Lab Practical 2: Sorting

Aaryan gupta 24293916141 (CSE B)

2(a) OBJECTIVE:

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.

Algorithm: Merge Sort

1. Start
2. If length of array > 1 then
   a. Find middle point: mid = (low + high) / 2
   b. Divide array into two halves: left = A[low...mid], right = A[mid+1...high]
   c. Recursively apply MergeSort(A, low, mid) to the left subarray
   d. Recursively apply MergeSort(A, mid+1, high) to the right subarray
   e. Call Merge(A, low, mid, high) to combine the sorted subarrays
3. End If
4. Stop

PSEUDO CODE:

```
MergeSort(A)
if length(A) > 1
    mid = length(A) // 2
    L = A[0..mid-1]
    R = A[mid..end]
    MergeSort(L)
    MergeSort(R)
    Merge(L, R, A)
Merge(L, R, A)
    p = q = r = 0
    while p < length(L) and q < length(R)
```

Aaryan gupta 24293916141 (CSE B)

```
        if L[p] <= R[q]
            A[r] = L[p]
            p = p + 1
        else
            A[r] = R[q]
            q = q + 1
        r = r + 1
    while p < length(L)
        A[r] = L[p]
        p = p + 1
        r = r + 1
    while q < length(R)
        A[r] = R[q]
        q = q + 1
        r = r + 1
```

CODE:

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

void combine(int arr[], int low, int mid, int high) {
    int i, j, k;
    int n1 = mid - low + 1;
    int n2 = high - mid;
    int *L = (int*)malloc(n1 * sizeof(int));
    int *R = (int*)malloc(n2 * sizeof(int));

    for(i = 0; i < n1; i++)
        L[i] = arr[low + i];
    for(j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    i = 0, j = 0, k = low;
    while(i < n1 && j < n2) {
        if(L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
```

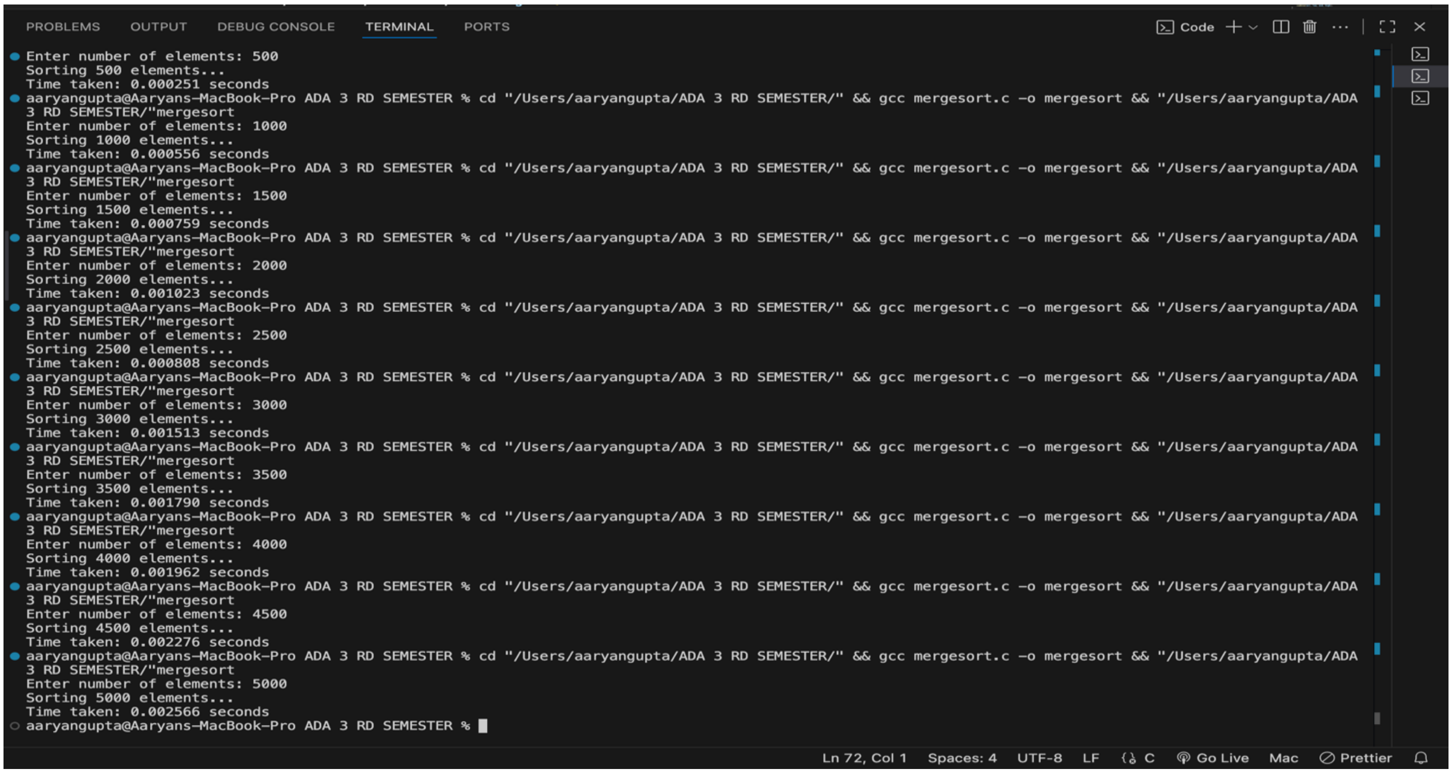
```
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while(i < n1) {
        arr[k] = L[i];
        i++; k++;
    }
    while(j < n2) {
        arr[k] = R[j];
        j++; k++;
    }
    free(L); free(R);
}

