Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

Academic Year: 2023 - 24

COURSE CODE: DJS22ITL302 **CLASS: S. Y. B. Tech. Sem III (I1-1)**

COURSE NAME: Data Structures Lab **SAP ID : 60003220045**

Name: Anish Sharma **DATE:** 04-12-2023

EXPERIMENT NO. 9

CO/LO: Solve the problem using sorting techniques.

Objective: Write a program to implement different quadratic sorting algorithms with various parameters. Analyze the performance of all the algorithms.

Code

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

// Function prototypes
void quick_sort(int arr[], int low, int high, int *swaps, int *comparisons); void
merge_sort(int arr[], int low, int high, int *swaps, int *comparisons); void
selection_sort(int arr[], int n, int *swaps, int *comparisons); void radix_sort(int
arr[], int n, int *moves);


// Helper functions void
print_array(int arr[], int n); void
swap(int *a, int *b);

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

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

int arr[n];
```

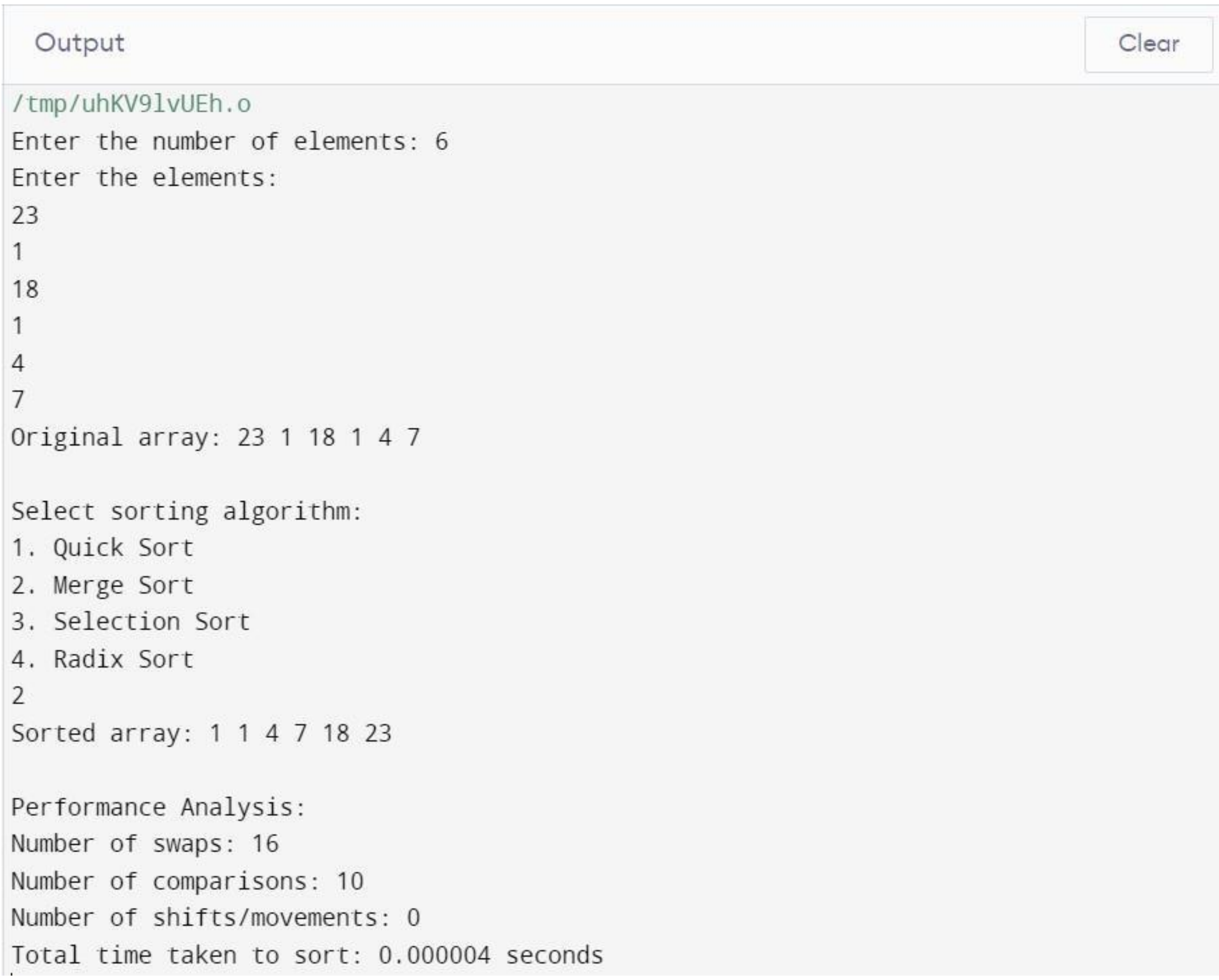
```
    printf("Enter the elements:\n"); for
    (int i = 0; i < n; i++) { scanf("%d",
    &arr[i]);

