SORTING:

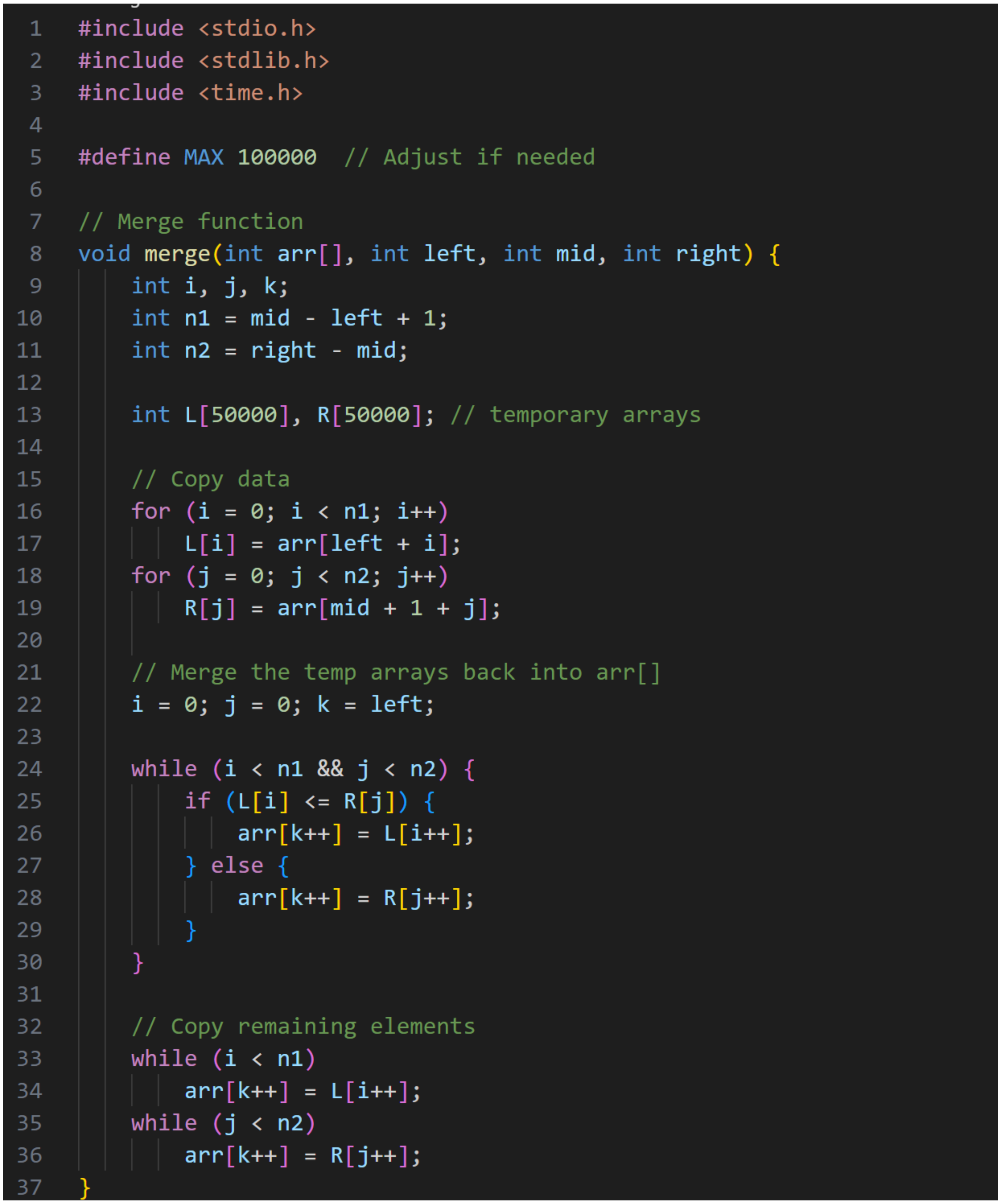
2 (a) Design and implement C Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

Pseudocode:

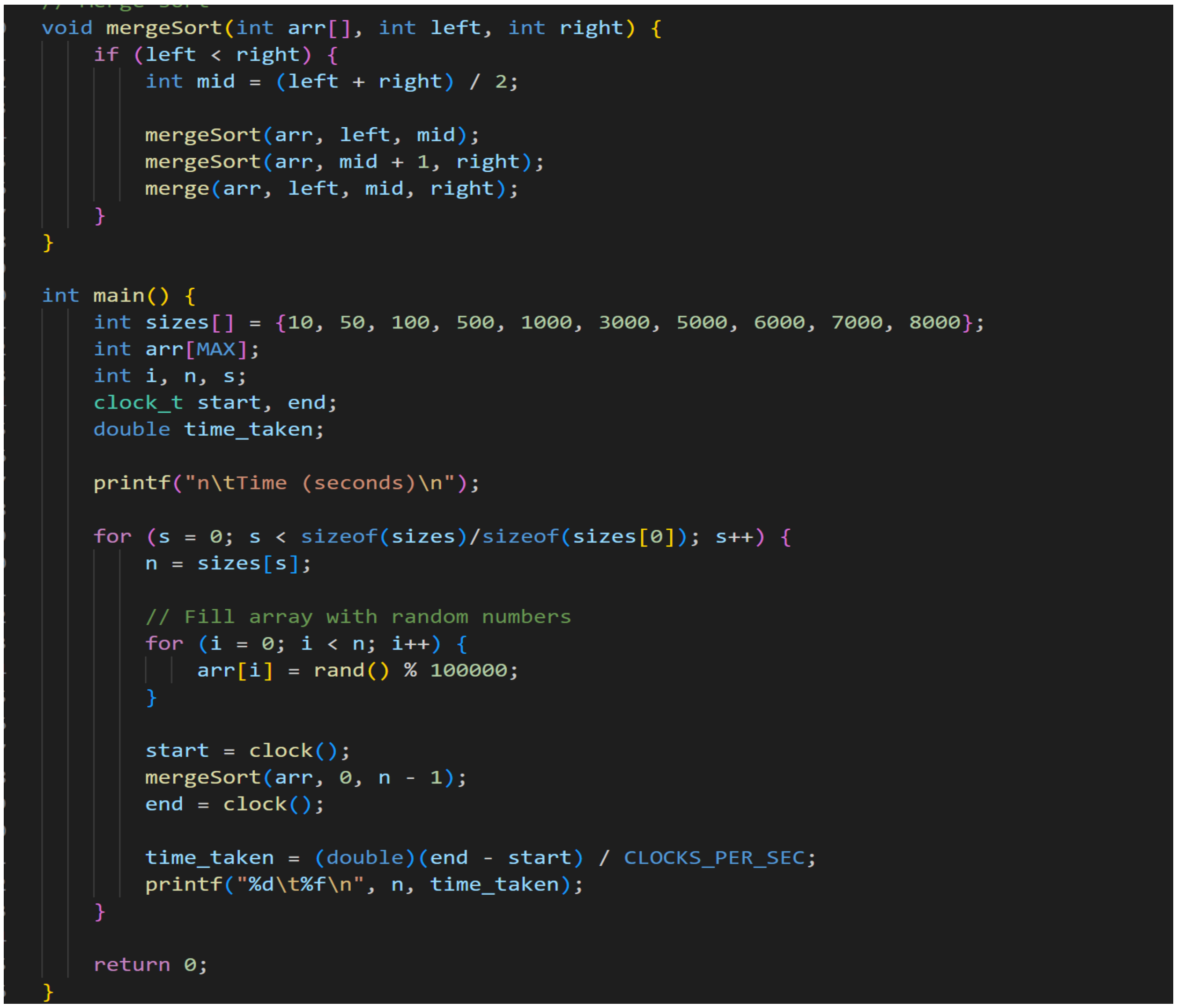
```
FUNCTION Merge(arr, left, mid, right)
    n1 ← mid - left + 1
    n2 ← right - mid

  CREATE array L[1..n1]
  CREATE array R[1..n2]

    FOR i ← 0 TO n1-1
        L[i] ← arr[left + i]
  END FOR

    FOR j ← 0 TO n2-1
        R[j] ← arr[mid + 1 + j]
  END FOR

    i ← 0, j ← 0, k ← left

  WHILE i < n1 AND j < n2
        IF L[i] ≤ R[j] THEN
            arr[k] ← L[i]
            i ← i + 1
    ELSE
            arr[k] ← R[j]
            j ← j + 1
    END IF
        k ← k + 1
  END WHILE

  WHILE i < n1
        arr[k] ← L[i]
        i ← i + 1
```

```
            k ← k + 1
    END WHILE

    WHILE j < n2
            arr[k] ← R[j]
            j ← j + 1
            k ← k + 1
    END WHILE
END FUNCTION


FUNCTION MergeSort(arr, left, right)
    IF left < right THEN
            mid ← (left + right) / 2
        MergeSort(arr, left, mid)
        MergeSort(arr, mid + 1, right)
        Merge(arr, left, mid, right)
    END IF
END FUNCTION


MAIN PROGRAM
    PRINT "n   Time(seconds)"

    FOR n = 1000 TO 20000 STEP 3000
        CREATE array arr[1..n]

        FOR i = 0 TO n-1
                arr[i] ← RANDOM(0..100000)    // generate random numbers
        END FOR

            start_time ← CURRENT_CLOCK()

        CALL MergeSort(arr, 0, n-1)

            end_time ← CURRENT_CLOCK()

            time_taken ← (end_time - start_time)

        PRINT n, time_taken
    END FOR
```

Code:

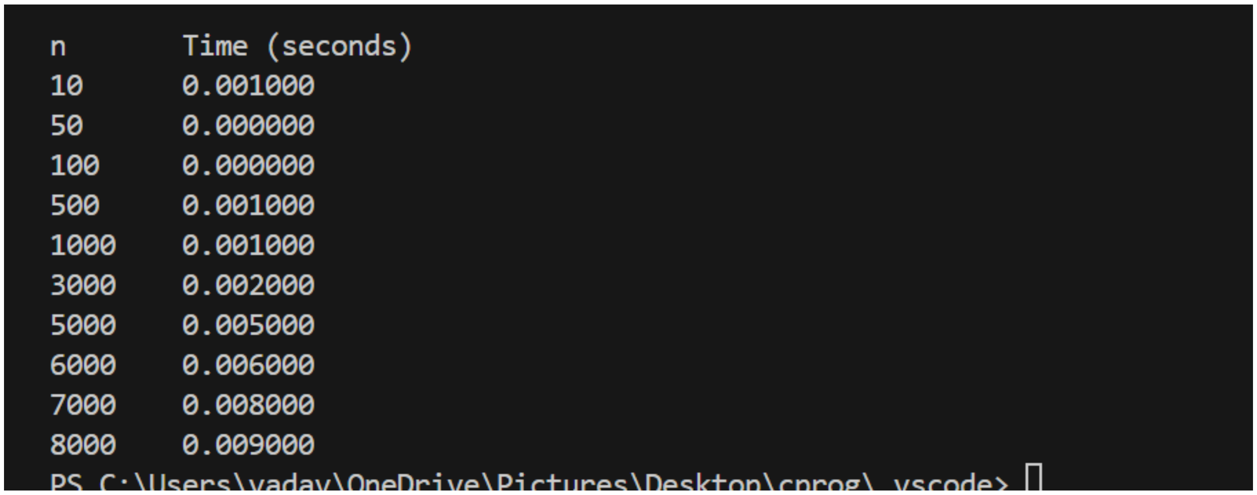


```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX 100000  // Adjust if needed

// Merge function
void merge(int arr[], int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[50000], R[50000]; // temporary arrays

    // Copy data
    for (i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    // Merge the temp arrays back into arr[]
    i = 0; j = 0; k = left;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k++] = L[i++];
        } else {
            arr[k++] = R[j++];
        }
    }

    // Copy remaining elements
    while (i < n1)
        arr[k++] = L[i++];
    while (j < n2)
        arr[k++] = R[j++];
}
```