void mergeSort(int arr[], int low, int high) {
    if(low < high) {
        int mid = low + (high - low) / 2;
        mergeSort(arr, low, mid);
        mergeSort(arr, mid + 1, high);
        combine(arr, low, mid, high);
    }
}

int main() {
    int n;
    srand(time(NULL));

    printf("Enter number of elements: ");
    scanf("%d", &n);

    int *arr = (int*)malloc(n * sizeof(int));

    for(int i = 0; i < n; i++)
        arr[i] = rand() % 10000;

    printf("Sorting %d elements...\n", n);

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

    double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("Time taken: %.6f seconds\n", time_taken);

    free(arr);
```

Lab Practical 2: Sorting

Aaryan gupta 24293916141 (CSE B)

```
    return 0;
}
```

Output



```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
Enter number of elements: 500
Sorting 500 elements...
Time taken: 0.000251 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc mergesort.c -o mergesort && "/Users/aaryangupta/ADA
3 RD SEMESTER/"mergesort
Enter number of elements: 1000
Sorting 1000 elements...
Time taken: 0.000556 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc mergesort.c -o mergesort && "/Users/aaryangupta/ADA
3 RD SEMESTER/"mergesort
Enter number of elements: 1500
Sorting 1500 elements...
Time taken: 0.000759 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc mergesort.c -o mergesort && "/Users/aaryangupta/ADA
3 RD SEMESTER/"mergesort
Enter number of elements: 2000
Sorting 2000 elements...
Time taken: 0.001023 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc mergesort.c -o mergesort && "/Users/aaryangupta/ADA
3 RD SEMESTER/"mergesort
Enter number of elements: 2500
Sorting 2500 elements...
Time taken: 0.000808 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc mergesort.c -o mergesort && "/Users/aaryangupta/ADA
3 RD SEMESTER/"mergesort
Enter number of elements: 3000
Sorting 3000 elements...
Time taken: 0.001513 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc mergesort.c -o mergesort && "/Users/aaryangupta/ADA
3 RD SEMESTER/"mergesort
Enter number of elements: 3500
Sorting 3500 elements...
Time taken: 0.001790 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc mergesort.c -o mergesort && "/Users/aaryangupta/ADA
3 RD SEMESTER/"mergesort
Enter number of elements: 4000
Sorting 4000 elements...
Time taken: 0.001962 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc mergesort.c -o mergesort && "/Users/aaryangupta/ADA
3 RD SEMESTER/"mergesort
Enter number of elements: 4500
Sorting 4500 elements...
Time taken: 0.002276 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc mergesort.c -o mergesort && "/Users/aaryangupta/ADA
3 RD SEMESTER/"mergesort
Enter number of elements: 5000
Sorting 5000 elements...
Time taken: 0.002566 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER %
Ln 72, Col 1   Spaces: 4   UTF-8   LF   { } C   Go Live   Mac   Prettier
```

Fig1 :Output

Python code for mergesort Graph:

```python
import matplotlib.pyplot as plt

n = [500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000]
time = [0.000251, 0.000556, 0.000759, 0.001023, 0.000808, 0.001513, 0.001790, 0.001962, 0.002276, 0.002566]

