|  |  |
| --- | --- |
| **NAME:** | Harshey Kaur Soi |
| **UID:** | 2021300057 |
| **SUBJECT** | Design and Analysis of Algorithm |
| **EXPERIMENT NO :** | 2 |
| **AIM:** | Experiment on finding the running time of an algorithm. |
| **Algorithm** | 1. **Insertion sort-**    1. procedure insertionSort(A: list of sortable items)    2. n = length(A)    3. for i = 1 to n - 1 do    4. j = i    5. while j > 0 and A[j-1] > A[j] do    6. swap(A[j], A[j-1])    7. j = j - 1    8. end while    9. end for    10. end procedure 2. **Selection sort-**    1. Repeat Steps b and c for i = 0 to n-1    2. CALL SMALLEST(arr, i, n, pos)    3. [INITIALIZE] SET key = i    4. Repeat for j = i+1 to n    5. if (arr[key] > arr[j])    6. SET key=j    7. [END OF if]    8. [END OF LOOP]    9. SWAP arr[i] with arr[pos]    10. [END OF LOOP]    11. EXIT |
| **PROGRAM:** | *#include* <bits/stdc++.h>  *#include* <fstream>  using namespace std;  void insertionsort(vector<int> arr, int num)  {      int key, j;  *for* (int i = 1; i < num; i++)      {          key = arr[i];          j = i - 1;  *while* (j >= 0 && arr[j] > key)          {              arr[j + 1] = arr[j];              j--;          }          arr[j + 1] = key;      }  }  void selectionsort(vector<int> arr, int num)  {      int key;  *for* (int i = 0; i < num - 1; i++)      {          key = i;  *for* (int j = i + 1; j < num; j++)          {  *if* (arr[j] < arr[key])              {                  key = j;              }          }          int temp;          temp = arr[key];          arr[key] = arr[i];          arr[i] = temp;      }  }  int main()  {      vector<int> arr;      clock\_t t1, t2, t3, t4;      string filename("values.txt");      ifstream fin(filename);  *if* (!fin.is\_open())      {          cerr << "Could not open the file - '"               << filename << "'" << endl;  *return* EXIT\_FAILURE;      }  *while* (!fin.eof())      {          int tmp;          fin >> tmp;          arr.push\_back(tmp);      }      int num = 100;  *for* (int i = 0; i < 150; i++)      {          t1 = clock();          insertionsort(arr, num);          t2 = clock();          t3 = clock();          selectionsort(arr, num);          t4 = clock();          double insertiontime = double(t2 - t1) / double(CLOCKS\_PER\_SEC);          double selectiontime = double(t4 - t3) / double(CLOCKS\_PER\_SEC);          cout << endl;          cout << i + 1 << "  " << fixed << insertiontime << setprecision(5) << '\t';          cout << fixed << selectiontime << setprecision(5);          num += 100;      }      fin.close();  *return* 0;  }  Values.cpp  *#include* <iostream>  *#include* <cstdlib>  *#include* <cstdio>  using namespace std;  int main()  {  *for* (int i = 1; i <= 100000; i++)      {          cout << rand() << " ";      }  *return* 0;  } |
| **Observation ( SNAPSHOT)**  **Observation**  **For insertion sort**  The graph of the running time of insertion sort reveals that as the size of the input data increases, the running time also increases at a steady and consistent rate. The graph exhibits a characteristic upward trend, with the slope of the curve becoming steeper as the size of the input data increases. This is indicative of the quadratic nature of the running time of insertion sort.  **For selection sort**  The graph of the running time of selection sort shows that it also increases as the size of the input data increases, similar to insertion sort. However, the rate of increase is relatively slower compared to insertion sort, which is evident from the less steep slope of the graph. This suggests that selection sort is a more efficient algorithm for small input sizes. | |
| **Conclusion**  The running time of insertion sort is quadratic in nature, while that of selection sort is also quadratic but performs better on small inputs.  The experiment on finding the running time of insertion sort and selection sort shows that the running time of both algorithms is dependent on the size of the input data. | |