**TASK 1 : Sort a given set of n integer elements using Selection Sort method in C language and compute its time complexity. Run the program for varied values of n> 5000 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. Demonstrate how the brute force method works along with its time complexity analysis: worst case, average case and best case.**

### **Steps to Follow:**

1. **Implement Selection Sort in C.**
2. **Generate or read input data (random numbers or file input).**
3. **Measure sorting time using** clock() **function.**
4. **Run tests for varied values of** n > 5000**.**
5. **Analyze time complexity (best, worst, and average cases).**
6. **Plot a graph of** n **vs. time taken using Python or Excel.**
7. **Selection Sort Implementation in C**

#include <stdio.h>

#include <stdlib.h> // For rand() and srand()

#include <time.h> // For clock() and CLOCKS\_PER\_SEC

// Function to perform selection sort

void selectionSort(int arr[], int n)

{

int i, j, min\_idx;

// One by one move boundary of unsorted subarray

for (i = 0; i < n - 1; i++)

{

// Find the minimum element in the unsorted array

min\_idx = i;

for (j = i + 1; j < n; j++)

{

if (arr[j] < arr[min\_idx])

{

min\_idx = j;

}

}

// Swap the found minimum element with the first element

int temp = arr[min\_idx];

arr[min\_idx] = arr[i];

arr[i] = temp;

}

}

// Function to print an array (first 10 elements only)

void printArray(int arr[], int size)

{

int i;

for (i = 0; i < 10; i++)

{ // Print only the first 10 elements

printf("%d ", arr[i]);

}

printf("\n");

}

int main()

{

int n;

printf("Enter the number of elements in the array (>= 5000): ");

scanf("%d", &n);

if (n < 5000)

{

printf("Please enter a number greater than or equal to 5000.\n");

return 1; // Exit the program

}

int \*arr = (int \*)malloc(n \* sizeof(int)); // Dynamically allocate memory for the array

// Seed the random number generator

srand(time(0));

// Fill the array with random numbers

for (int i = 0; i < n; i++)

{

arr[i] = rand() % 100000; // Random numbers between 0 and 99999

}

printf("Original array (first 10 elements): \n");

printArray(arr, n);

// Measure the time taken by selection sort

clock\_t start = clock(); // Start the clock

selectionSort(arr, n); // Perform selection sort

clock\_t end = clock(); // End the clock

double time\_taken = ((double)(end - start)) / CLOCKS\_PER\_SEC; // Calculate time in seconds

printf("Sorted array (first 10 elements): \n");

printArray(arr, n); // Print only the first 10 elements for brevity

printf("Time taken by Selection Sort: %f seconds\n", time\_taken);

free(arr); // Free dynamically allocated memory

return 0;

}

**OUTPUT**

Enter the number of elements in the array (>= 5000): 50000

Original array (first 10 elements):

63097 21655 20786 78214 5362 61840 63677 68340 78958 21352

Sorted array (first 10 elements):

0 1 1 6 8 8 10 16 16 19

Time taken by Selection Sort: 2.288079 seconds

### ****Time Complexity Analysis****

Selection Sort time complexity for different cases:

* **Best Case (Already Sorted):** O(n^2)
* **Worst Case (Reverse Sorted):** O(n^2)
* **Average Case (Random Data):** O(n^2)

Even in the best case, the algorithm performs n(n−1)/2 comparisons.

1. **Plotting the Graph using Excel**

### ****Step 1: Prepare Your Data****

First, create a table in Excel with two columns:

| **Array Size (n)** | **Time Taken(seconds)** |
| --- | --- |
| 5000 | 0.14 |
| 10000 | 0.58 |
| 15000 | 1.30 |
| 20000 | 2.64 |

### ****Step 2: Select the Data****

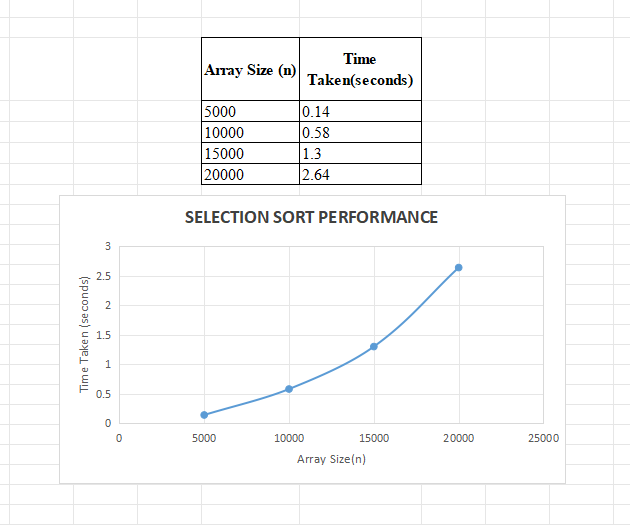
1. Open Excel and enter the data as shown in Step 1.
2. Select the two columns (including headers).

### ****Step 3: Insert a Chart****

1. Go to the **"Insert"** tab on the ribbon.
2. In the **Charts** section, choose **"Scatter Plot"** (recommended for performance data).
   1. Select **"Scatter with Smooth Lines"** to visualize trends better.

### ****Step 4: Customize the Chart****

1. Add axis labels and chart title:
   1. X-axis: Array Size (n)
   2. Y-axis: Time Taken (seconds)
   3. Chart title: Selection Sort Performance
2. Adjust colors and styles as needed.



### ****4. Explanation of the Brute Force Approach****

Selection Sort is considered a brute force method because:

1. It scans the entire unsorted portion of the array to find the minimum element.
2. Even if the array is sorted, it performs all comparisons.
3. It does not take advantage of any structure in the data, unlike more optimized algorithms.