|  |  |
| --- | --- |
| NAME: | Darsh Shah |
| UID No. | 2021700057 |
| BRANCH: | SY CSE DS |
| BATCH: | D4 |
| SUBJECT | DAA |
| EXPERIMENT No. | 1 |
| DATE: | 6th February , 2023 |

|  |  |
| --- | --- |
| AIM: | Aim – To implement the various functions e.g. linear, non-linear, quadratic, exponential etc. |
| Program 1 | |
| PROBLEM STATEMENT : | A function is a relation between a set of inputs and a set of permissible outputs with the property that each  input is related to exactly one output. Let A & B be any two non-empty sets; mapping from A to B will be a function  only when every element in set A has one end, only one image in set B. |
| GRAPHS: | X  Y  X  Y  Graph 1 is for functions:- 2,4,5,10,11  Graph 2 is for functions:- 3,6,7,8,9  The extremity of large and small values in both the graphs have led to overlap in the region along the X axis.  Minimum values for both graphs is 0 , while maximum value for some become a very large number.  We have kept different graphs because of large values.  Graph 3 specifically for factorial function  X  Y |
| PROGRAM: | #include<stdio.h>  #include<math.h>  double func1(double i)  {  double r = 0.0;  r = pow(1.5 , i);  return r;  }  double func2(double i)  {  double r = 0.0;  r = pow(i , 3);  return r;  }  double func3(double i)  {  double r = 0.0;  r = i;  return r;  }  double func4(double i)  {  double r = 0.0;  r = pow(2 , i);  return r;  }  double func5(double i)  {  double e = 2.71;  double r = 0.0;  r = pow(e , i);  return r;  }  double func6(double i)  {  double r = 0.0;  r = log(i);  return r;  }  double func7(double i)  {  double r = 0.0;  r = log2(i);  return r;  }  double func8(double i)  {  double r = 0.0;  r = log(log(i));  return r;  }  double func9(double i)  {  double r = 0.0;  r = log2(log2(i));  return r;  }  double func10(double i)  {  double r = 0.0;  r = pow(2 , pow( 2 , i + 1));  return r;  }  double func11(double i)  {  if(i)  {  return ( i \* func11(i-1));  }  else  {  return 1;  }  }  int main()  {  int i=0;  double x=0;  printf("x = (3/2)^n \n");  for(i=0;i<=100;i=i+10) // x = (3/2)^n  {  x = func1(i);  printf("%d " , i);  printf("\t");  printf("%.2f \n" , x);  }  printf("\n");  printf("x = n^3 \n");  for(i=0;i<=100;i=i+10) // x = n^3  {  x = func2(i);  printf("%d " , i);  printf("\t");  printf("%.2f \n" , x);  }    printf("\n");  printf("x = n \n");  for(i=0;i<=100;i=i+10) // x = n  {  x = func3(i);  printf("%d " , i);  printf("\t");  printf("%.2f \n" , x);  }    printf("\n");  printf("x = 2^n \n");  for(i=0;i<=100;i=i+10) // x = 2^n  {  x = func4(i);  printf("%d " , i);  printf("\t");  printf("%.2f \n" , x);  }    printf("\n");  printf("x = e^n \n");  for(i=0;i<=100;i=i+10) // x = e^n  {  x = func5(i);  printf("%d " , i);  printf("\t");  printf("%.2f \n" , x);  }    printf("\n");  printf("x = loge n \n");  for(i=0;i<=100;i=i+10) // x = loge n  {  x = func6(i);  printf("%d " , i);  printf("\t");  printf("%.2f \n" , x);  }  printf("\n");  printf("x = log2 n \n");  for(i=0;i<=100;i=i+10) // x = log2 n  {  x = func7(i);  printf("%d " , i);  printf("\t");  printf("%.2f \n" , x);  }    printf("\n");  printf("x = loge loge n \n");  for(i=0;i<=100;i=i+10) // x = loge loge n  {  x = func8(i);  printf("%d " , i);  printf("\t");  printf("%.2f \n" , x);  }    printf("\n");  printf("x = log2 log2 n \n");  for(i=0;i<=100;i=i+10) // x = log2 log2 n  {  x = func9(i);  printf("%d " , i);  printf("\t");  printf("%.2f \n" , x);  }    printf("\n");  printf("x = 2 ^ 2 ^ (n + 1) \n");  for(i=0;i<=10;i=i+1) // x = 2 ^ 2 ^ (n + 1)  {  x = func10(i);  printf("%d " , i);  printf("\t");  printf("%.2f \n" , x);  }    printf("\n");  printf("x = n! \n");  for(i=0;i<=20;i=i+2) // x = 2 ^ 2 ^ (n + 1)  {  x = func11(i);  printf("%d " , i);  printf("\t");  printf("%.2f \n" , x);  }      } |