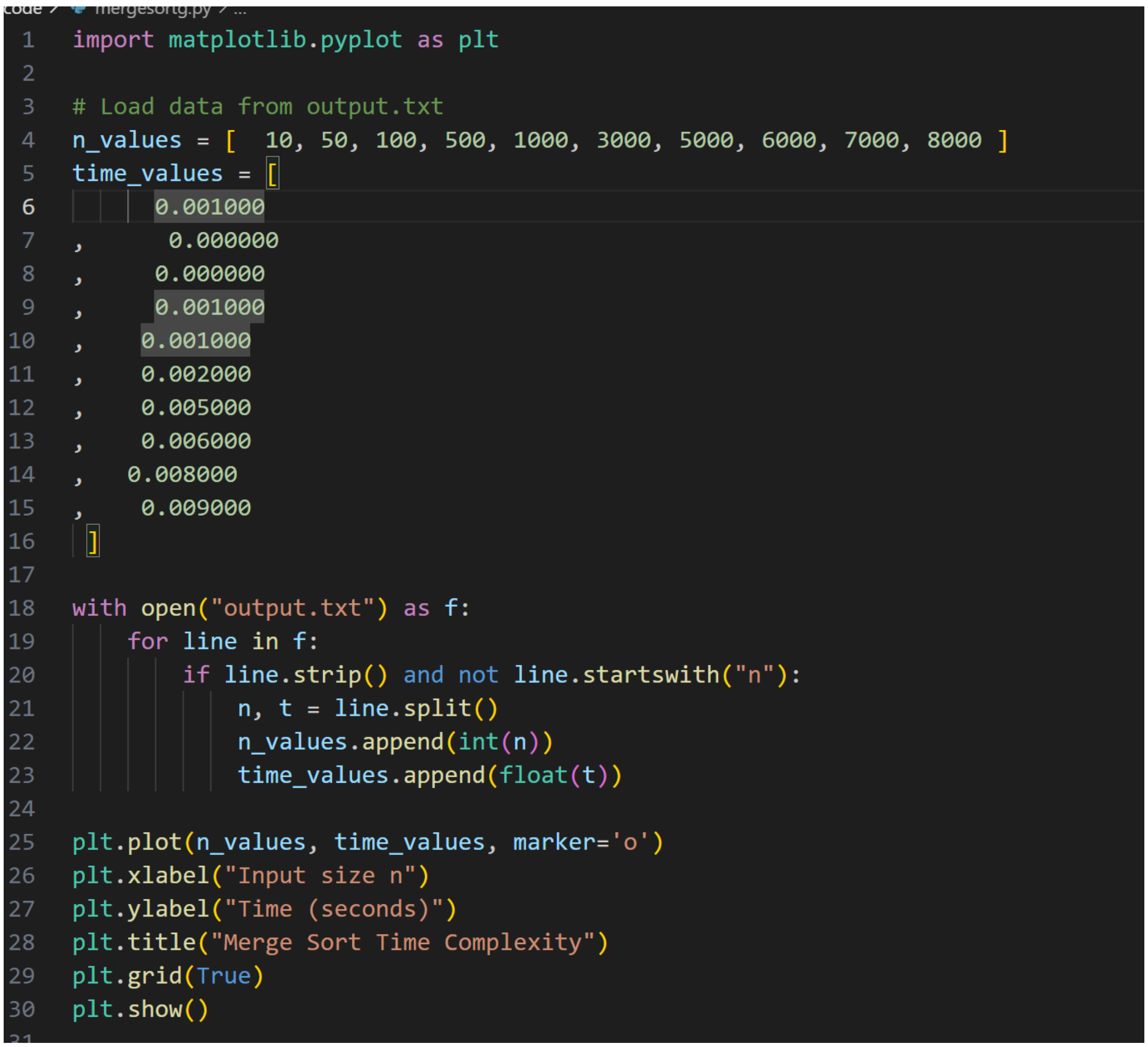
```c
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

int main() {
    int sizes[] = {10, 50, 100, 500, 1000, 3000, 5000, 6000, 7000, 8000};
    int arr[MAX];
    int i, n, s;
    clock_t start, end;
    double time_taken;

    printf("n\tTime (seconds)\n");

    for (s = 0; s < sizeof(sizes)/sizeof(sizes[0]); s++) {
        n = sizes[s];

        // Fill array with random numbers
        for (i = 0; i < n; i++) {
            arr[i] = rand() % 100000;
        }

        start = clock();
        mergeSort(arr, 0, n - 1);
        end = clock();

        time_taken = (double)(end - start) / CLOCKS_PER_SEC;
        printf("%d\t%f\n", n, time_taken);
    }

    return 0;
}
```

OUTPUT:



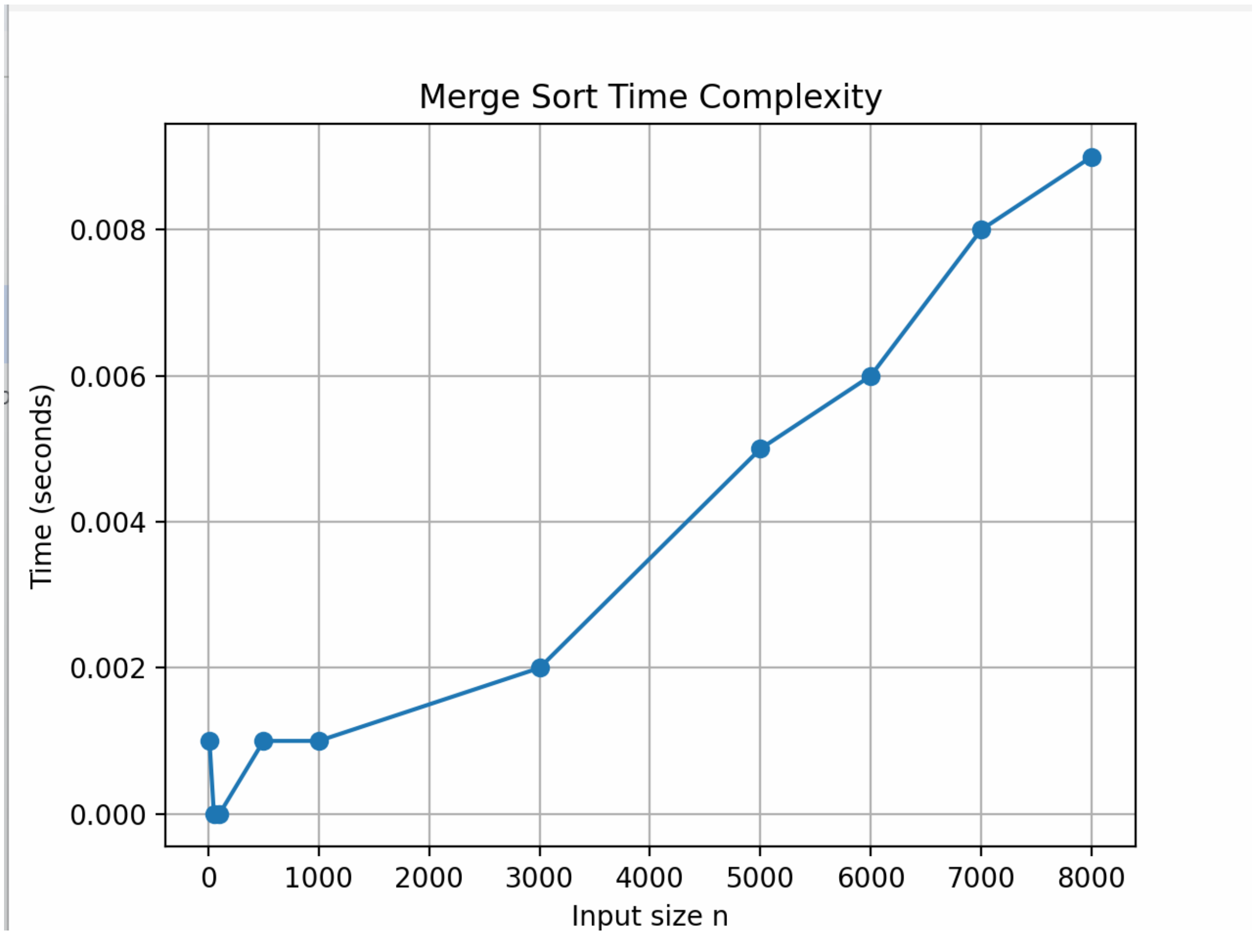
```
n       Time (seconds)
10      0.001000
50      0.000000
100     0.000000
500     0.001000
1000    0.001000
3000    0.002000
5000    0.005000
6000    0.006000
7000    0.008000
8000    0.009000
PS C:\Users\yadav\OneDrive\Pictures\Desktop\cprog\.vscode>
```

PYTHON CODE:



```python
import matplotlib.pyplot as plt

# Load data from output.txt
n_values = [  10, 50, 100, 500, 1000, 3000, 5000, 6000, 7000, 8000 ]
time_values = [
      0.001000
,      0.000000
,     0.000000
,     0.001000
,    0.001000
,    0.002000
,    0.005000
,    0.006000
,   0.008000
,    0.009000
 ]

with open("output.txt") as f:
    for line in f:
        if line.strip() and not line.startswith("n"):
            n, t = line.split()
            n_values.append(int(n))
            time_values.append(float(t))

plt.plot(n_values, time_values, marker='o')
plt.xlabel("Input size n")
plt.ylabel("Time (seconds)")
plt.title("Merge Sort Time Complexity")
plt.grid(True)
plt.show()
```

GRAPH:



Merge Sort Time Complexity
Time (seconds)
0.000
0.002
0.004
0.006
0.008
0
1000
2000
3000
4000
5000
6000
7000
8000
Input size n

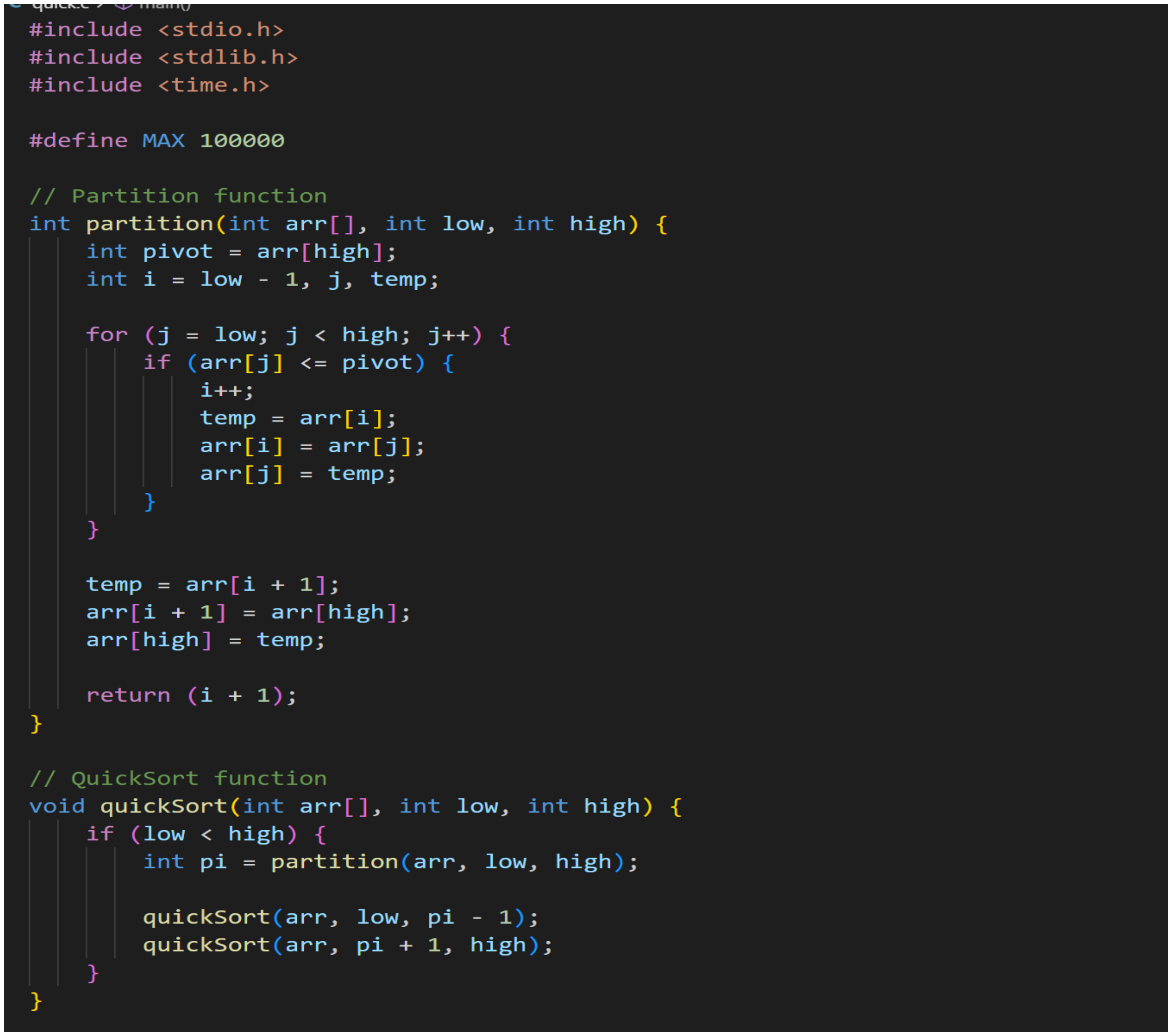
b)Design and implement C Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

Pseudocode:

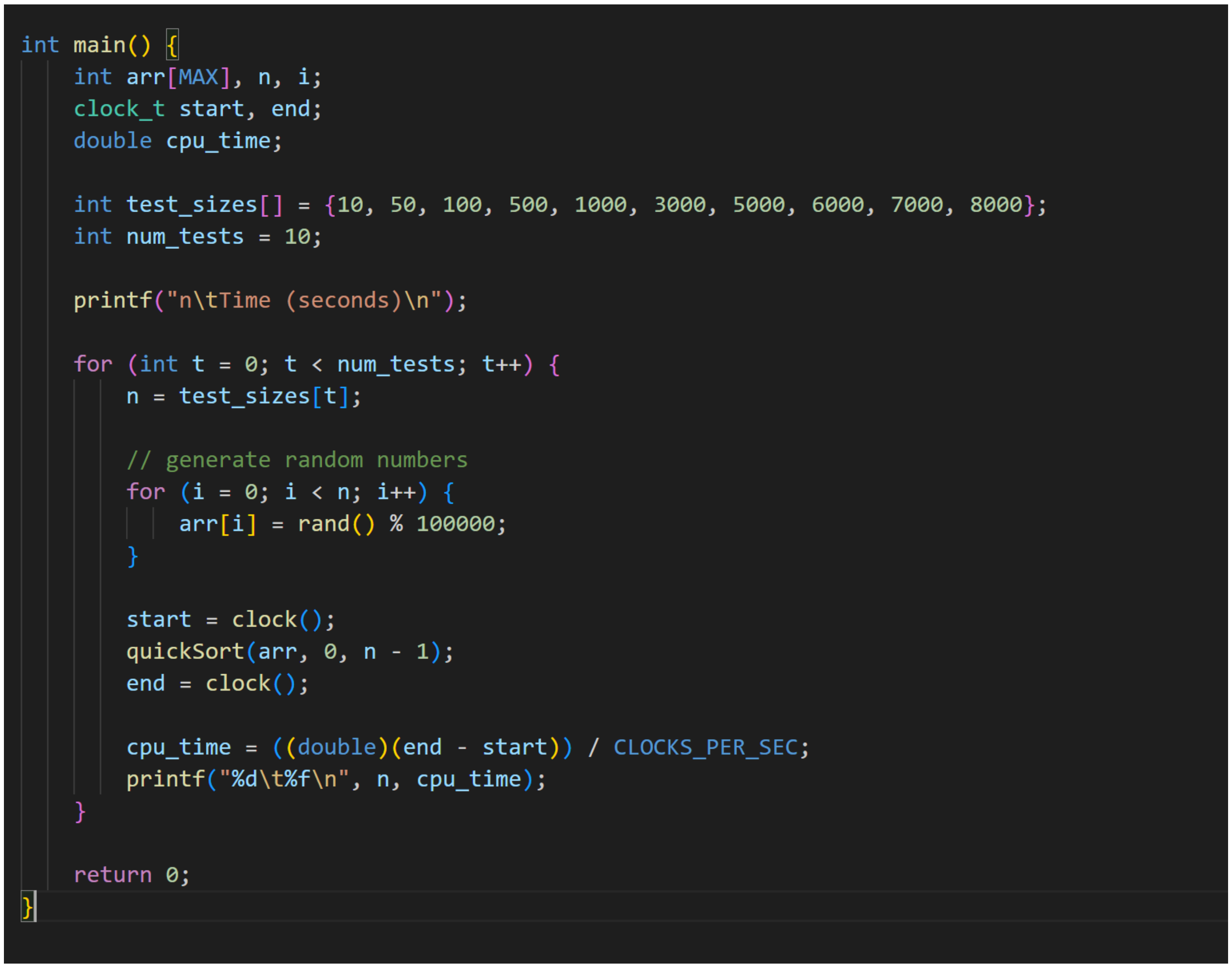
```
FUNCTION Partition(arr, low, high)
    pivot ← arr[high]          // choose last element as pivot
    i ← low - 1

    FOR j ← low TO high - 1
        IF arr[j] ≤ pivot THEN
            i ← i + 1
      SWAP arr[i] and arr[j]
    END IF
  END FOR

  SWAP arr[i + 1] and arr[high]
  RETURN (i + 1)
END FUNCTION


FUNCTION QuickSort(arr, low, high)
  IF low < high THEN
        pi ← Partition(arr, low, high)

    QuickSort(arr, low, pi - 1)   // left half
    QuickSort(arr, pi + 1, high)  // right half
  END IF
END FUNCTION


MAIN PROGRAM
  PRINT "n   Time(seconds)"

  FOR n = 1000 TO 20000 STEP 3000
    CREATE array arr[1..n]

    FOR i = 0 TO n-1
            arr[i] ← RANDOM(0..100000)
    END FOR
```

```
        start_time ← CURRENT_CLOCK()
    CALL QuickSort(arr, 0, n-1)
        end_time ← CURRENT_CLOCK()

        time_taken ← (end_time - start_time)
    PRINT n, time_taken
END FOR
```

Code:

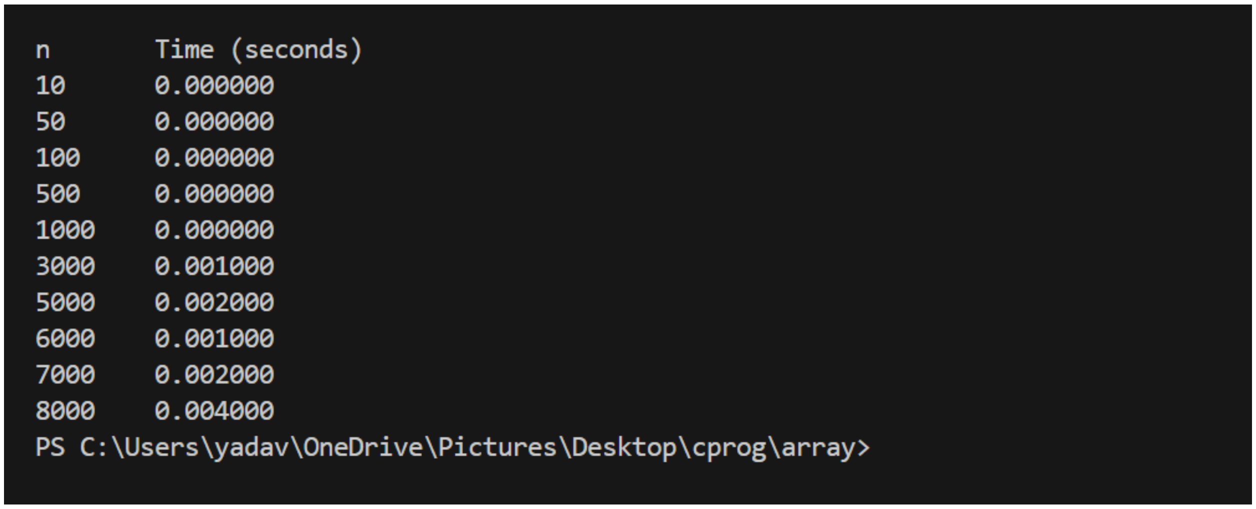


```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX 100000

// Partition function
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1, j, temp;

    for (j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    return (i + 1);
}

// QuickSort function
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```



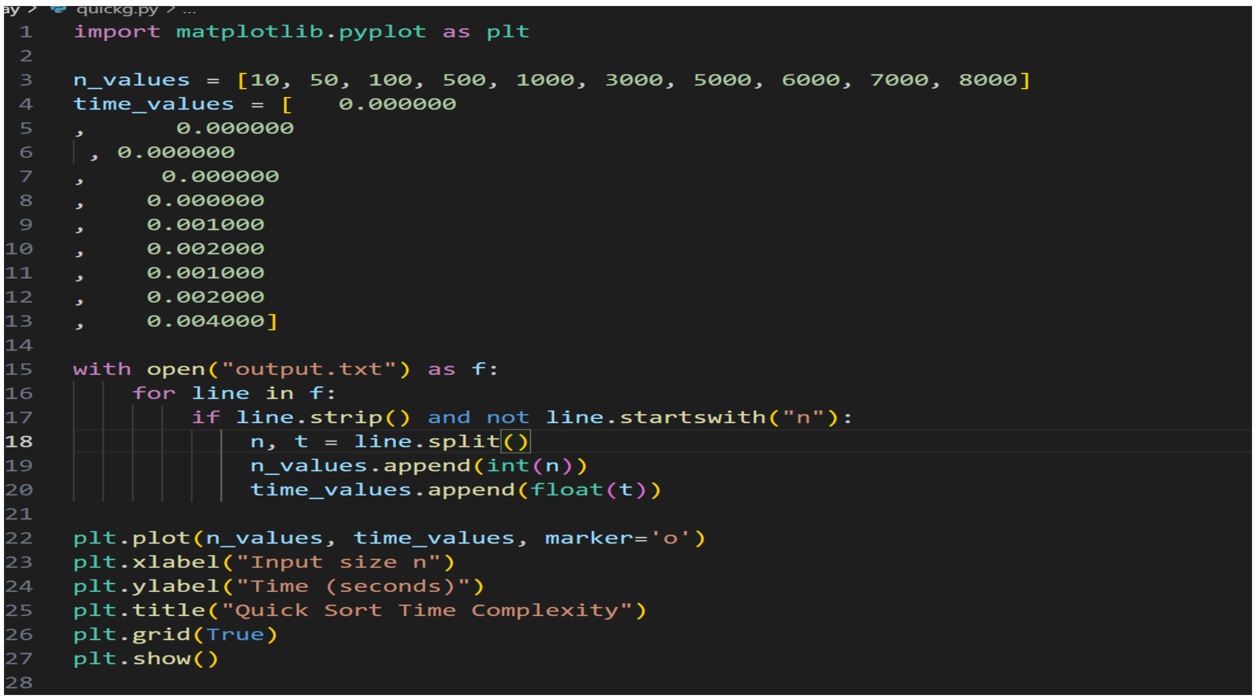
```c
int main() {
    int arr[MAX], n, i;
    clock_t start, end;
    double cpu_time;

    int test_sizes[] = {10, 50, 100, 500, 1000, 3000, 5000, 6000, 7000, 8000};
    int num_tests = 10;

    printf("n\tTime (seconds)\n");

    for (int t = 0; t < num_tests; t++) {
        n = test_sizes[t];

        // generate random numbers
        for (i = 0; i < n; i++) {
            arr[i] = rand() % 100000;
        }

        start = clock();
        quickSort(arr, 0, n - 1);
        end = clock();

        cpu_time = ((double)(end - start)) / CLOCKS_PER_SEC;
        printf("%d\t%f\n", n, cpu_time);
    }

    return 0;
}
```

OUTPUT:



```
n       Time (seconds)
10      0.000000
50      0.000000
100     0.000000
500     0.000000
1000    0.000000
3000    0.001000
5000    0.002000
6000    0.001000
7000    0.002000
8000    0.004000
PS C:\Users\yadav\OneDrive\Pictures\Desktop\cprog\array>
```

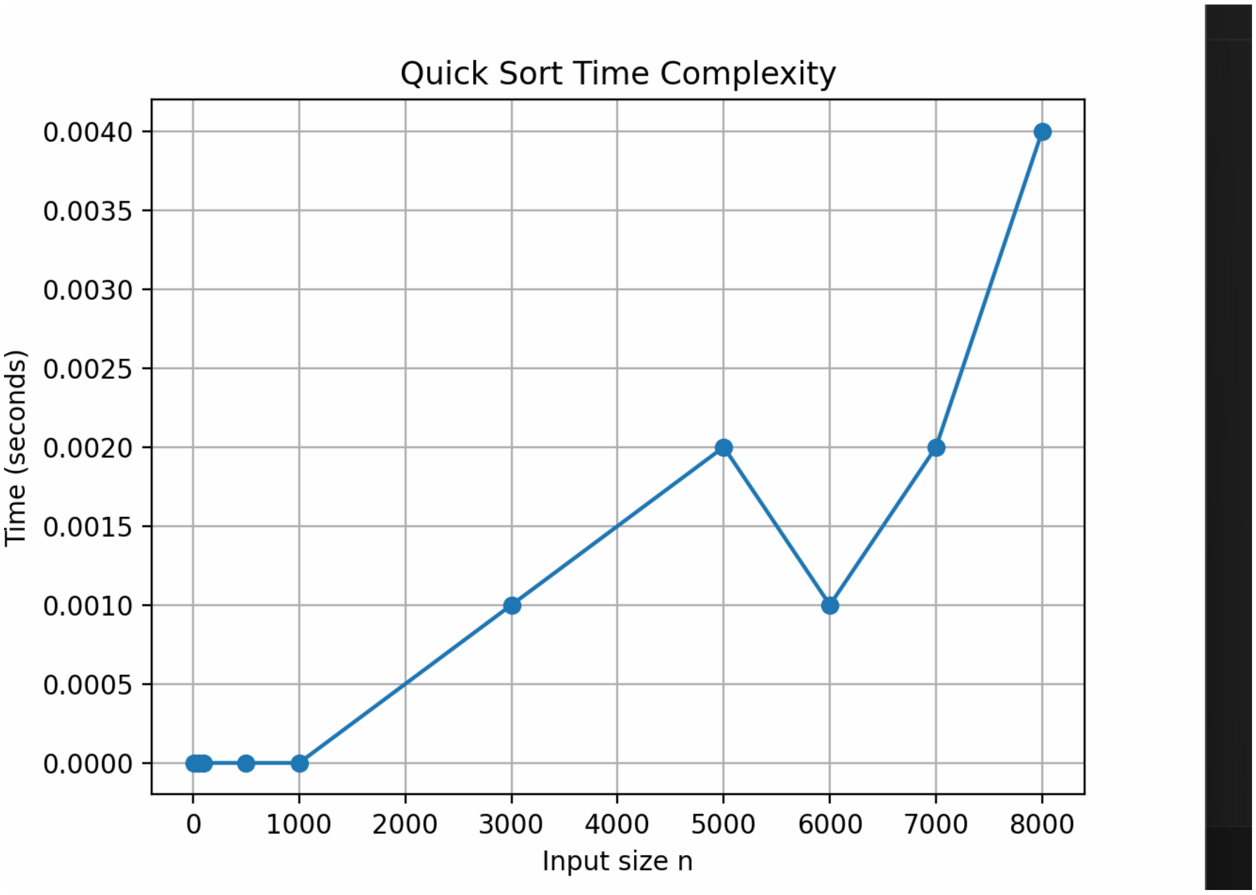
PYTHON CODE:



ay > quickg.py > ...

```python
import matplotlib.pyplot as plt

n_values = [10, 50, 100, 500, 1000, 3000, 5000, 6000, 7000, 8000]
time_values = [    0.000000
,       0.000000
 , 0.000000
,      0.000000
,     0.000000
,     0.001000
,     0.002000
,     0.001000
,     0.002000
,     0.004000]

with open("output.txt") as f:
    for line in f:
        if line.strip() and not line.startswith("n"):
            n, t = line.split()
            n_values.append(int(n))
            time_values.append(float(t))

