|  |  |
| --- | --- |
| **Name** | Siddiqui Hawaiza Sabeel |
| **UID no.** | 2022701010 |
| **Experiment No.** | 1 B |
| **SUBJECT** | DAA |

|  |  |
| --- | --- |
| **AIM:** | Experiment on finding the running time of an algorithm. |
| **Program 1** | |
| **PROBLEM STATEMENT :** | For this experiment, you need to implement two sorting algorithms namely Insertion and Selection sort methods. Compare these algorithms based on time and space complexity. Time required to sorting algorithms can be performed using high\_resolution\_clock::now() under namespace std::chrono. You have to generate1,00,000 integer numbers using C/C++ Rand function and save them in a text file. Both the sorting algorithms uses these 1,00,000 integer numbers as input as follows. Each sorting algorithm sorts a block of 100 integers numbers with array indexes numbers A[0..99], A[0..199], A[0..299],…, A[0..99999]. You need to use high\_resolution\_clock::now() function to find the time required for 100, 200, 300…. 100000 integer numbers. Finally, compare two algorithms namely Insertion and Selection by plotting the time required to sort 100000 integers using LibreOffice Calc/MS Excel. The x-axis of 2-D plot represents the block no. of 1000 blocks. The y-axis of 2-D plot representsthe tunning time to sort 1000 blocks of 100,200,300,...,100000 integer numbers. Note – You have to use C/C++ file processing functions for reading and writing randomly generated 100000 integer numbers. |
| **ALGORITHM/**  **THEORY:** | **Step 1:** Start.  **Step 2:** Include the required libraries stdio.h, stdlib.h, time.h, and limits.h.  **Step 3:** Define two sorting functions as per problem statement selection\_sort and insertion\_sort.  **Step 4:** In the main function, using file handling open the file for writing.  **Step 5:** Generate 1000 blocks of 100 random numbers each and store them in the file.  **Step 6:** Close the file after writing.  **Step 7:** Open the file for reading.  **Step 8:** For each block of 100 elements, read the elements from the file into two arrays.  **Step 9:** Sort the elements of array using the selection\_sort function.  **Step 10:** Use clock() to measure the time taken by the algorithm, and store the value inside a variable.  **Step 11:** Sort the elements of array using the insertion\_sort function.  **Step 12:** Use clock() to measure the time taken by the algorithm, and store the value inside a variable.  **Step 13:** Display the number of blocks and time taken by both of the algorithm to sort a specific blocks.  **Step 14:** Repeat the process for 500 blocks.  **Step 15:** Close the file after reading.  **Step 16:** Stop. |
| **PROGRAM:** | #include<stdio.h>  #include<stdlib.h>  #include<time.h>  #include<limits.h>  void selection\_sort(int arr[],int size) {      for(int i=0; i<size-1; i++) {          int min=i;          for(int j=i+1; j<size; j++)              if(arr[j] < arr[min])                  min = j;          int temp = arr[min];          arr[min] = arr[i];          arr[i] = temp;      }  }  void insertion\_sort(int arr[],int n) {      int i,key,j;      for(int 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 main() {      FILE \*fp;      fp  = fopen ("exp1b.txt", "w");      srand((unsigned int) time(NULL));      for(int block=0; block<1000; block++) {          for(int i=0; i<100; i++) {              int number = (int)(((float) rand() / (float)(RAND\_MAX))\*100000);              fprintf(fp,"%d ",number);          }          fputs("\n",fp);      }      fclose (fp);     fp = fopen("exp1b.txt", "r");     printf("Block\tSelection\_sort\tInsertion\_sort\n");     for(int block=0; block<500; block++) {      clock\_t t,t1;        int arr[(block+1)\*100];      int arr1[(block+1)\*100];      for(int i=0; i<(block+1)\*100; i++){          fscanf(fp, "%d", &arr[i]);          arr1[i] = arr[i];      }      fseek(fp, 0, SEEK\_SET);      t = clock();      selection\_sort(arr,(block+1)\*100);      t = clock() - t;      t1 = clock();      insertion\_sort(arr1,(block+1)\*100);      t1 = clock() - t1;      double time\_taken\_selection\_sort = ((double)t)/CLOCKS\_PER\_SEC;      double time\_taken\_insertion\_sort = ((double)t1)/CLOCKS\_PER\_SEC;      printf("%d\t%f\t%f\n",(block+1),time\_taken\_selection\_sort,time\_taken\_insertion\_sort);     }      fclose(fp);  } |
|  | |
| **RESULT:**            **GRAPH:** | |
| **OBSERVATION:** | * The above graph represents the amount of time (in seconds) required to sort blocks of integers using the Selection sort & Insertion sort algorithm. * In the above graph, X-axis represents no. of blocks and Y-axis represents Time. The maximum no. of blocks is 500 on X-axis. * Maximum amount of time required to the sort 500th block using selection sort is approx. 4.92 seconds and using insertion sort is 2.57 seconds. * As the no. of integers in blocks increases both the lines for selection sort and insertion sort grow exponentially and not linearly. * No sudden major spikes were observed * The graph depicts that amount of time required to sort a block using selection sort increases quickly compared to insertion sort as no. of integers in the block increases. |
| **CONCLUSION:** | We found the running time of insertion sort and selection sort on each block and plotted a 2-D chart that shows the comparison of both algorithm’s running time. |