plt.figure(figsize=(10, 6))
plt.plot(n, time, 'bo-', linewidth=2, markersize=6)
plt.xlabel('Number of Elements (n)')
plt.ylabel('Time (seconds)')
plt.title('Merge Sort Time Complexity Analysis')
plt.grid(True, alpha=0.3)
plt.show()
```

Lab Practical 2: Sorting

Aaryan gupta 24293916141 (CSE B)

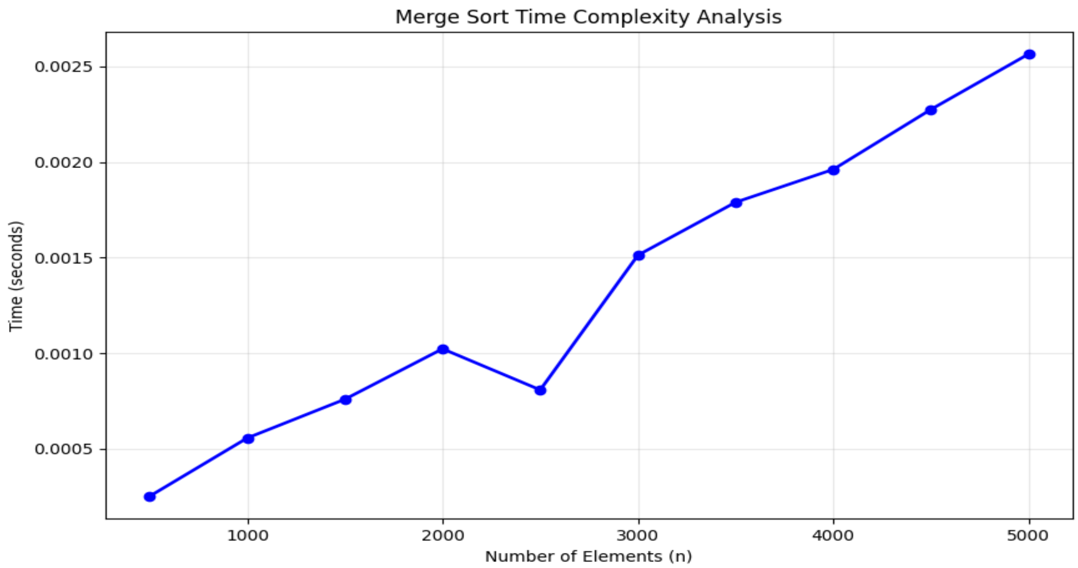


Fig2 :Graph Merge sort

2(b) OBJECTIVE:

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

Algorithm : Quick Sort

1. Start

2. Input: An array A with indices (start, end)

3. If start < end then

    a. Choose a pivot element (commonly the last element)

    b. Rearrange the array so that:

        - Elements smaller than pivot are placed before it

        - Elements greater than pivot are placed after it

    c. Let the pivot settle at its correct position (pIndex)

    d. Recursively apply Quick Sort to:

        - Left subarray: A[start ... pIndex - 1]

- Right subarray: A[pIndex + 1 ... end]

4. End If

5. Stop

PSEUDOCODE:

```
QuickSort(A, start, end)
  if start < end
    pIndex = Partition(A, start, end)
    QuickSort(A, start, pIndex - 1)
    QuickSort(A, pIndex + 1, end)
Partition(A, start, end)
  pivotElement = A[end]
  i = start - 1
  for j = start to end - 1
    if A[j] <= pivotElement
      i = i + 1
      swap A[i], A[j]
  swap A[i + 1], A[end]
  return i + 1
```

Code

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

void swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}

int partition(int arr[], int low, int high) {
```

Aaryan gupta 24293916141 (CSE B)

```
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

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);
    }
}

int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int *arr = (int*)malloc(n * sizeof(int));
    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 10000;
    }

    clock_t start, end;
    double time_taken;

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

    time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("Time taken to sort %d elements: %f seconds\n", n, time_taken);

    free(arr);
    return 0;
}
```

Lab Practical 2: Sorting

Aaryan gupta 24293916141 (CSE B)

Output:

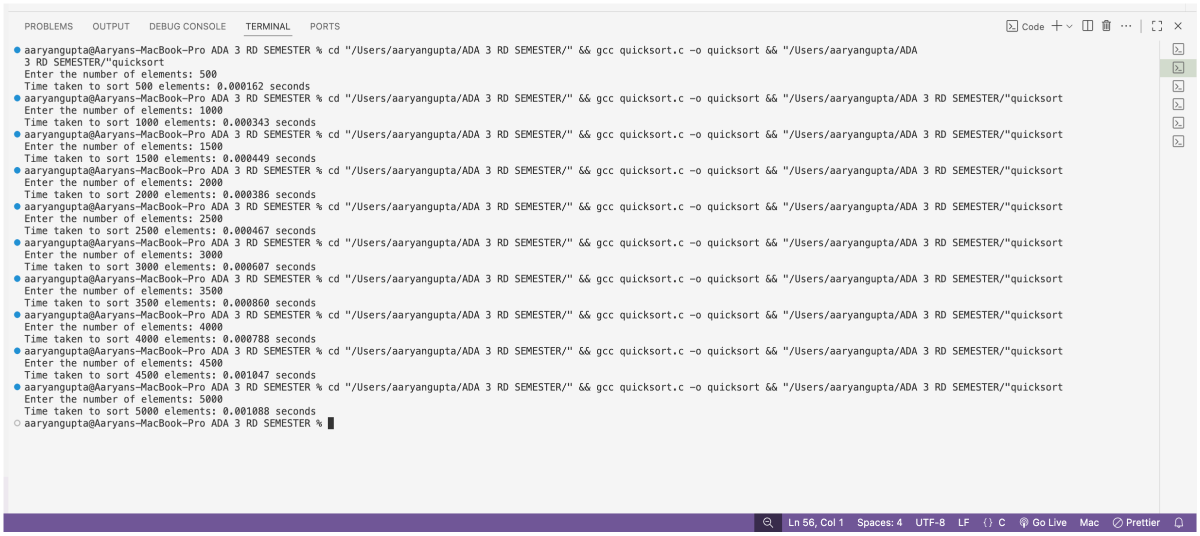


Fig3 :Output

Python code for Quick sort graph:

```python
import matplotlib.pyplot as plt

n = [500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000]
time = [0.000162, 0.000343, 0.000449, 0.000386, 0.000467,
        0.000607, 0.000860, 0.000788, 0.001047, 0.001088]

plt.figure(figsize=(8,6))
plt.plot(n, time, marker='o', linestyle='-', linewidth=2)

plt.title("Quick Sort: Input Size vs Time", fontsize=14)
plt.xlabel("Number of Elements (n)", fontsize=12)
plt.ylabel("Time Taken (seconds)", fontsize=12)

plt.grid(True, linestyle="--", alpha=0.6)
plt.show
```

Aaryan gupta 24293916141 (CSE B)

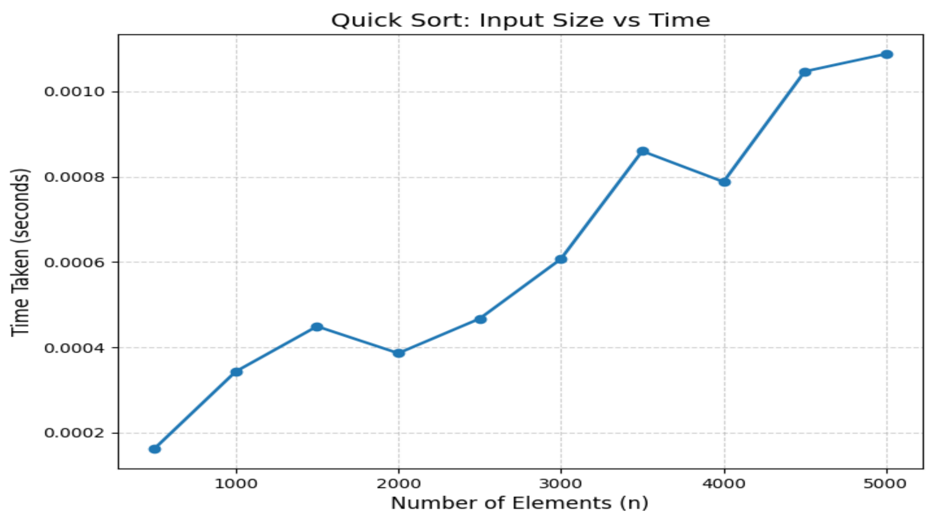


Fig4 :Graph Quick Sort

2(c) OBJECTIVE:

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

Algorithm: Insertion Sort

1. Start
2. Input an array A of size n
3. For i = 1 to n - 1 do
    a. key = A[i]
    b. j = i - 1
    c. While j >= 0 and A[j] > key do
        A[j + 1] = A[j]
        j = j - 1

Aaryan gupta 24293916141 (CSE B)

d. A[j + 1] = key

4. End For

5. Output the sorted array

6. Stop

PSEUDO CODE:

```
InsertionSort(arr)
for i from 1 to length(arr) - 1
key = arr[i]
j = i - 1
while j >= 0 and arr[j] > key
arr[j + 1] = arr[j]
j = j - 1
arr[j + 1] = key
```

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void insertionSort(int arr[], int n) {
    int i, j, temp;
    for (i = 1; i < n; i++) {
        temp = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > temp) {
            arr[j + 1] = arr[j];
            j--;
        }
```

```
        arr[j + 1] = temp;
    }
}

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

int main() {
    srand(time(0));
    int step = 500;
    for (int size = step; size <= step * 10; size += step) {
        int *original = malloc(size * sizeof(int));
        int *copy = malloc(size * sizeof(int));
        if (!original || !copy) {
            printf("Allocation failed!\n");
            return 1;
        }
        fillRandom(original, size);
        clock_t begin = clock();
        for (int repeat = 0; repeat < 1000; repeat++) {
            for (int k = 0; k < size; k++) {
                copy[k] = original[k];
            }
            insertionSort(copy, size);
```

```
        }
        clock_t finish = clock();

        double sec = (double)(finish - begin) / CLOCKS_PER_SEC / 1000.0;
        printf("Sorted %5d numbers in %.6f sec\n", size, sec);

