|  |  |
| --- | --- |
| **Experiment No.** | **1-B** |
| **Aim** | **Experiment on finding the running time of an algorithm.** |
| **Name** | **Atharva Gite** |
| **UID No.** | **2021300038** |
| **Class & Division** | **COMPS-A(C)** |

**Theory:**

**1. Insertion Sort:**

Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time by comparisons. Insertion sort iterates, consuming one input element each repetition and grows a sorted output list. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list and inserts it there. It repeats until no input elements remain.

Array prior to the insertion of x Array after the insertion of x

Worst complexity: n^2

Average complexity: n^2

Best complexity: n

Space complexity: 1

**2. Selection Sort:**

selection sort is an in-place comparison sorting algorithm. It has an O(n2) time complexity, which makes it inefficient on large lists, and generally performs worse than the similar insertion sort. The algorithm divides the input list into two parts: a sorted sublist of items which is built up from left to right at the front (left) of the list and a sublist of the remaining unsorted items that occupy the rest of the list. Initially, the sorted sublist is empty and the unsorted sublist is the entire input list. The algorithm proceeds by finding the smallest (or largest, depending on sorting order) element in the unsorted sublist, exchanging (swapping) it with the leftmost unsorted element (putting it in sorted order), and moving the sublist boundaries one element to the right.

Worst complexity: n^2

Average complexity: n^2

Best complexity: n^2

Space complexity: 1

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define N 100000

#define BLOCK\_SIZE 100

void insertion\_sort(int arr[], int n) {

    int i, j, key;

    for (i = 1; i < n; i++) {

        key = arr[i];

        j = i - 1;

        while (j >= 0 && arr[j] > key) {

            arr[j + 1] = arr[j];

            j = j - 1;

        }

        arr[j + 1] = key;

    }

}

void selection\_sort(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;

            }

        }

        int temp = arr[min\_idx];

        arr[min\_idx] = arr[i];

        arr[i] = temp;

    }

}

int main() {

    FILE \*fp,\*ptri,\*ptrs;

    time\_t t;

    int i,j,n,k;

    int array[N];

    //open the file

    fp = fopen("random\_numbers.txt", "w");

    srand((unsigned) time(&t));

    // print 100,000 random numbers

    for( k = 0 ; k < N ; k++ )

    {

    int num=rand()%100000;

    fprintf(fp, "%d\n", num);

    }

    // close the file

    fclose(fp);

    //open the file

    fp = fopen("random\_numbers.txt", "r");

    k=0;

    // Read each number from the file and store it in the array

    while (fscanf(fp, "%d", &n) == 1)

    {

        array[k++] = n;

    }

    // Close the file

    fclose(fp);

    ptri= fopen("insertion\_sort\_time.txt", "w");

    ptrs= fopen("selection\_sort\_time.txt", "w");

    //clock start

    clock\_t start, end;

    for (i=1;i<1001;i++)

    {

        double insertion\_sort\_time = 0;

        double selection\_sort\_time = 0;

        // 1\*100= 100 blocks, 2\*100=200 blocks and so on

        int size;

        size=i\*BLOCK\_SIZE;

        // block arr to find out combien de time for sorting 0-99,0-199 and so on

        int block\_arr[size];

        for(j = 0; j<size ; j++)

        {

        block\_arr[j] = array[j];

        }

            start = clock();

            insertion\_sort(block\_arr,size);

            end = clock();

            insertion\_sort\_time += ((double) (end - start)) / CLOCKS\_PER\_SEC;

            start = clock();

            selection\_sort(block\_arr,size);

            end = clock();

            selection\_sort\_time += ((double) (end - start)) / CLOCKS\_PER\_SEC;

            fprintf(ptri, "%lf\n",insertion\_sort\_time);

            fprintf(ptrs, "%lf\n",selection\_sort\_time);

            }

    return 0;

}

**GRAPHS:**

**Result:**  For smaller inputs, both algorithms perform similarly. But for larger inputs insertion sort performs better than selection sort.