    }


    printf("\nOriginal array: "); print_array(arr,
    n);

int choice; printf("\nSelect sorting algorithm:\n1. Quick Sort\n2. Merge Sort\n3.
 Selection Sort\n4. Radix Sort\n"); scanf("%d", &choice);

int swaps = 0, comparisons = 0, moves = 0;
    clock_t start_time, end_time;

start_time = clock(); switch (choice) { case 1: quick_sort(arr, 0, n - 1,
    &swaps, &comparisons); break; case 2: merge_sort(arr, 0, n - 1, &swaps,
    &comparisons); break; case 3: selection_sort(arr, n, &swaps,
    &comparisons); break; case 4: radix_sort(arr, n, &moves); break;
    default: printf("Invalid choice\n"); return 1;
    }
    end_time  = clock();


    printf("\nSorted array: "); print_array(arr,
    n);

printf("\nPerformance Analysis:\n");
    printf("Number of swaps: %d\n", swaps);
    printf("Number of comparisons: %d\n", comparisons); printf("Number
    of shifts/movements: %d\n", moves);
      printf("Total time taken        to sort: %lf seconds\n",        ((double)(end_time -
 start_time)) / CLOCKS_PER_SEC);


    return 0;
 }
 void quick_sort(int arr[], int low, int high, int *swaps, int *comparisons) {
     if (low < high) { int pivot = arr[high]; int i = low - 1;
for (int j = low; j < high; j++) {
         (*comparisons)++;
             if (arr[j] <= pivot) { i++;
                 // Swap arr[i] and arr[j]
                 int temp = arr[i]; arr[i] =
                 arr[j]; arr[j] = temp;
                 (*swaps)++;
             }
         }


         // Swap arr[i+1] and arr[high] (pivot)
         int temp = arr[i + 1]; arr[i + 1] =
         arr[high]; arr[high] = temp;
         (*swaps)++;

         int partition_index = i + 1;


         // Recursively sort the sub-arrays
         quick_sort(arr, low, partition_index - 1, swaps, comparisons);
         quick_sort(arr, partition_index + 1, high, swaps, comparisons);
     }
 }
 void
 mer
 ge(i
 nt
 arr[]
 , int
 low,
 int
 mid,
 int
 high
 , int
 *sw
 aps,
 int
 *co
 mpa
 riso
 ns) {
 int
 n1 =
 mid
 -
 low
 + 1;
     int n2 = high - mid;

     int left[n1], right[n2];

     for (int i = 0; i < n1; i++) left[i]
         = arr[low + i];
     for (int j = 0; j < n2; j++) right[j]
         = arr[mid + 1 + j];

int i = 0, j = 0, k = low; while
     (i < n1 && j < n2) {
     (*comparisons)++;
         if (left[i] <= right[j]) {
             arr[k] = left[i];
             i++;
```

