|  |  |
| --- | --- |
| **NAME:** | Akshaya Karande |
| **UID:** | 2021300056 |
| **SUBJECT** | Design and Analysis of Algorithms |
| **EXPERIMENT NO :** | 1 (B) |
| **AIM:** | To implement Selection and Insertion Sort |
| **ALGORITHM:** |  |
| **PROGRAM:** | #include<bits/stdc++.h>  #include<chrono>  #include<fstream>  using namespace std;  int main()  {  int n=100000;  int arr[n];  ofstream outdata;  outdata.open("output.csv");  for(int i=0;i<n;i++)  {  arr[i]=rand()%n;  // cout<<arr[i]<<"\n";  }  int copyarr[n];  copy(arr,arr+n,copyarr);    for(int i=100;i<=n;i+=100)  {  //Insertion Sort  auto start=chrono::high\_resolution\_clock::now();  for(int j=1;j<i;j++)  {  int k=j;  while(arr[k-1]>arr[k] && k>0)  {  int temp;  temp=arr[k-1];  arr[k-1]=arr[k];  arr[k]=temp;  k--;  }  }  auto end=chrono::high\_resolution\_clock::now();  outdata<<"\n"<<i/100<<","<<chrono::duration\_cast<chrono::nanoseconds>(end-start).count()<<",";  // for(int a=0;a<i;a++)  // {  // cout<<arr[a]<<" ";  // }  //Selection Sort  start=chrono::high\_resolution\_clock::now();  for (int j=0;j<i;j++)  {  int min=j;  for(int k=j+1;k<i;k++)  {  if(copyarr[min]>copyarr[k])  {  min=k;  }  }  int temp=copyarr[min];  copyarr[min]=copyarr[j];  copyarr[j]=temp;  }  end=chrono::high\_resolution\_clock::now();  outdata<<chrono::duration\_cast<chrono::nanoseconds>(end-start).count();  // for(int a=0;a<i;a++)  // {  // cout<<copyarr[a]<<" ";  // }  }  return 0;  } |
| Observations: | |
| **Conclusion**  **In conclusion, both insertion sort and selection sort are simple and efficient sorting algorithms that are commonly used in computer science. Although they have different time complexities, they both work by comparing and swapping elements in a list until the list is sorted in ascending or descending order.**  **Insertion sort is best suited for small lists or for lists that are almost sorted. It has a time complexity of O(n^2) in the worst case, but its best-case time complexity is O(n) when the list is already sorted. Insertion sort also requires minimal additional memory space, making it a good choice for systems with limited memory resources.**  **Selection sort is a good choice for small or medium-sized lists. It has a time complexity of O(n^2) in all cases, making it less efficient than some other sorting algorithms, but it is simple to implement and requires minimal additional memory space. Selection sort also works well on lists that are partially sorted or have repeated elements.**  **Overall, the choice between insertion sort and selection sort depends on the size and characteristics of the list being sorted. Both algorithms have their strengths and weaknesses, but they are effective tools for sorting data in a computer program.** | |