|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Analysis of Algorithms** | **Semester:** | **IV** |
| **Date of Performance:** | **21 / 01 / 2024** | **Batch No:** | **EXCP B1** |
| **Faculty Name:** | **Prof. Payal Varangoankar** | **Roll No:** | **16014022096** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** |  |

**Experiment No: 1**

**Title: Implementation of Insertion sort.**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| To analyze performance of sorting methods |

|  |
| --- |
| **COs to be achieved:** |
| **CO1: Analyze the asymptotic running time and space complexity of algorithms.** |

## 

## **Apparatus / Software tools used:**

|  |
| --- |
| **Theory:** |
| Given a function to compute on n inputs the divide-and-conquer strategy suggests splitting the inputs into k distinct subsets, 1< k ≤n, yielding k subproblems. These sub-problems must be solved and then a method must be found to combine sub-solutions into a solution of the whole. The divide-and-conquer strategy can be reapplied if the sub-problems are still relatively large. Often the sub-problems resulting from a divide-and-conquer design are the same type as the original problem. For those cases, a recursive algorithm naturally expresses the reapplication of the divide-and-conquer principle. Now smaller and smaller subproblems of the same kind are generated until eventually subproblems that are small enough to be solved without splitting are produced. |

|  |
| --- |
| **Code:**  **INSERTION SORT:**  #include <stdio.h>  #include <stdlib.h>  **int** count\_i = 0;  **void** insertionSort(**int** A**[]**, **int** n) {  **int** i, j, key;        for (j = 1; j < n; j++) {          count\_i++;          key = A[j];          i = j - 1;          while (i >= 0 && A[i] > key) {              A[i + 1] = A[i];              i = i - 1;            }          A[i + 1] = key;      }    }  **int** main() {  **int**  n;      printf("Enter value of n: ");      scanf("%d", &n);  **int** arr[n];      printf("Original Array: ");      for (**int** i = 0; i < n; i++) {          arr[i] = rand() % 10;          printf("%d ", arr[i]);      }      printf("\n");      insertionSort(arr, n);      printf("Sorted Array: ");      for (**int** i = 0; i < n; i++) {          printf("%d ", arr[i]);      }      printf("\n");        printf("Count : %d \n", count\_i);      return 0;  }    **SELECTION SORT:**  #include <stdio.h>  #include <stdlib.h>  #include <time.h>  **int** count\_i = 0;  **void** selectionSort(**int** arr**[]**, **int** n) {      for (**int** i = 0; i < n - 1; i++) {  **int** minIndex = i;          for (**int** j = i + 1; j < n; j++) {              if (arr[j] < arr[minIndex]) {                  minIndex = j;              }              count\_i++;          }  **int** temp = arr[i];          arr[i] = arr[minIndex];          arr[minIndex] = temp;        }  }  **int** main() {    **int**  n;      printf("Enter value of n: ");      scanf("%d", &n);  **int** arr[n];      printf("Original Array: ");      for (**int** i = 0; i < n; i++) {          arr[i] = rand() % 10;          printf("%d ", arr[i]);      }      printf("\n");      selectionSort(arr, n);      printf("Sorted Array: ");      for (**int** i = 0; i < n; i++) {          printf("%d ", arr[i]);      }      printf("\n");        printf("Count : %d \n", count\_i);      return 0;  } |

|  |
| --- |
| **Stepwise-Procedure / Algorithm:** |
| **Algorithm Insertion Sort**  INSERTION\_SORT (A,n)  //The algorithm takes as parameters an array A[1.. n] and the length n of the array.  //The array A is sorted in place: the numbers are rearranged within the array  // A[1..n] of eletype, n: integer  FOR j ← 2 TO length[A]  DO key ← A[j]  {Put A[j] into the sorted sequence A[1 . . j − 1]}  i ← j − 1  WHILE i > 0 and A[i] > key  DO A[i +1] ← A[i]  i ← i − 1  A[i + 1] ← key |

|  |
| --- |
| **Observation Table:** |
| **Graphs for varying input sizes of Insertion Sort**      **Graphs for varying input sizes of Selection Sort:** |

|  |
| --- |
| **Output:** |
| **INSERTION SORT:**        **SELECTION SORT:** |

|  |
| --- |
| **Calculation:** |
| **The Time and Space complexity of Insertion sort:** |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| Solve the problem theoretically which was implemented during practical |

|  |
| --- |
| **Conclusion:** |
| We have successfully implemented Insertion sort and Selection sort and derived following analysis  **Insertion Sort:**   * Best Case Time Complexity: O(n) * Average Case Time Complexity: O(n^2) * Worst Case Time Complexity: O(n^2) * Space Complexity: O(1)   **Selection Sort**   * Best Case Time Complexity: O(n^2) * Average Case Time Complexity: O(n^2) * Worst Case Time Complexity: O(n^2) * Space Complexity: O(1) |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |