| **NAME:** | Bhavesh Prashant Chaudhari |
| --- | --- |
| **UID:** | 2021300018 |
| **SUBJECT** | DAA |
| **EXPERIMENT NO :** | 1B |
| **DATE OF PERFORMANCE** | 2/2/23 |
| **DATE OF SUBMISSION** | 7/2/23 |
| **AIM:** | Experiment on finding the running time of an algorithm. |
| **PROBLEM STATEMENT 1:** |  |
| **THEORY** | Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.  Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list. The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted portion. This process is repeated for the remaining unsorted portion of the list until the entire list is sorted. |
| **ALGORITHM** | Insertion Sort-  1.)Iterate from arr[1] to arr[N] over the array.  2.)Compare the current element (key) to its predecessor.  3.)If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.  INSERTION SORT(A,n):  for i = 1 to n  key = A[i]  j = i-1  while j > 0 and A[j] > key  A[j+1] = A[j]  j = j -1  A[j+1] = key  Selection Sort-  1.)Initialize minimum value(min\_idx) to location 0.  2.)Traverse the array to find the minimum element in the array.  3.)While traversing if any element smaller than min\_idx is found then swap both the values.  4.)Then, increment min\_idx to point to the next element.  5.)Repeat until the array is sorted.  SELECTION SORT(A,n):  for i = 0 to n-1  min\_index = i  for j = i to n  if A[j] < A[min\_index] then min\_index = j  swap(A[i],A[min\_index]) |
| **PROGRAM:** | #include<iostream>  #include<time.h>  #include <cstdlib>  #include<fstream>  #include<iomanip>  #include<numeric>  #include<chrono>  using namespace std;  void genInput(){  ofstream fp;  srand(time(0));  fp.open("input.txt");  for(int i=0;i<100000;i++)  fp << rand() << endl;  fp.close();  }  void readInput(int \*arr,int size){  int i = 0;  ifstream fp;  fp.open("input.txt");  while(i<=size){  fp >> arr[i];  i++;  }  fp.close();  }  void insertionSort(int \*arr,int size){  for(int i=1;i<=size;i++){  int key = arr[i];  int j = i-1;  while(j>=0 && arr[j]>key){  arr[j+1] = arr[j];  j--;  }  arr[j+1] = key;  }  }  void SelectionSort(int \*arr,int size){  for(int i=0;i<size;i++){  int min\_idx = i;  for(int j=min\_idx+1;j<size;j++)  if(arr[min\_idx] > arr[j])  min\_idx = j;  if(min\_idx != i){  int temp = arr[i];  arr[i] = arr[min\_idx];  arr[min\_idx] = temp;  }  }  }  void print(int \*arr){  ofstream fp;  fp.open("sorted.txt");  for(int i=0;i<100000;i++)  fp << arr[i] << endl;  fp.close();  }  int main(){  int \*arr = new int[100000];  genInput();  ofstream fp;  fp.open("selection\_time.txt");  for(int i=1;i<=1000;i++){  auto start = std::chrono::high\_resolution\_clock::now();  int size = (i\*100) - 1;  readInput(arr,size);  SelectionSort(arr,size);  std::chrono::duration<double> time = std::chrono::high\_resolution\_clock::now() - start;  fp << time.count() << endl;  cout << time.count() << endl;  }  return 0;  } |
| **RESULT :**  Insertion Sort-  Graph obtained for insertion sort with 100000 numbers as input:    Above graph depicts the order of growth for Insertion Sort.It can be seen that as the size of input increase,time required to sort increase in quadratic manner.Hence time complexity is O(N^2).    Above graph was obtained when insertion sort was applied on an array which was already sorted.From graph we understand that the as size of input increases, time required for sorting increase very slowly.This is because the array is already sorted which gives time complexity of O(N).  Selection Sort-  Graph obtained for selection sort with 100000 numbers as input:    Above graph shows the growth of time required to sort the array as size of array increases.As size of input increases time required for sorting increases rapidly.    Above graph was obtained when selection sort was applied on a sorted array.As it can be seen the time complexity for selection sort even when array is sorted is O(N^2). | |
| **CONCLUSION:** | From the above graph we can conclude that Selection sort is inefficent when compared with insertion sort as time complexity of selection sort is O(N^2) regardless of the worst case or best case while time complexity of insertion sort is O(N) in best case and O(N^2) in worst case. |