plt.plot(n_values, time_values, marker='o')
plt.xlabel("Input size n")
plt.ylabel("Time (seconds)")
plt.title("Quick Sort Time Complexity")
plt.grid(True)
plt.show()
```

GRAPH:

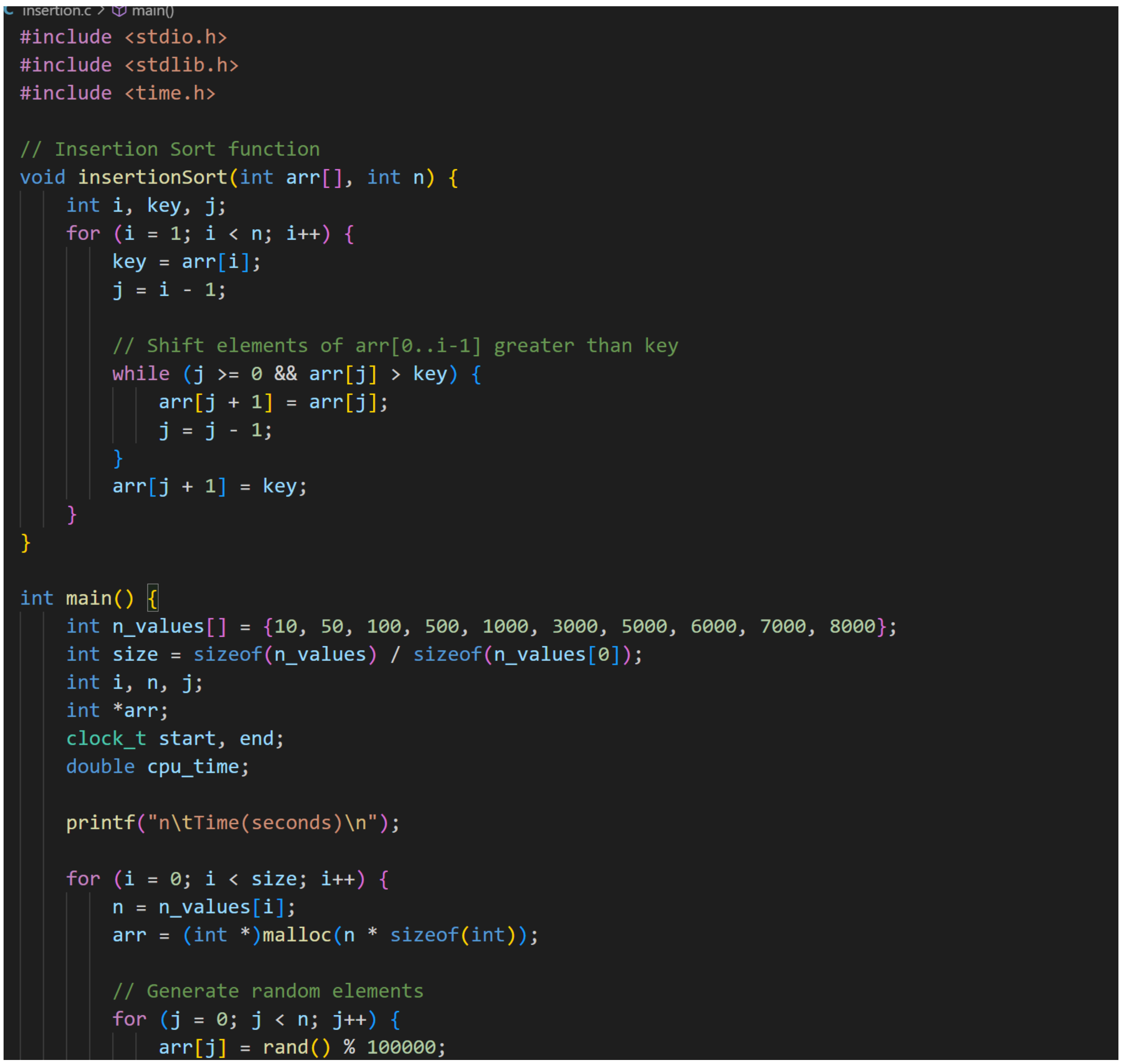

Quick Sort Time Complexity
Time (seconds)
0.0040
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
1000
2000
3000
4000
5000
6000
7000
8000
Input size n

(C)Design and implement C Program to sort a given set of n integer elements using Insertion Sort method and compute its time complexity. Run the program for varied values of n, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

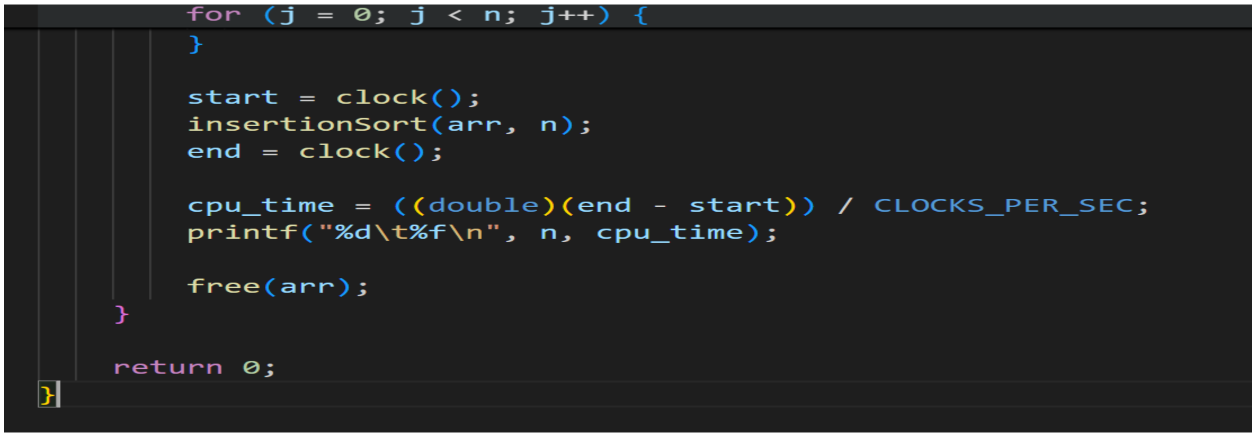
PSEUDOCODE:

```
BEGIN

FUNCTION InsertionSort(arr, n)
    FOR i ← 1 TO n-1 DO
        key ← arr[i]
        j ← i - 1
        WHILE j ≥ 0 AND arr[j] > key DO
            arr[j+1] ← arr[j]
            j ← j - 1
    END WHILE
        arr[j+1] ← key
  END FOR
END FUNCTION

MAIN
    n_values ← {10, 50, 100, 500, 1000, 3000, 5000, 6000, 7000, 8000}
  PRINT "n   Time(seconds)"

  FOR each n in n_values DO
    CREATE array arr[1..n]
        FOR i ← 0 TO n-1 DO
            arr[i] ← RANDOM(0..100000)
    END FOR

        start_time ← CURRENT_CLOCK()
    CALL InsertionSort(arr, n)
        end_time ← CURRENT_CLOCK()

        time_taken ← (end_time - start_time) / CLOCK_TICKS_PER_SEC
    PRINT n, time_taken
  END FOR
END
```

CODE:

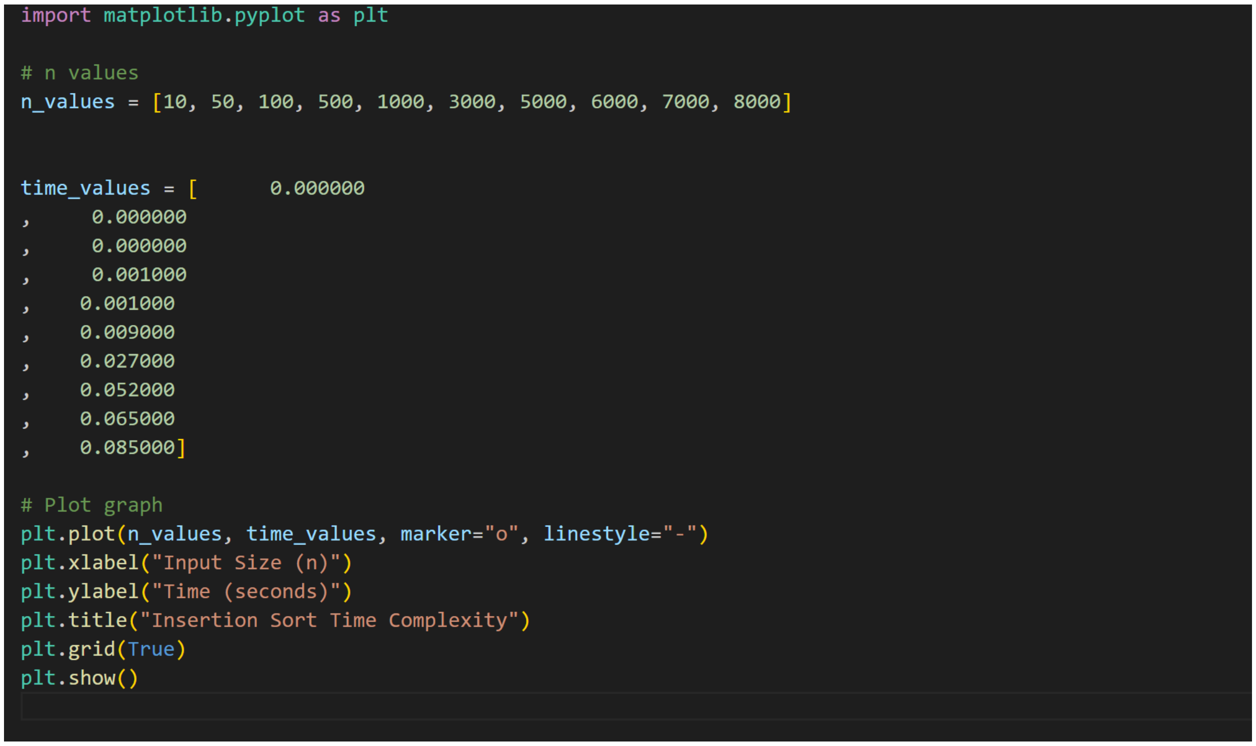


```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Insertion Sort function
void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        // Shift elements of arr[0..i-1] greater than key
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

int main() {
    int n_values[] = {10, 50, 100, 500, 1000, 3000, 5000, 6000, 7000, 8000};
    int size = sizeof(n_values) / sizeof(n_values[0]);
    int i, n, j;
    int *arr;
    clock_t start, end;
    double cpu_time;

    printf("n\tTime(seconds)\n");

    for (i = 0; i < size; i++) {
        n = n_values[i];
        arr = (int *)malloc(n * sizeof(int));

        // Generate random elements
        for (j = 0; j < n; j++) {
            arr[j] = rand() % 100000;
```



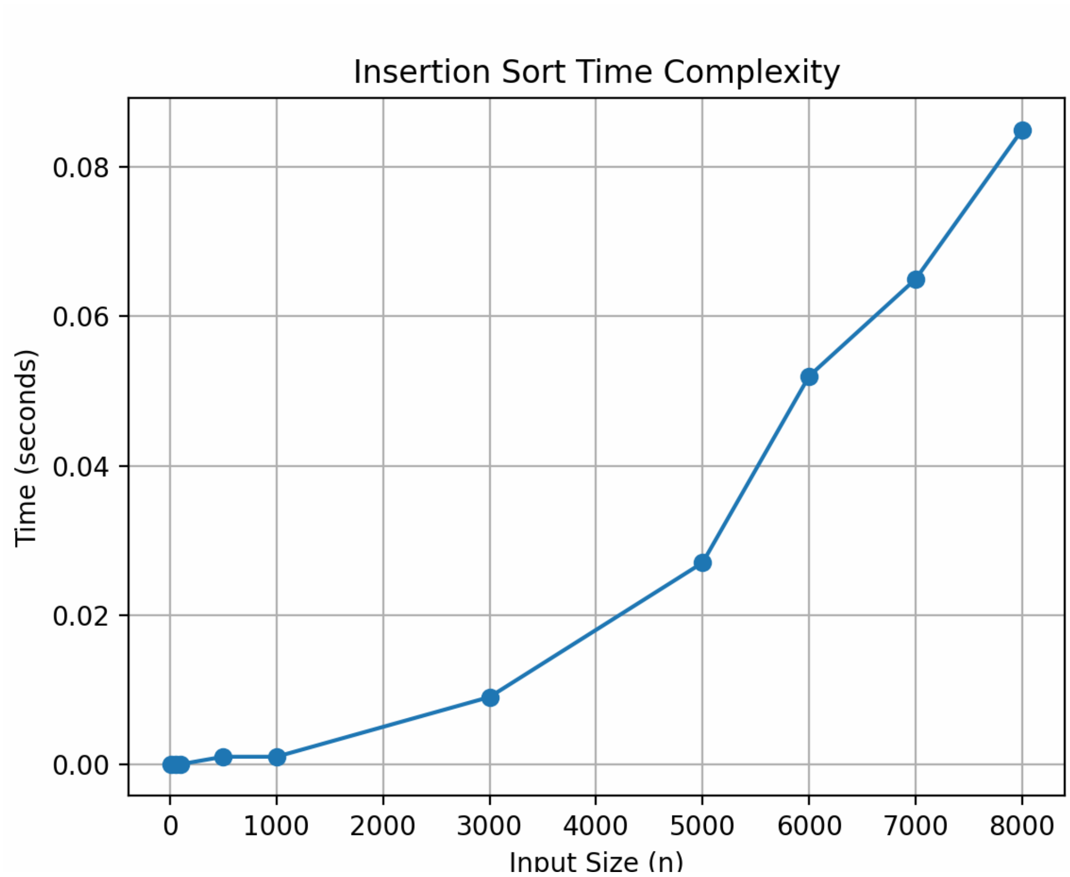
```c
        for (j = 0; j < n; j++) {
        }

        start = clock();
        insertionSort(arr, n);
        end = clock();

        cpu_time = ((double)(end - start)) / CLOCKS_PER_SEC;
        printf("%d\t%f\n", n, cpu_time);

        free(arr);
    }

    return 0;
}
```

PYTHON CODE:



