|  |  |
| --- | --- |
| **NAME:** | MIHIKA CHACHAD |
| **UID:** | 2021300017 |
| **SUBJECT** | Design and Analysis of Algorithms |
| **EXPERIMENT NO:** | 1B |
| **DATE OF PERFORMANCE** | 02.02.2023 |
| **DATE OF SUBMISSION** | 09.02.2023 |
| **AIM:** | Experiment on finding the running time of an algorithm. |
| **THEORY** | The understanding of the running time of algorithms is explored by implementing two basic sorting algorithms  namely Insertion and Selection sorts. These algorithms work as follows.  Insertion sort– It works similarly to the sorting of playing cards in hands. It is assumed that the first card is already sorted in the card game, and then we select an unsorted card. If the selected unsorted card is greater than the first card, it will be placed at the right side; otherwise, it will be placed at the left side. Similarly, all unsorted cards are  taken and put in their exact place.  Selection sort– It first finds the smallest value among the unsorted elements of the array is selected in every pass  and inserted into its appropriate position into the array. In this algorithm, the array is divided into two parts, the first is  the sorted part, and another one is the unsorted part. Initially, the sorted part of the array is empty, and the unsorted part is  the given array. The sorted part is on the left, while the unsorted part is on the right. In selection sort, the  first smallest element is selected from the unsorted array and placed at the first position. After that second smallest  element is selected and placed in the second position. The process continues until the array is entirely sorted. |
| **ALGORITHM** | 1. Selection sort: 2. Initialize minimum value to the first index. 3. Traverse the array. 4. While traversing if any element smaller than minimum value is found, then swap both the values. 5. Then, increment the minimum index to point to the next element. 6. Repeat until the array is sorted. 7. Insertion sort: 8. Iterate from first index to the last element of the array. 9. Compare the current element to the value at the previous index. 10. If the current element is smaller than its predecessor, compare it to the elements before. 11. Increment the greater elements one position up to make space for the swapped element. |
| **PROGRAM:** | #include <stdio.h>  #include <math.h>  #include <stdlib.h>  #include <time.h>  void insertionsort(int arr[], int n) {      for (int i = 1; i < n; i++) {          int key = arr[i];          int j = i - 1;          while (key < arr[j] && j >= 0) {              arr[j + 1] = arr[j];              --j;          }          arr[j + 1] = key;      }  }  void selectionsort(int arr[], int n){      int min, temp;      for(int i=0; i<n; i++){          min = i;          for(int j=i+1; j<n; j++){              if(arr[j] < arr[min]){                  min = j;              }          }          temp = arr[min];          arr[min] = arr[i];          arr[i] = temp;      }  }  int main(){      FILE \*fptr, \*sPtr;      int index=99;      int a[100000];      clock\_t t;      fptr = fopen("Random.txt", "r");      sPtr = fopen("Result.txt", "w");      for(int i=0; i<=999; i++){          for(int j=0; j<=index; j++){              fscanf(fptr, "%d", &a[j]);          }          t = clock();          insertionsort(a, index+1);          t = clock() - t;          double time\_taken = ((double)t)/CLOCKS\_PER\_SEC;          fprintf(sPtr, "%lf\n", time\_taken);          printf("%d\t%lf\n", (i+1), time\_taken);          index = index + 100;          fseek(fptr, 0, SEEK\_SET);      }      fclose(sPtr);      fclose(fptr);      return 0;  }  For selection sort, the function called for insertionsort() was replaced by selectionsort(). |
| RESULT:       * Insertion sort is a simple sorting algorithm that builds the final sorted list by transferring one element at a time. Selection sort, in contrast, is a simple sorting algorithm that repeatedly searches for remaining items to find the smallest element and moves it to the correct location. * Insertion sort is more efficient than selection sort. * The insertion sort is fast, efficient, and stable while the selection sort only works efficiently when a small set of elements is involved or the list is partially previously sorted. | |
| **CONCLUSION:** | In this experiment, we learned about insertion and selection sort. We performed sorting 1000 times and compared the time complexities for both algorithms using charts. |