        free(original);
        free(copy);
    }

    return 0;
}
```

OUTPUT:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc insertion.c -o insertion && "/Users/aaryangupta/ADA 3 RD SEMESTER/"insertion
Enter number of elements: 500
Time taken: 0.000586 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc insertion.c -o insertion && "/Users/aaryangupta/ADA 3 RD SEMESTER/"insertion
Enter number of elements: 1000
Time taken: 0.001705 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc insertion.c -o insertion && "/Users/aaryangupta/ADA 3 RD SEMESTER/"insertion
Enter number of elements: 1500
Time taken: 0.003667 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc insertion.c -o insertion && "/Users/aaryangupta/ADA 3 RD SEMESTER/"insertion
Enter number of elements: 2000
Time taken: 0.006612 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc insertion.c -o insertion && "/Users/aaryangupta/ADA 3 RD SEMESTER/"insertion
Enter number of elements: 2500
Time taken: 0.006008 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc insertion.c -o insertion && "/Users/aaryangupta/ADA 3 RD SEMESTER/"insertion
Enter number of elements: 3000
Time taken: 0.012023 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc insertion.c -o insertion && "/Users/aaryangupta/ADA 3 RD SEMESTER/"insertion
Enter number of elements: 4000
Time taken: 0.018106 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc insertion.c -o insertion && "/Users/aaryangupta/ADA 3 RD SEMESTER/"insertion
Enter number of elements: 3500
Time taken: 0.015633 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc insertion.c -o insertion && "/Users/aaryangupta/ADA 3 RD SEMESTER/"insertion
Enter number of elements: 4500
Time taken: 0.021737 seconds
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc insertion.c -o insertion && "/Users/aaryangupta/ADA 3 RD SEMESTER/"insertion
Enter number of elements: 5000
Time taken: 0.024781 seconds
```

Fig: 5 Output

Python code for Insertion sort graph:

```python
import matplotlib.pyplot as plt

n = [500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000]
time = [0.000586, 0.001705, 0.003667, 0.006612, 0.006008,
      0.012023, 0.015633, 0.018106, 0.021737, 0.024781]
```

Aaryan gupta 24293916141 (CSE B)

```
plt.plot(n, time, marker='o', linestyle='-', color='b')
plt.xlabel("Number of Elements (n)")
plt.ylabel("Time Taken (seconds)")
plt.title("Insertion Sort Time Complexity")
plt.grid(True)
plt.show()
```

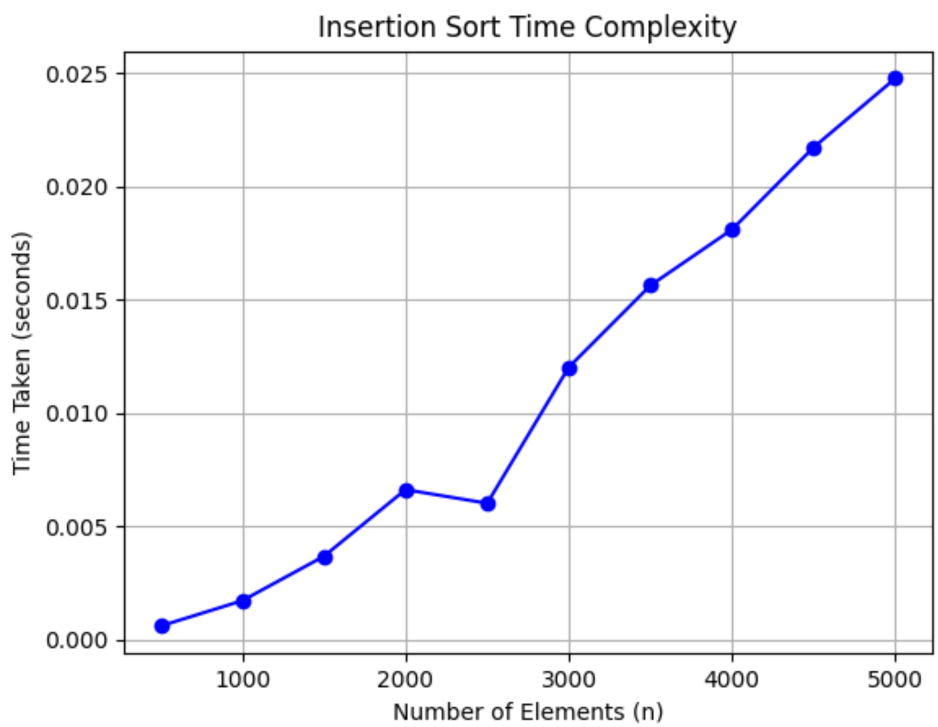


Fig 6: Graph Insertion Sort

2(d) OBJECTIVE:

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

Algorithm:

1. Start
2. Input number of elements n
3. Generate n random integers

Aaryan gupta 24293916141 (CSE B)

4. For each position i from 0 to n-2
   Find the minimum element from i to n-1
   Swap it with element at position i
5. Record the time taken for sorting
6. Repeat for different values of n
7. Print time taken
8. Stop

Pseudocode:

procedure selectionSort(A, n)

  for i = 0 to n-2 do

    minIndex = i

    for j = i+1 to n-1 do

      if A[j] < A[minIndex] then

        minIndex = j

    swap A[i] and A[minIndex]

end procedure

Code:

