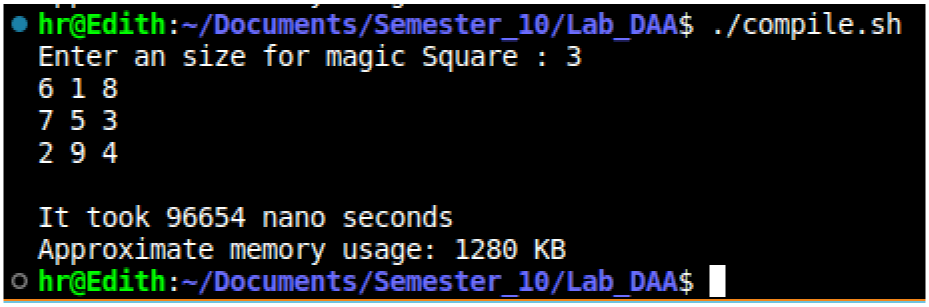
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/resource.h>
#include "/home/hr/Documents/Semester_10/Lab_DAA/HRA.h"


int main() {
   int length;
   printf("Enter an size for magic Square : ");
   scanf("%d", &length);
   if (length%2==0) {
       printf("Enter odd number\n");
       exit(0);
   }
   struct timespec start, end;
   struct rusage usage;
   getrusage(RUSAGE_SELF, &usage);

   clock_gettime(CLOCK_MONOTONIC, &start);

   /* ---------------------- METHOD 01 - Simple direct method
---------------------- */
   // int r = length;
   // int c = length;
   // int i, j, range;

   // if(length%2==0) {
   //     printf("Enter odd number\n");
   //     exit (0);
   // }

   // int count = 1;
   // int mat[length][length];
   // for (int i=0;i<length;i++) {
   //     for (int j=0;j<length;j++) {
   //         mat[i][j] = 0;
   //     }
   // }

   // range = r*c;
   // i = 0;
   // j = c/2;
   // mat[i][j] = count;

   // while(count<range) {
   //     count++;

   //     if(( (i-1)<0 || (i-1)>length-1) && ( (j-1)<0 || (j-1)>length-1)
) {
   //         i = i+1;
   //     }
   //     else if((i-1)<0 || (i-1)>length-1) {
   //         i = c-1;
   //         j = j-1;
   //     }
   //     else if((j-1)<0 || (j-1)>length-1) {
   //         j = c-1;
   //         i = i-1;
   //     }
   //     else if((mat[i-1][j-1]!=0)) {
   //         i = i+1;
   //     }
   //     else {
   //         i = i-1;
   //         j = j-1;
   //     }
   //     mat[i][j] = count;
   // }

   // for (int i=0;i<length;i++) {
   //     for (int j=0;j<length;j++) {
   //         printf("%d  ", mat[i][j]);
   //     }
   //     printf("\n");
   // }
```

```
    /* ---------------------- METHOD 02 - Calls predefined function
---------------------- */
    /* This function will create magic square for given size and will
return the pointer to that 2d function */
    int** matrix = create_magic_square(length);
    for (int i = 0; i < length; i++) {
        for (int j = 0; j < length; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
    /*
--------------------------------------------------------------------------
--------- */

    clock_gettime(CLOCK_MONOTONIC, &end);

    long long elapsed_ns = (end.tv_sec - start.tv_sec) * 1000000000 +
(end.tv_nsec - start.tv_nsec);
    printf("\nIt took %lld nano seconds\n", elapsed_ns);

    long maxrss = usage.ru_maxrss;
    printf("Approximate memory usage: %ld KB\n", maxrss);

    // Freeing the allocated memory
    for (int i = 0; i < length; i++) {
        free(matrix[i]);
    }
    free(matrix);

    return 0;

}
```

- Output Screenshots :



```
hr@Edith:~/Documents/Semester_10/Lab_DAA$ ./compile.sh
Enter an size for magic Square : 5
15 8 1 24 17
16 14 7 5 23
22 20 13 6 4
3 21 19 12 10
9 2 25 18 11

It took 150189 nano seconds
Approximate memory usage: 1280 KB
hr@Edith:~/Documents/Semester_10/Lab_DAA$
```



```
hr@Edith:~/Documents/Semester_10/Lab_DAA$ ./compile.sh
Enter an size for magic Square : 3
6 1 8
7 5 3
2 9 4

It took 96654 nano seconds
Approximate memory usage: 1280 KB
hr@Edith:~/Documents/Semester_10/Lab_DAA$
```