```python
import matplotlib.pyplot as plt

# n values
n_values = [10, 50, 100, 500, 1000, 3000, 5000, 6000, 7000, 8000]


time_values = [      0.000000
,     0.000000
,     0.000000
,     0.001000
,    0.001000
,    0.009000
,    0.027000
,    0.052000
,    0.065000
,    0.085000]

# Plot graph
plt.plot(n_values, time_values, marker="o", linestyle="-")
plt.xlabel("Input Size (n)")
plt.ylabel("Time (seconds)")
plt.title("Insertion Sort Time Complexity")
plt.grid(True)
plt.show()
```

GRAPH:



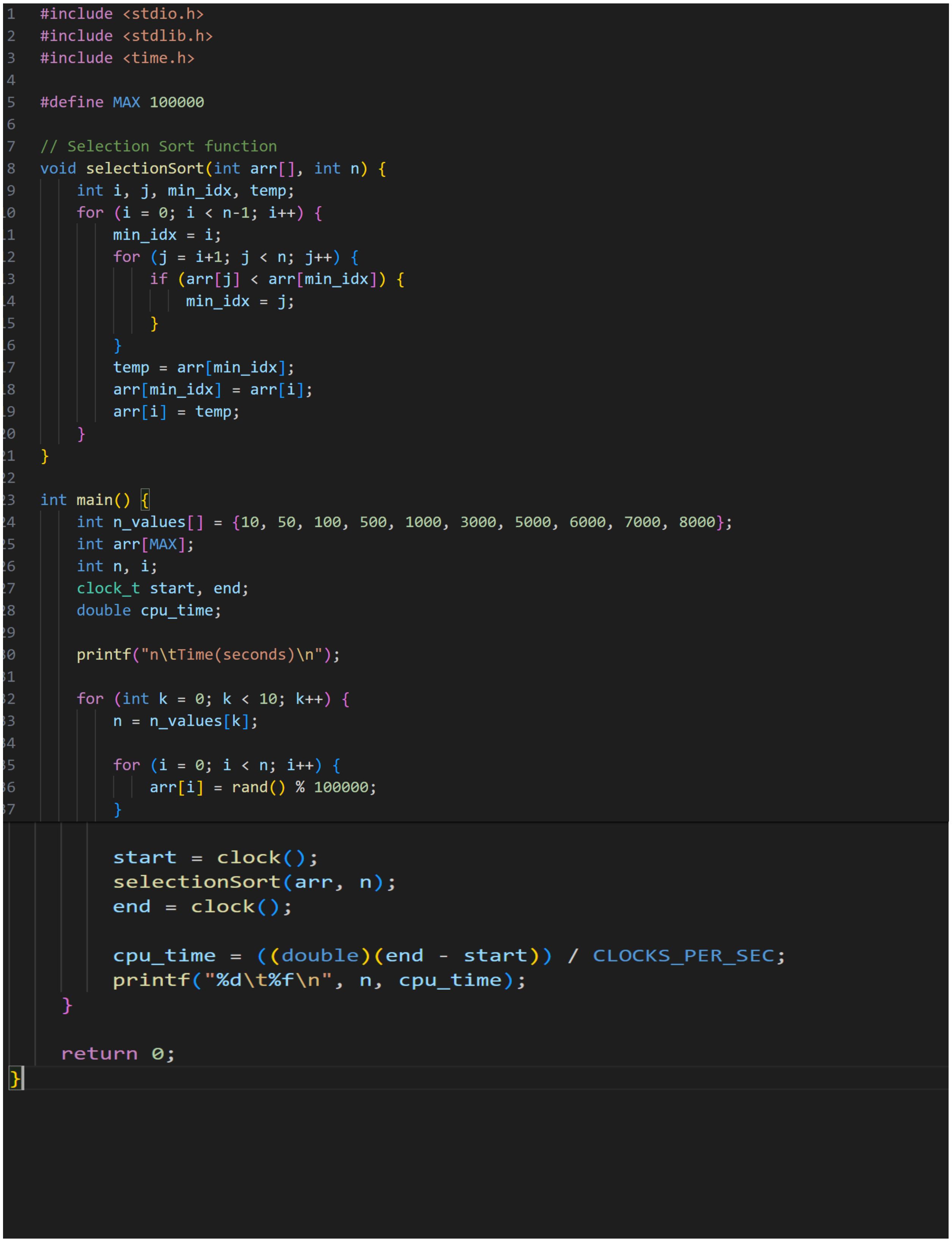
```
7000,
"_")
```

2(d) Design and implement C Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

PSEUDOCODE:

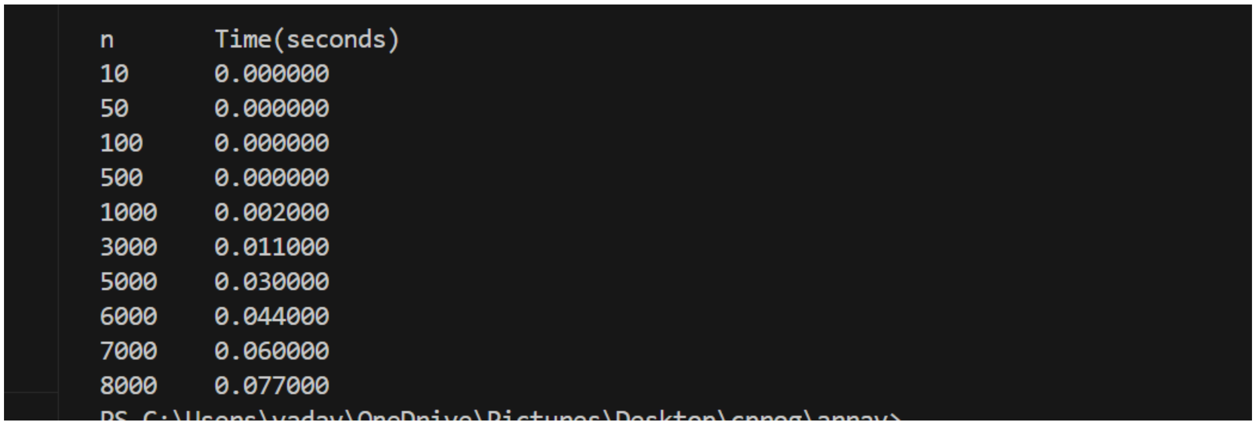
```
BEGIN

FUNCTION SelectionSort(arr, n)
    FOR i ← 0 TO n-2 DO
        min_idx ← i
        FOR j ← i+1 TO n-1 DO
            IF arr[j] < arr[min_idx] THEN
                min_idx ← j
            END IF
        END FOR
        SWAP arr[min_idx], arr[i]
    END FOR
END FUNCTION

MAIN PROGRAM
  INPUT n_values = [10, 50, 100, 500, 1000, 3000, 5000, 6000, 7000, 8000]

  FOR each n in n_values DO
    CREATE array arr[1..n]
    FOR i = 0 TO n-1 DO
            arr[i] ← RANDOM(0..100000)
    END FOR

        start_time ← CURRENT_CLOCK()
    CALL SelectionSort(arr, n)
        end_time ← CURRENT_CLOCK()

        time_taken ← (end_time - start_time) / CLOCK_TICKS_PER_SEC
    PRINT n, time_taken
  END FOR
END
```

CODE:



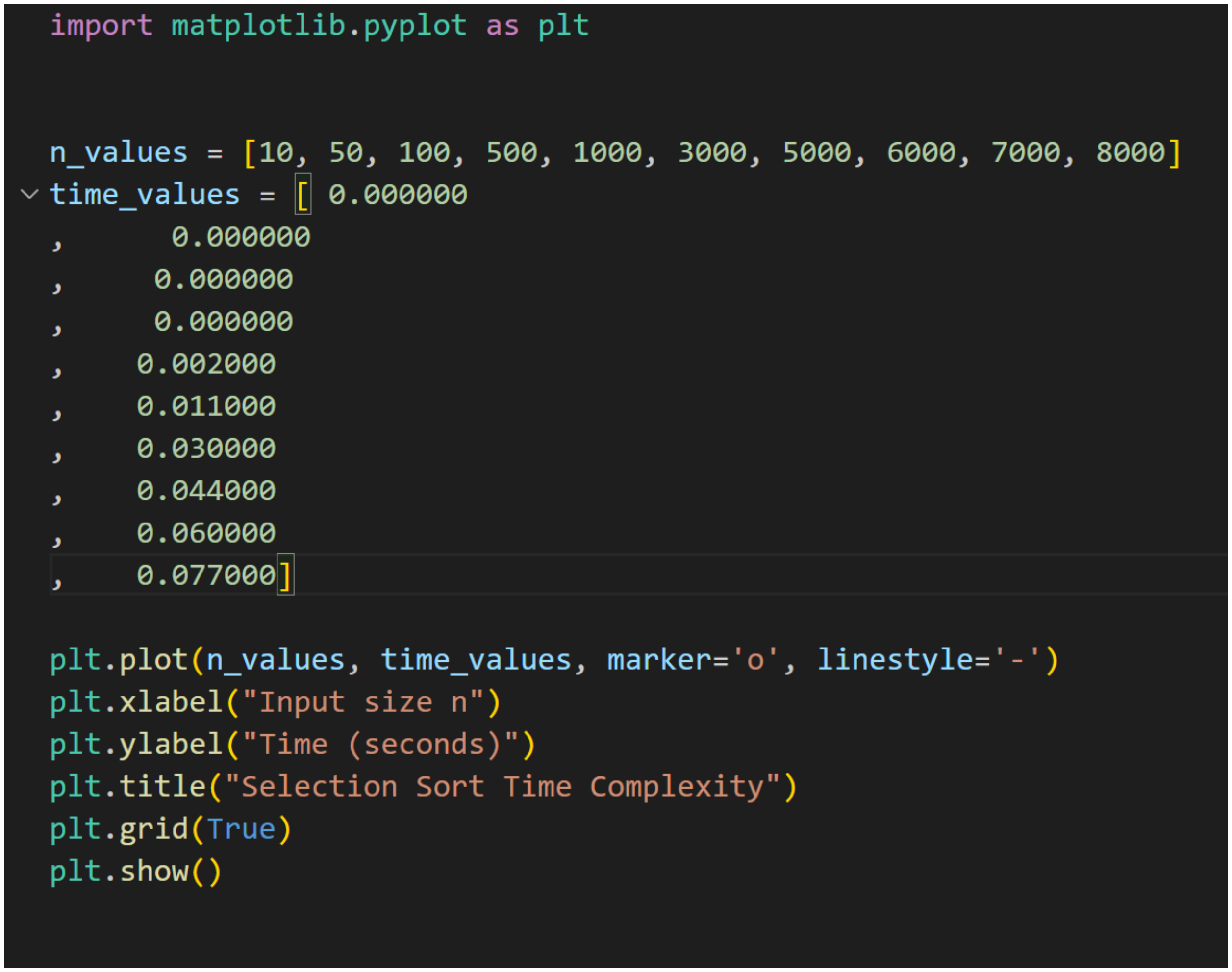
```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX 100000