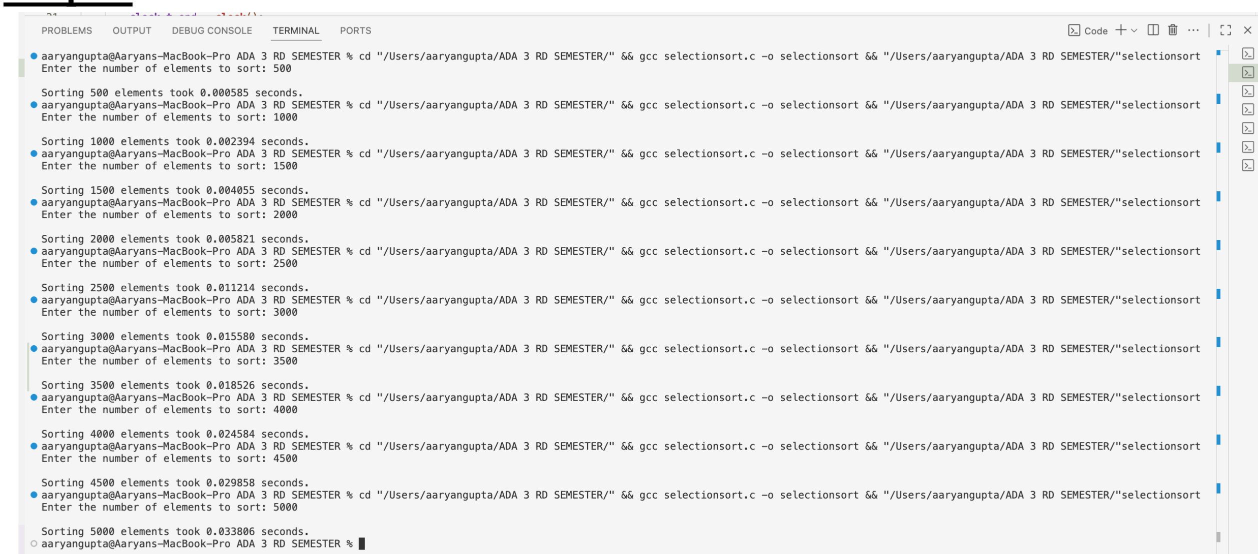
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void selectionSort(int arr[], int n);
void swap(int *xp, int *yp);

int main() {
    int n, i, j;
    FILE *fp = fopen("data.txt", "w");
    if (fp == NULL) return 1;

    srand(time(0));

    for (n = 1000; n <= 5000; n += 1000) {
        int *arr = (int *)malloc(n * sizeof(int));
        for (i = 0; i < n; i++) arr[i] = rand() % 1000 + 1;

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

Lab Practical 2: Sorting

Aaryan gupta 24293916141 (CSE B)

```
        double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
        fprintf(fp, "%d %f\n", n, time_taken);
        free(arr);
    }
    fclose(fp);
    return 0;
}
void selectionSort(int arr[], int n) {
    int i, j, min_idx;
    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;
        }
        if (min_idx != i) swap(&arr[min_idx], &arr[i]);
    }
}
void swap(int *xp, int *yp) {
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
```

Output:



```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc selectionsort.c -o selectionsort && "/Users/aaryangupta/ADA 3 RD SEMESTER/"selectionsort
Enter the number of elements to sort: 500

Sorting 500 elements took 0.000585 seconds.
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc selectionsort.c -o selectionsort && "/Users/aaryangupta/ADA 3 RD SEMESTER/"selectionsort
Enter the number of elements to sort: 1000

Sorting 1000 elements took 0.002394 seconds.
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc selectionsort.c -o selectionsort && "/Users/aaryangupta/ADA 3 RD SEMESTER/"selectionsort
Enter the number of elements to sort: 1500

Sorting 1500 elements took 0.004055 seconds.
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc selectionsort.c -o selectionsort && "/Users/aaryangupta/ADA 3 RD SEMESTER/"selectionsort
Enter the number of elements to sort: 2000

Sorting 2000 elements took 0.005821 seconds.
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc selectionsort.c -o selectionsort && "/Users/aaryangupta/ADA 3 RD SEMESTER/"selectionsort
Enter the number of elements to sort: 2500

Sorting 2500 elements took 0.011214 seconds.
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc selectionsort.c -o selectionsort && "/Users/aaryangupta/ADA 3 RD SEMESTER/"selectionsort
Enter the number of elements to sort: 3000

Sorting 3000 elements took 0.015580 seconds.
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc selectionsort.c -o selectionsort && "/Users/aaryangupta/ADA 3 RD SEMESTER/"selectionsort
Enter the number of elements to sort: 3500

Sorting 3500 elements took 0.018526 seconds.
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc selectionsort.c -o selectionsort && "/Users/aaryangupta/ADA 3 RD SEMESTER/"selectionsort
Enter the number of elements to sort: 4000

Sorting 4000 elements took 0.024584 seconds.
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc selectionsort.c -o selectionsort && "/Users/aaryangupta/ADA 3 RD SEMESTER/"selectionsort
Enter the number of elements to sort: 4500

Sorting 4500 elements took 0.029858 seconds.
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc selectionsort.c -o selectionsort && "/Users/aaryangupta/ADA 3 RD SEMESTER/"selectionsort
Enter the number of elements to sort: 5000

Sorting 5000 elements took 0.033806 seconds.
aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER %
```

Fig 7: Output

Aaryan gupta 24293916141 (CSE B)

Python code for Selection Sort graph:

```
import matplotlib.pyplot as plt

n = [500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000]
time = [0.000585, 0.002394, 0.004055, 0.005821, 0.011214,
        0.015580, 0.018526, 0.024584, 0.029858, 0.033806]

plt.plot(n, time, marker='o', linestyle='-', color='b')
plt.xlabel("Input Size (n)")
plt.ylabel("Time Taken (seconds)")
plt.title("Selection Sort: Time vs Input Size")
plt.grid(True)
plt.show()
```

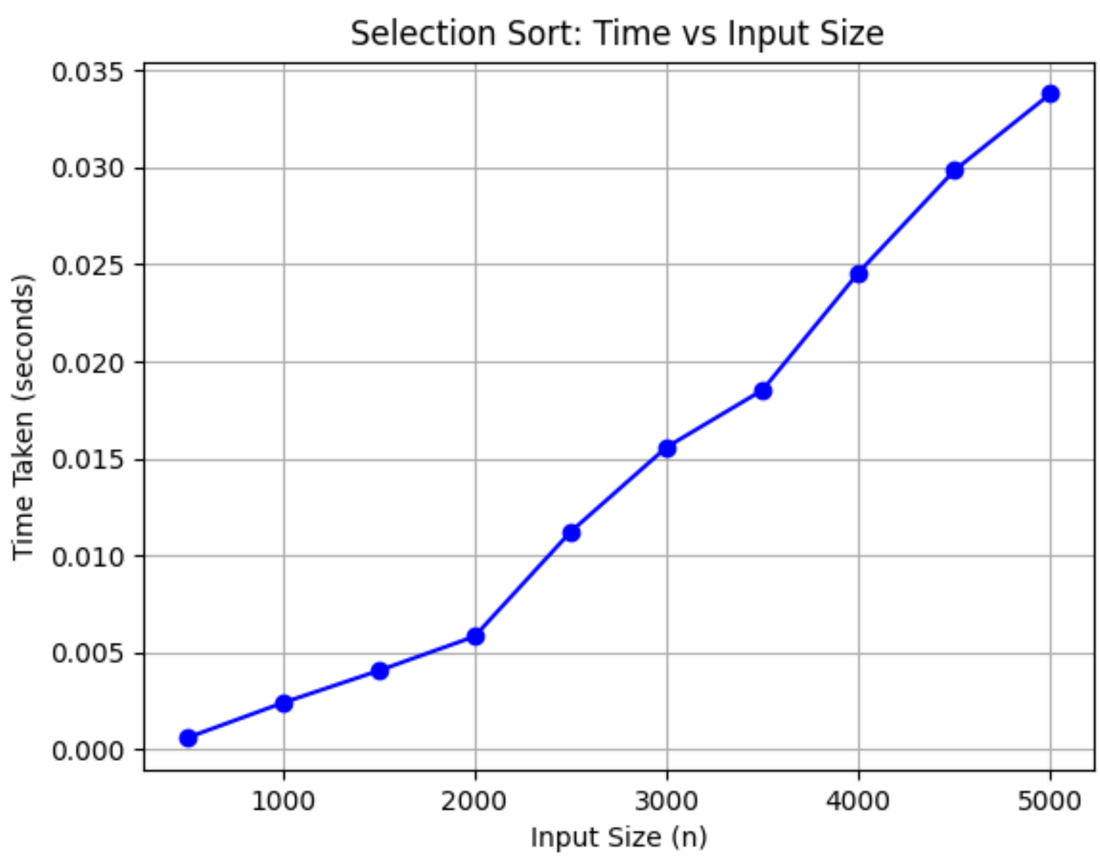


Fig 8: Graph Selection sort

Lab Practical 2: Sorting

Aaryan gupta 24293916141 (CSE B)

2(e) OBJECTIVE:

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.

Algorithm: Bubble Sort

1. Start
2. Input array A of size n
3. For i = 0 to n-1
   a. For j = 0 to n-i-2
      If A[j] > A[j+1] then
         Swap A[j] and A[j+1]
4. End For
5. End

PSEUDO CODE:

BubbleSort(arr)

for i from 0 to length(arr) - 1

for j from 0 to length(arr) - i - 2

if arr[j] > arr[j + 1]

swap arr[j] and arr[j + 1]

Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void bubbleSort(int arr[], int n) {

    for (int pass = 0; pass < n - 1; pass++) {

        int flag = 0;

        for (int j = 0; j < n - pass - 1; j++) {
```

Aaryan gupta 24293916141 (CSE B)

```
            if (arr[j] > arr[j + 1]) {
                int tmp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = tmp;
                flag = 1;
            }
        }
        if (flag == 0) break;
    }
}
void fillRandom(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 100000;
    }
}
int main() {
    int base = 500;
    srand(time(NULL));
    for (int size = base; size <= 10 * base; size += base) {
        int *original = malloc(size * sizeof(int));
        int *temp = malloc(size * sizeof(int));
        if (!original || !temp) {
            printf("Memory allocation error\n");
            return 1;
```

