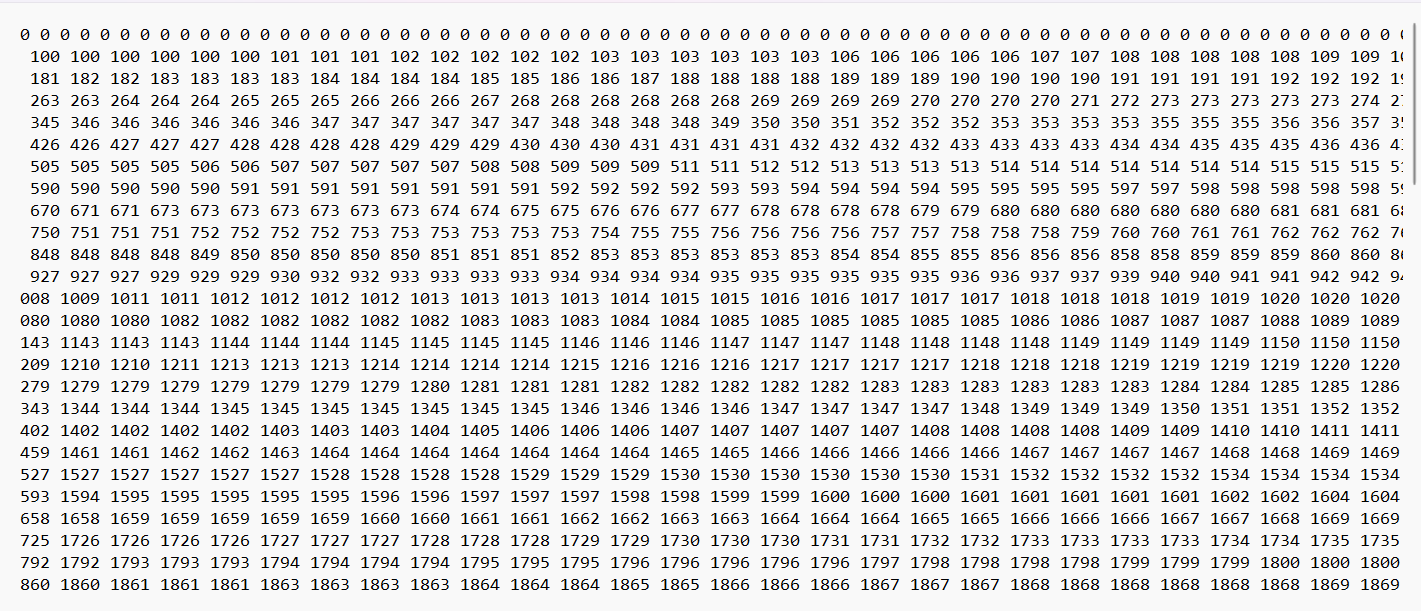
|  |  |
| --- | --- |
| Name: | Loukik Sainath Tayshete |
| Branch: | SY Data Science |
| Batch: | D4 |
| UID: | 2021700065 |
| Subject: | Design and Analysis Of Algorithm |
| Experiment No. | 1b |
| Date: | 12-02-2023 |

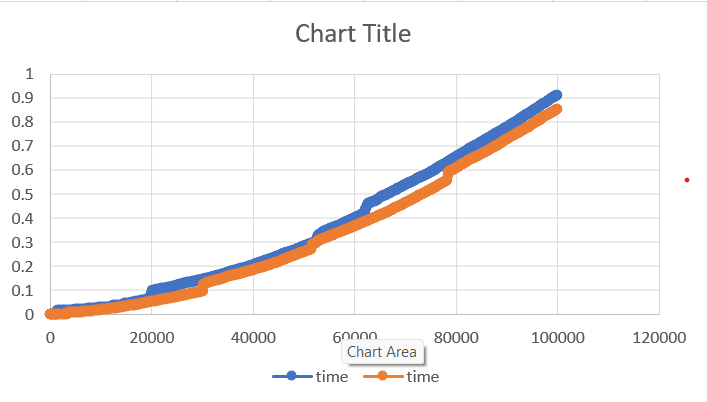
|  |  |
| --- | --- |
| Aim: | To find the running time of algorithm using insertion and selection. |