// Selection Sort function
void selectionSort(int arr[], int n) {
    int i, j, min_idx, temp;
    for (i = 0; i < n-1; i++) {
        min_idx = i;
        for (j = i+1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }
        temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}

int main() {
    int n_values[] = {10, 50, 100, 500, 1000, 3000, 5000, 6000, 7000, 8000};
    int arr[MAX];
    int n, i;
    clock_t start, end;
    double cpu_time;

    printf("n\tTime(seconds)\n");

    for (int k = 0; k < 10; k++) {
        n = n_values[k];

        for (i = 0; i < n; i++) {
            arr[i] = rand() % 100000;
        }

        start = clock();
        selectionSort(arr, n);
        end = clock();

        cpu_time = ((double)(end - start)) / CLOCKS_PER_SEC;
        printf("%d\t%f\n", n, cpu_time);
    }

    return 0;
}
```

OUTPUT:



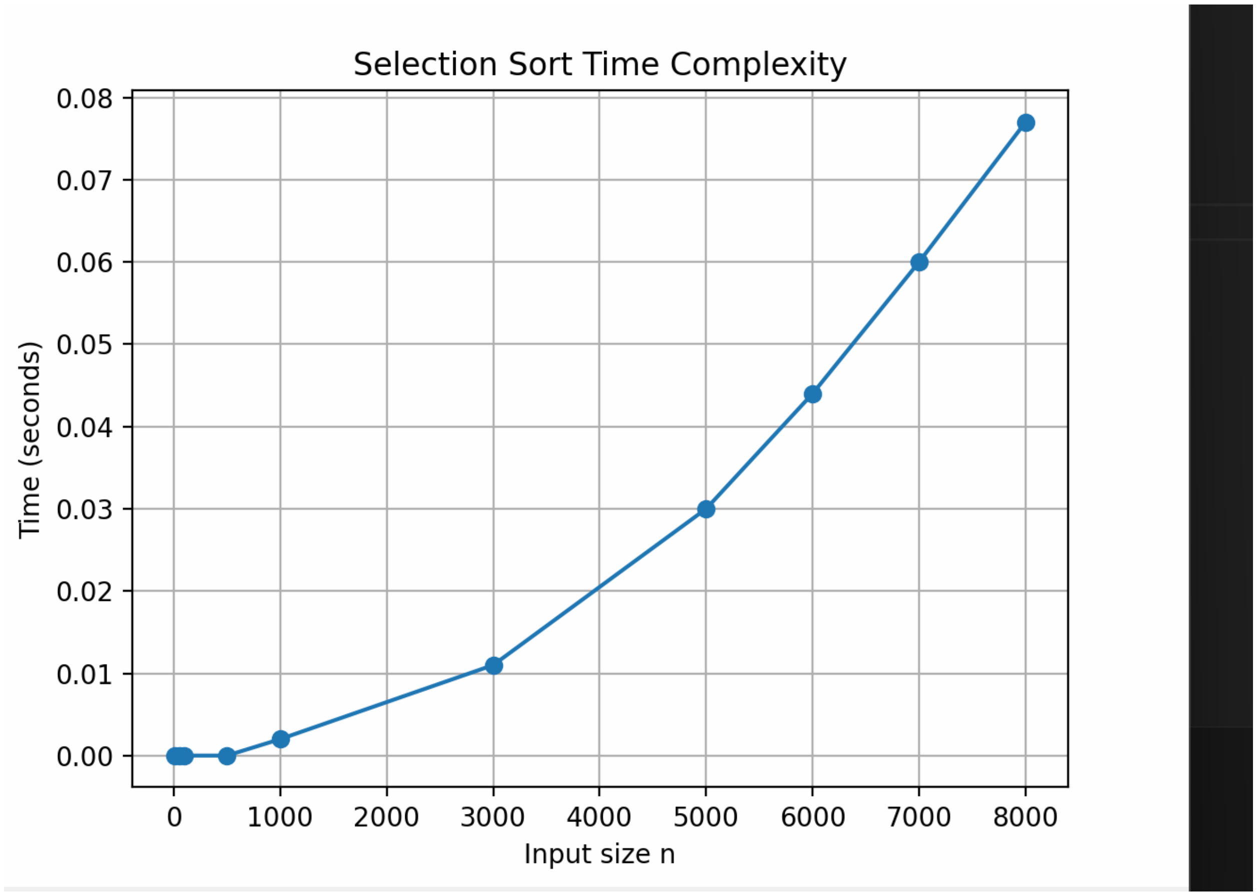
```
n       Time(seconds)
10      0.000000
50      0.000000
100     0.000000
500     0.000000
1000    0.002000
3000    0.011000
5000    0.030000
6000    0.044000
7000    0.060000
8000    0.077000
PS C:\Users\yadav\OneDrive\Pictures\Desktop\cprog\array>
```

PYTHON CODE:



```python
import matplotlib.pyplot as plt


n_values = [10, 50, 100, 500, 1000, 3000, 5000, 6000, 7000, 8000]
time_values = [ 0.000000
,       0.000000
,      0.000000
,      0.000000
,     0.002000
,     0.011000
,     0.030000
,     0.044000
,     0.060000
,     0.077000]

plt.plot(n_values, time_values, marker='o', linestyle='-')
plt.xlabel("Input size n")
plt.ylabel("Time (seconds)")
plt.title("Selection Sort Time Complexity")
plt.grid(True)
plt.show()
```

GRAPH:



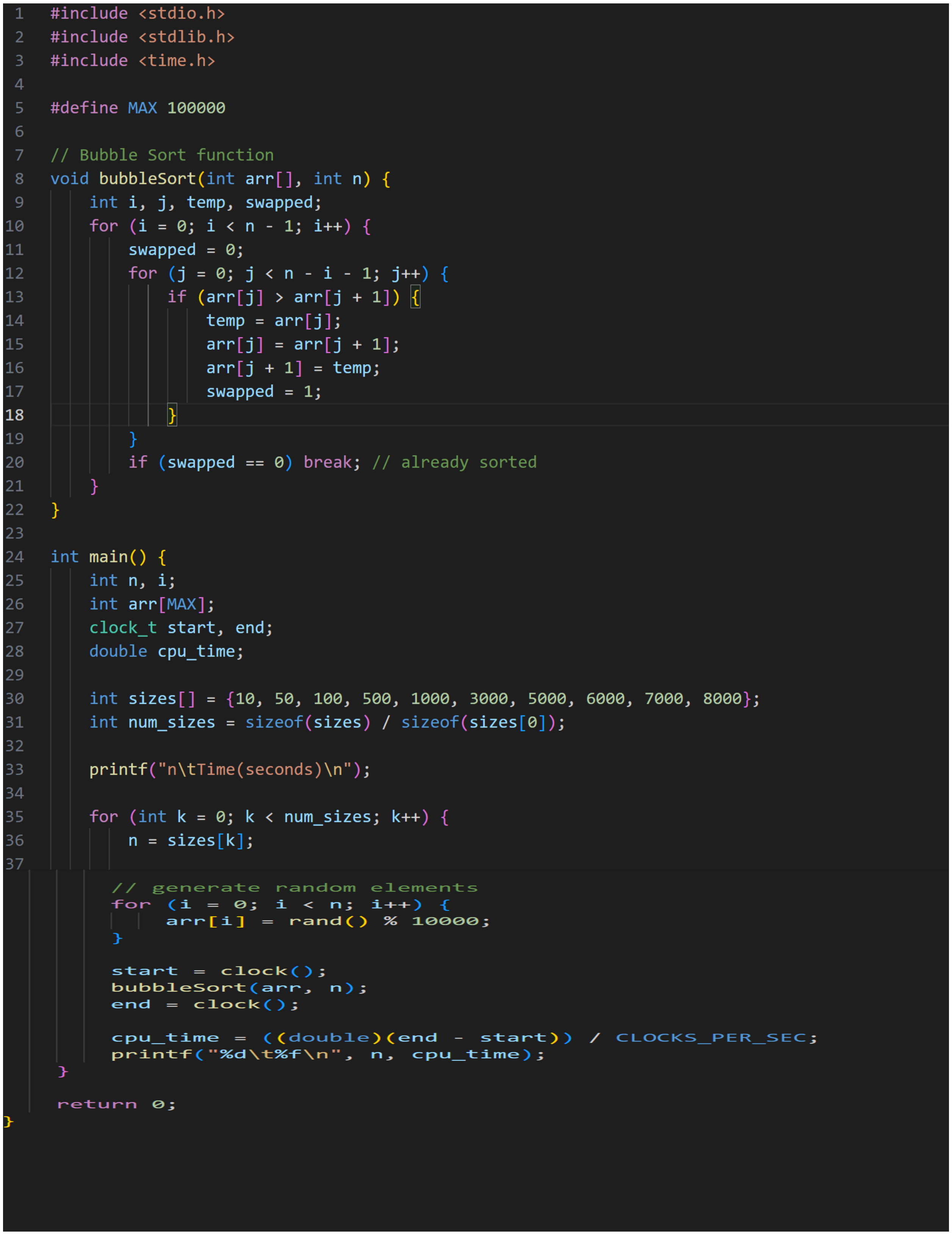
2 (e)Design and implement C Program to sort a given set of n integer elements using Bubble Sort method and compute its time complexity. Run the program for varied values of n, and

record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

PSEUDOCODE:

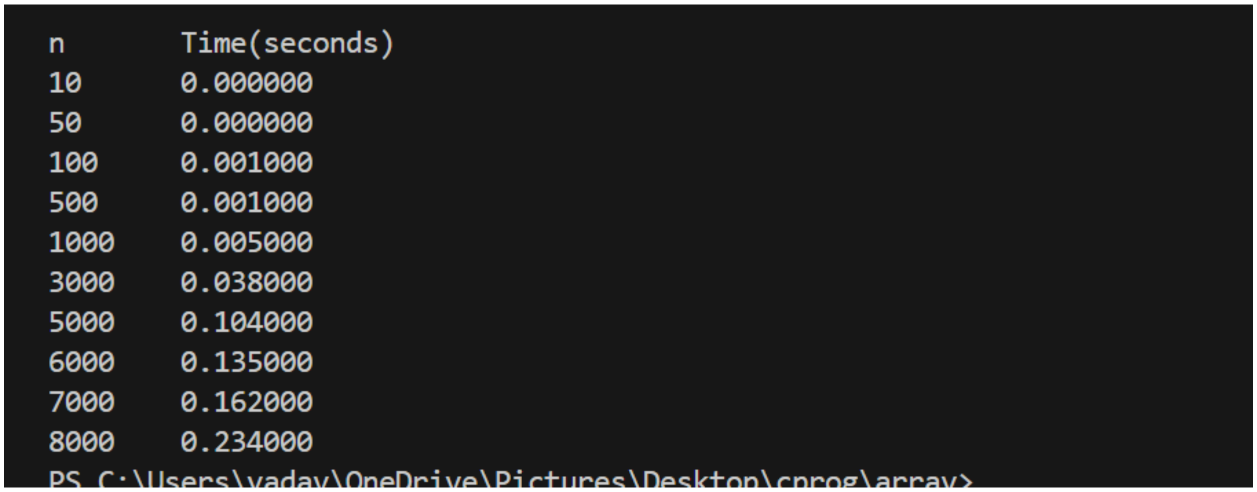
```
BEGIN