```
        }
        fillRandom(original, size);
        clock_t start = clock();
        for (int run = 0; run < 1000; run++) {
            for (int k = 0; k < size; k++) {
                temp[k] = original[k];
            }
            bubbleSort(temp, size);
        }
        clock_t finish = clock();
        double avgTime = (double)(finish - start) / CLOCKS_PER_SEC / 1000.0;
        printf("Sorting %d elements took %.6f seconds (avg)\n", size, avgTime);

        free(original);
        free(temp);
    }
    return 0;
}
```

<u>Output:</u>

```
● aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % cd "/Users/aaryangupta/ADA 3 RD SEMESTER/" && gcc bubblesort.c -o bubblesort && "/Users/aaryangupta/ADA 3 RD SE
  Sorting 500 elements took 0.000250 seconds (avg)
  Sorting 1000 elements took 0.000888 seconds (avg)
  Sorting 1500 elements took 0.002005 seconds (avg)
  Sorting 2000 elements took 0.003505 seconds (avg)
  Sorting 2500 elements took 0.005355 seconds (avg)
  Sorting 3000 elements took 0.008162 seconds (avg)
  Sorting 3500 elements took 0.010963 seconds (avg)
  Sorting 4000 elements took 0.014569 seconds (avg)
  Sorting 4500 elements took 0.018581 seconds (avg)
  Sorting 5000 elements took 0.022890 seconds (avg)
○ aaryangupta@Aaryans-MacBook-Pro ADA 3 RD SEMESTER % 
```

Fig 9: Output

Aaryan gupta 24293916141 (CSE B)

Python code for graph:

```
import matplotlib.pyplot as plt

n = [500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000]
time = [0.000250, 0.000888, 0.002005, 0.003505, 0.005355, 0.008162, 0.010963, 0.014569, 0.018581, 0.022890]

plt.plot(n, time, marker='o')
plt.xlabel("Input Size (n)")
plt.ylabel("Time (seconds)")
plt.title("Bubble Sort: Time vs Input Size")
plt.grid(True)
plt.show()
```

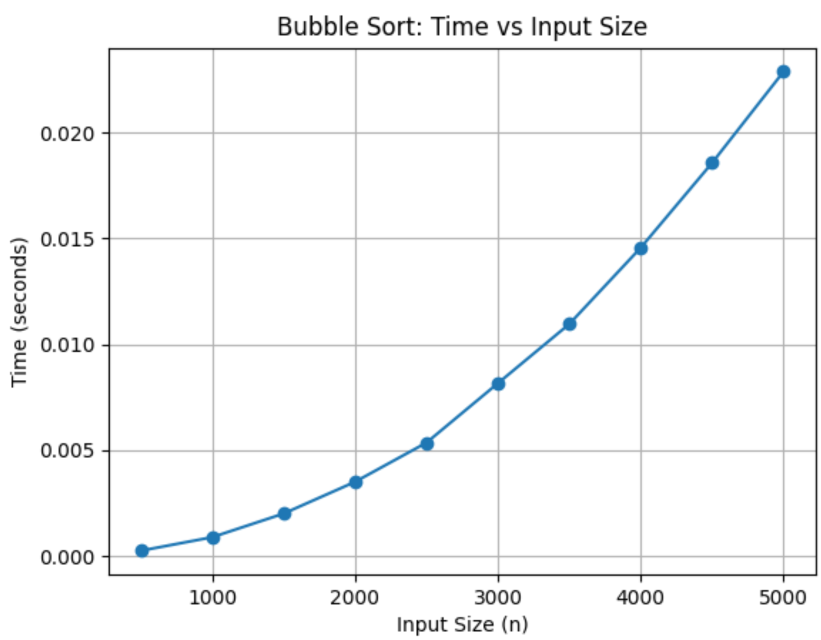


Fig 10: Graph Bubble Sort

Conclusion

1) **Limited Input Sizes (n is small):**
   For smaller arrays, execution time differences between algorithms are minimal, resulting in potentially inconsistent graph patterns.

Aaryan gupta 24293916141 (CSE B)

2) **System Interference:**
   Background processes and operating system scheduling can impact execution time measurements and create irregularities in performance graphs.
3) **Input Data Characteristics:**
   Sorting pre-ordered arrays versus randomized versus reverse-sorted data can produce significantly different performance results.
4) **Hardware Dependencies:**
   Algorithm performance varies across different hardware configurations, processor speeds, and memory architectures.
5) **Measurement Precision:**
   Timing mechanisms and system clock granularity can introduce inaccuracies, especially for fast-executing small datasets.
6) **Algorithmic Overhead:**
   Implementation details and constant factors can mask theoretical complexity differences in smaller input sizes.
7) **Statistical Variability:**
   Individual test runs may produce varying results due to system state changes, requiring multiple iterations for reliable conclusions.

Aaryan Gupta

24293916141(CSE B)