```
          }   else  {  arr[k]  =
              right[j]; j++;
          }
          k++;
          (*swaps)++;
      }

      while (i < n1) {
          arr[k] = left[i];
          i++; k++;
          (*swaps)++;
      }

      while (j < n2) { arr[k]
          = right[j]; j++;
          k++;
          (*swaps)++;
      }
  }

  void merge_sort(int arr[],  int  low,  int  high,  int  *swaps,  int  *comparisons)  {
      if (low < high) { int mid = low + (high - low) / 2;
          merge_sort(arr, low, mid, swaps, comparisons); merge_sort(arr, mid
          + 1, high, swaps, comparisons); merge(arr, low, mid, high, swaps,
          comparisons);
      }
  }

  void selection_sort(int arr[], int n, int *swaps, int *comparisons) {
      for (int i = 0; i < n - 1; i++) { int min_index = i;
```

```
            for (int j = i + 1; j < n; j++) {
                (*comparisons)++; if
                (arr[j] < arr[min_index])
                min_index = j;
            }


            int temp = arr[min_index];
            arr[min_index] = arr[i];
            arr[i] = temp;
            (*swaps)++;
        }
    }


    void counting_sort(int arr[], int n, int exp, int *moves) { int
        output[n]; int count[10] = {0};

        for (int i = 0; i < n; i++)
            count[(arr[i] / exp) % 10]++;

  for (int i = 1; i < 10; i++)
            count[i] += count[i - 1];

  for (int i = n - 1; i >= 0; i--) {  output[count[(arr[i] /
        exp) % 10] - 1] = arr[i]; count[(arr[i] / exp) % 10]--
        ;
            (*moves)++;
        }


        for (int i = 0; i < n; i++)
            arr[i] = output[i];
    }


    void radix_sort(int arr[], int n, int *moves) {
```

```
    int max = 0; for (int i =
    0; i < n; i++) {
        if (arr[i] > max) max
            = arr[i];
    }

    for (int exp = 1; max / exp > 0; exp *= 10)
        counting_sort(arr, n, exp, moves);
}

void generate_data(int arr[], int n, int order) {
    switch (order) {
        case 1: // In-order
            for (int i = 0; i < n; i++) {
                arr[i] = i + 1;
            }
            break;
        case 2: //  Reverse-order for
            (int i = 0; i < n; i++) {
            arr[i] = n - i;
            }
            break;
        case 3: // Random
            for (int i = 0; i < n; i++) {
                arr[i] = rand() % n + 1;
            }
            break;
        case 4:  //  Almost  order for
            (int  i  =  0;  i  <  n;  i++)  {
            arr[i] = i + 1;
            }
            // Shuffle a few elements to make it almost ordered for
            (int i = 0; i < n / 10; i++) {
                int pos1 = rand() % n; int
                pos2 = rand() % n; int
                temp = arr[pos1];
                arr[pos1] = arr[pos2];
                arr[pos2] = temp;
            }
            break;
        default: printf("Invalid data
            order\n"); exit(1);
    }
}

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

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

OUTPUT :



```
/tmp/uhKV9lvUEh.o
Enter the number of elements: 6
Enter the elements:
23
1
18
1
4
7
Original array: 23 1 18 1 4 7

Select sorting algorithm:
1. Quick Sort
2. Merge Sort
3. Selection Sort
4. Radix Sort
2
Sorted array: 1 1 4 7 18 23

Performance Analysis:
Number of swaps: 16
Number of comparisons: 10
Number of shifts/movements: 0
Total time taken to sort: 0.000004 seconds
```

Conclusion:

This C code implements a program for sorting an array using four different sorting algorithms: Quick Sort, Merge Sort, Selection Sort, and Radix Sort. The program allows the user to input the number of elements in the array and the elements themselves. It then provides the option to choose one of the sorting algorithms for sorting the array. Thus implemented sort successfully.

Website References:

- Geeksforgeeks

-Javatpoint

-programiz