FUNCTION BubbleSort(arr, n)
    FOR i ← 0 TO n-2 DO
        swapped ← FALSE
        FOR j ← 0 TO n-i-2 DO
            IF arr[j] > arr[j+1] THEN
                SWAP arr[j], arr[j+1]
                swapped ← TRUE
            END IF
        END FOR
        IF swapped = FALSE THEN
            BREAK
        END IF
    END FOR
END FUNCTION

MAIN
    sizes = [10, 50, 100, 500, 1000, 3000, 5000, 6000, 7000, 8000]

    FOR each n in sizes DO
        CREATE array of size n with RANDOM numbers
        start_time ← CLOCK()
        CALL BubbleSort(array, n)
        end_time ← CLOCK()
        time_taken ← (end_time - start_time) / CLOCK_TICKS_PER_SEC
        PRINT n, time_taken
    END FOR
END
```

CODE:



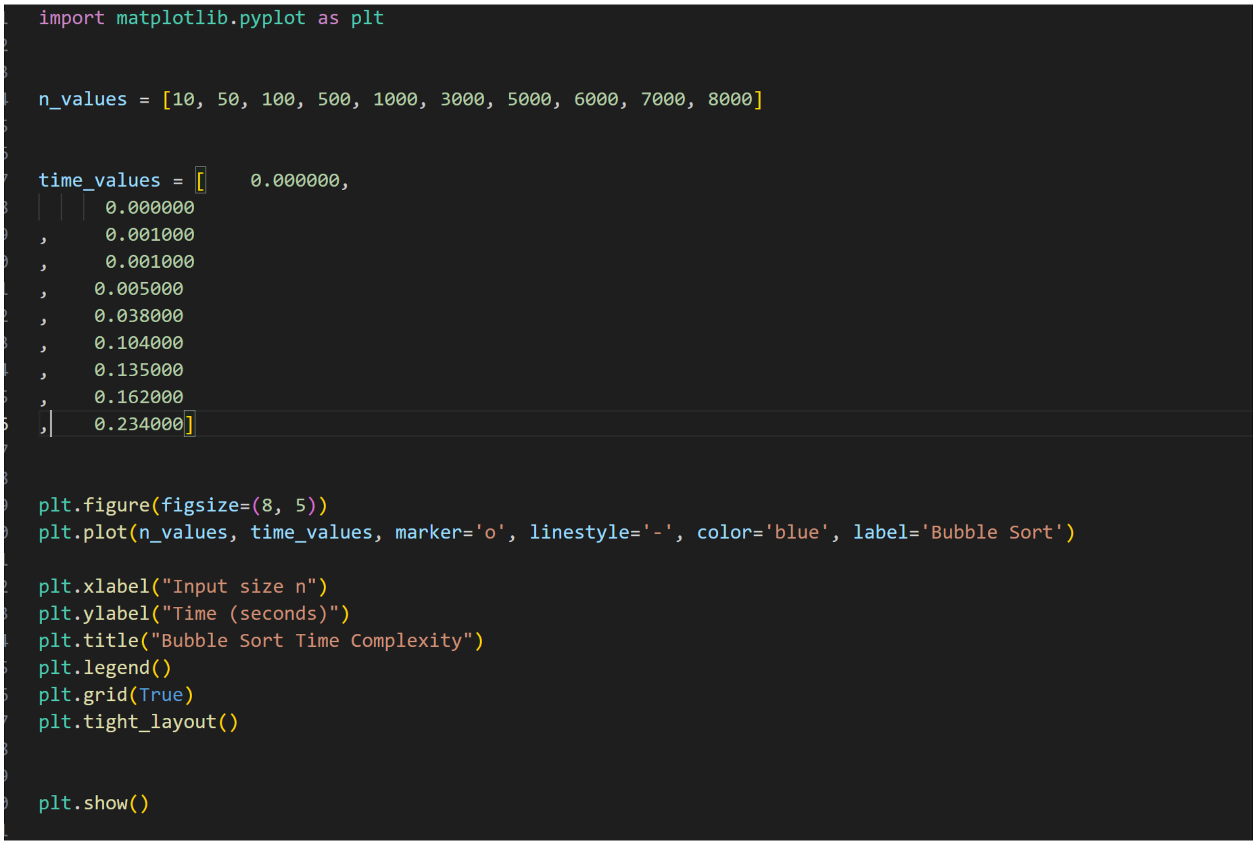
```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX 100000

// Bubble Sort function
void bubbleSort(int arr[], int n) {
    int i, j, temp, swapped;
    for (i = 0; i < n - 1; i++) {
        swapped = 0;
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = 1;
            }
        }
        if (swapped == 0) break; // already sorted
    }
}

int main() {
    int n, i;
    int arr[MAX];
    clock_t start, end;
    double cpu_time;

    int sizes[] = {10, 50, 100, 500, 1000, 3000, 5000, 6000, 7000, 8000};
    int num_sizes = sizeof(sizes) / sizeof(sizes[0]);

    printf("n\tTime(seconds)\n");

    for (int k = 0; k < num_sizes; k++) {
        n = sizes[k];

        // generate random elements
        for (i = 0; i < n; i++) {
            arr[i] = rand() % 10000;
        }

        start = clock();
        bubbleSort(arr, n);
        end = clock();

        cpu_time = ((double)(end - start)) / CLOCKS_PER_SEC;
        printf("%d\t%f\n", n, cpu_time);
    }

    return 0;
}
```

OUTPUT:



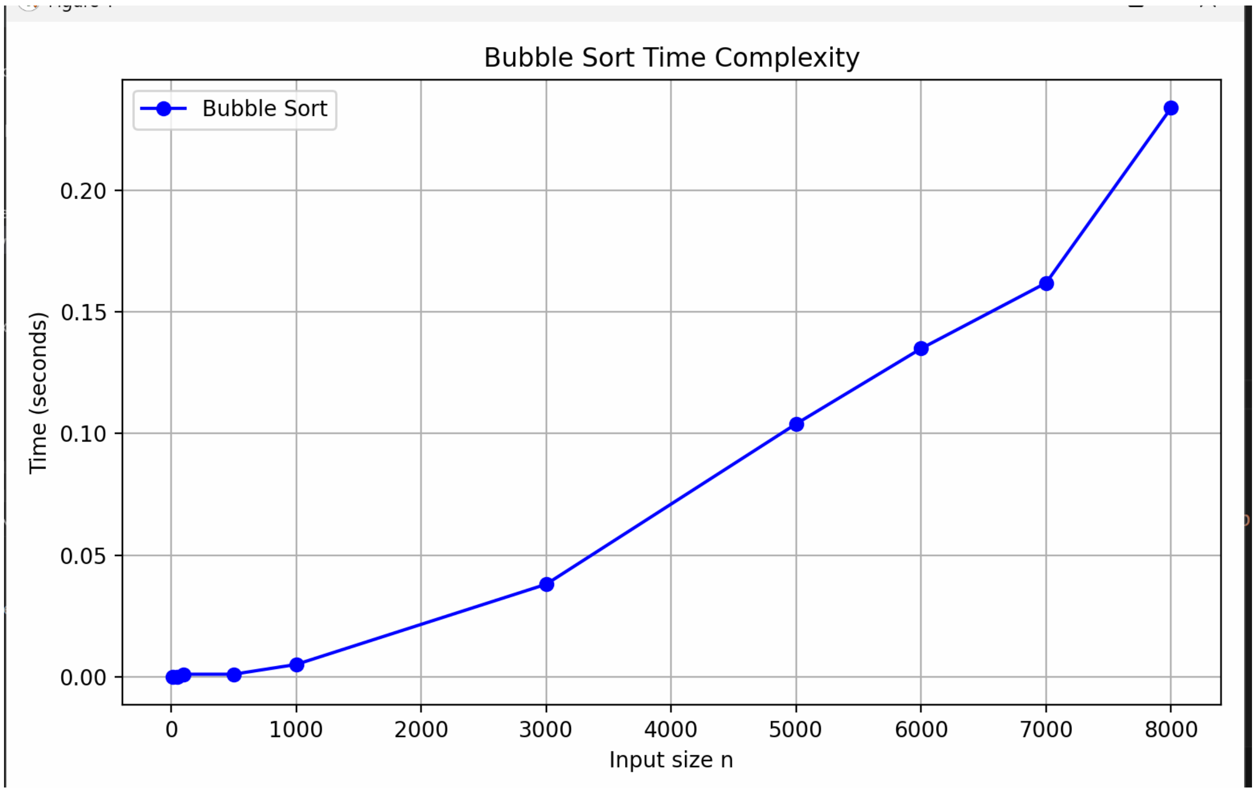
```
n        Time(seconds)
10       0.000000
50       0.000000
100      0.001000
500      0.001000
1000     0.005000
3000     0.038000
5000     0.104000
6000     0.135000
7000     0.162000
8000     0.234000
PS C:\Users\yadav\OneDrive\Pictures\Desktop\cprog\array>
```

PYTHON CODE:



```python
import matplotlib.pyplot as plt


n_values = [10, 50, 100, 500, 1000, 3000, 5000, 6000, 7000, 8000]


time_values = [    0.000000,
      0.000000
,     0.001000
,     0.001000
,    0.005000
,    0.038000
,    0.104000
,    0.135000
,    0.162000
,    0.234000]


plt.figure(figsize=(8, 5))
plt.plot(n_values, time_values, marker='o', linestyle='-', color='blue', label='Bubble Sort')

plt.xlabel("Input size n")
plt.ylabel("Time (seconds)")
plt.title("Bubble Sort Time Complexity")
plt.legend()
plt.grid(True)
plt.tight_layout()


plt.show()
```

GRAPH:


Bubble Sort Time Complexity
Bubble Sort
Time (seconds)
0.00
0.05
0.10
0.15
0.20
0
1000
2000
3000
4000
5000
6000
7000
8000
Input size n