|  |
| --- |
| Selection Sort:  **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.  Selection Sort Algorithm:  Step 1 − Set MIN to location 0  Step 2 − Search the minimum element in the list  Step 3 − Swap with value at location MIN  Step 4 − Increment MIN to point to next element  Step 5 − Repeat until list is sorted  Code For Selection Sort:  #include<stdio.h>  #include<stdlib.h>  #include<time.h>  void swap(int \*a,int \*b) //Swapping of element is happpening  {      int temp = \*a;      \*a = \*b;      \*b = temp;  }  void printArray(int \*a,int n) //for traversing the array  {      FILE \* ptr = NULL;      ptr = fopen("selection.txt","a");      for (int i = 0; i < n; i++)      {          fprintf(ptr,"%d ",a[i]);      }      fprintf(ptr,"\n");      fclose(ptr);    }  void SelectionSort(int \*a,int n)  // This is for selection sort  {      int indexOfmin;      for(int i = 0 ; i < n - 1; i++) //This is for treaversing element from index 0      // Here i am doing n-1 because at worst it will get sorted in n-1  passes;      // where n is no of the array      {          indexOfmin = i; //index of min will start from index 0          for(int j = i + 1; j < n; j++)  //And now it will check from 0+1 = 1; and also after one index is done it will increament by one          {              if (a[indexOfmin] > a[j])              {                  indexOfmin = j;              }          }          swap(&a[i],&a[indexOfmin]);      }  }  int main()  {      srand(time(0));      FILE \* ptr = NULL;      FILE \* ptr1 = NULL;      ptr = fopen("selection.txt","a");      ptr1 = fopen("selection.csv","a");      int n;      int \*a;      // printf("Enter the size of array\n"); //THis is dynamic allocation array      // scanf("%d",&n);      a = (int \*) malloc(n \* sizeof(int));      for(n = 0 ; n <= 100000; n = n+100)      {          for(int i = 0; i < n ; i++)          {              int rando = rand() % n;              a[i] = rando;              rando;          }          printf("For %d\n",n);          int ticks = clock();          fprintf(ptr1,"%f\n",(float)ticks / CLOCKS\_PER\_SEC);      }      SelectionSort(a,n);      printArray(a,n);      fclose(ptr);      fclose(ptr1);  }  Graph For Selection Sort: |

|  |  |
| --- | --- |
| Insertion Sort:  **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.  Algorithm of Insertion Sort:  Step 1 - If the element is the first element, assume that it is already sorted. Return 1.  Step2 - Pick the next element, and store it separately in a key.  Step3 - Now, compare the key with all elements in the sorted array.  Step 4 - If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right.  Step 5 - Insert the value.  Step 6 - Repeat until the array is sorted.  Code For Insertion Sort:  #include<stdio.h>  #include<stdlib.h>  #include<time.h>  void printArray(int \*a,int n)  {      FILE \* ptr2 =NULL;      ptr2 = fopen("insertion.txt","a");      for(int i = 0 ; i < n ; i++)      {          fprintf(ptr2,"%d ",a[i]);      }      fprintf(ptr2,"\n");      fclose(ptr2);  }  void InsertionSort(int \*a, int n)  {      int key,j;      for(int i = 0; i < n; i++)      {          key = a[i];          j = i-1;          while(j >= 0 && a[j] > key)          {              a[j+1] = a[j];              j -- ; // j will get reset to 0 or it will run till j become 0          }          a[j+1] = key;      }  }  int main()  {      srand(time(0));      FILE \* ptr2 = NULL;      FILE \* ptr3 = NULL;      ptr2 = fopen("insertion.txt","a");      ptr3 = fopen("insertion.csv","a");      int n ;      int \*a;      // printf("Enter the size of array\n"); //THis is dynamic allocation array      // scanf("%d",&n);      a = (int \*) malloc(n \* sizeof(int));      for(n=0 ; n <=100000;n = n+100)      {          for(int i = 0; i < n ; i++)          {              int rando = rand() % n;              a[i] = rando;              rando;          }          printf("For %d\n",n);          int ticks = clock();          fprintf(ptr3,"%f\n",(float)ticks / CLOCKS\_PER\_SEC);      }      InsertionSort(a,n);      printArray(a,n);      fclose(ptr2);      fclose(ptr3);      return 0;  } | |
| Graph For Insertion Sort: |

Sorted 100000 integers by insertion sort:

Graph For Both Insertion and Selection (Combined):



Conclusion:

Sorted 100000 integer by both insertion and selection sort and observed their run-time. Selection sort run-time is more than that of Insertion.