| RESULT: | x = (3/2)^n  0 1.00  10 57.67  20 3325.26  30 191751.06  40 11057332.32  50 637621500.21  60 36768468716.93  70 2120255184830.25  80 122264598055704.64  90 7050392822843069.00  100 406561177535215232.00  x = n^3  0 0.00  10 1000.00  20 8000.00  30 27000.00  40 64000.00  50 125000.00  60 216000.00  70 343000.00  80 512000.00  90 729000.00  100 1000000.00  x = n  0 0.00  10 10.00  20 20.00  30 30.00  40 40.00  50 50.00  60 60.00  70 70.00  80 80.00  90 90.00  100 100.00  x = 2^n  0 1.00  10 1024.00  20 1048576.00  30 1073741824.00  40 1099511627776.00  50 1125899906842624.00  60 1152921504606846976.00  70 1180591620717411303424.00  80 1208925819614629174706176.00  90 1237940039285380274899124224.00  100 1267650600228229401496703205376.00  x = e^n  0 1.00  10 21364.51  20 456442228.94  30 9751663938615.36  40 208339508359304896.00  50 4451071224011028955136.00  60 95094949571691216833085440.00  70 2031656870656286753797869404160.00  80 43405350743396877971235172998184960.00  90 927333990482707166514346952223881166848.00  100 19812035041219038986232623646666779910995968.00  x = loge n  0 -inf  10 2.30  20 3.00  30 3.40  40 3.69  50 3.91  60 4.09  70 4.25  80 4.38  90 4.50  100 4.61  x = log2 n  0 -inf  10 3.32  20 4.32  30 4.91  40 5.32  50 5.64  60 5.91  70 6.13  80 6.32  90 6.49  100 6.64  x = loge loge n  0 -nan  10 0.83  20 1.10  30 1.22  40 1.31  50 1.36  60 1.41  70 1.45  80 1.48  90 1.50  100 1.53  x = log2 log2 n  0 -nan  10 1.73  20 2.11  30 2.29  40 2.41  50 2.50  60 2.56  70 2.62  80 2.66  90 2.70  100 2.73  x = 2 ^ 2 ^ (n + 1)  0 4.00  1 16.00  2 256.00  3 65536.00  4 4294967296.00  5 18446744073709551616.00  6 340282366920938463463374607431768211456.00  7 115792089237316195423570985008687907853269984665640564039457584007913129639936.00  8 13407807929942597099574024998205846127479365820592393377723561443721764030073546976801874298166903427690031858186486050853753882811946569946433649006084096.00  9 inf  10 inf  x = n!  0 1.00  2 2.00  4 24.00  6 720.00  8 40320.00  10 3628800.00  12 479001600.00  14 87178291200.00  16 20922789888000.00  18 6402373705728000.00  20 2432902008176640000.00 |
| AIM: | Experiment on finding the running time of an algorithm. |
| 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 togenerate1,00,000 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. |
| GRAPHS | 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.  Y  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.  X  Y  The above graphs are plotted in Excel.  We can see that in Insertion sort, the value of time is small , ranging from 0 to 0.1  Minimum value is 0 while maximum is around 0.8  However in Selection sort,  The value of time increases are the numbers keep increasing.  Greater the value , greater the value of time.  It is directly proportional.  Selection sort has a higher slope compared to Insertion Sort.  Insertion sort is more efficient than Selection sort.  Insertion sort best case - O(n)  Selection sort best case - O(n2) |
| PROGRAM | #include <stdio.h>  #include <time.h>  #include <stdlib.h>  double populate(int a[], int b[], int n) {  clock\_t start, end;  double cpu\_time\_used;  start = clock();  for(int i = 0; i < n; i++)  {  int r = rand();  a[i] = b[i] = r;  }  end = clock();  FILE \*fp = fopen("./random.txt", "w+");  if(!fp) {  printf("Error opening file\n");  return -1;  }  for(int i = 0; i < n; i++) {  fprintf(fp, "%d\n", a[i]);  }  cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;  return cpu\_time\_used;  }  void swap(int \*x, int \*y) {  int t = \*x;  \*x = \*y;  \*y = t;  }  double selection(int a[], int n) {  FILE \*fp = fopen("./selection.csv", "w+");  double totalTime = 0;  if(!fp) {  printf("Error opening file\n");  return -1;  }  fprintf(fp, "n, time\n");  for (int i = 100; i <= n; i+=100)  {  // printf("%d\n", i);  clock\_t start, end;  double cpu\_time\_used;  start = clock();  for(int j = 0; j < i; j++) {  int min = j;  for(int k = j+1; k < i; k++) {  if(a[k] < a[min]) {  min = k;  }  }  swap(&a[j], &a[min]);  }  end = clock();  cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;  totalTime += cpu\_time\_used;  fprintf(fp, "%d, %f\n", i, cpu\_time\_used);  printf("Sorted from 0 to %d in %.2fs\n", i, cpu\_time\_used);    }  fclose(fp);  fp = fopen("./selection.txt", "w+");  for(int i = 0; i < n; i++) {  fprintf(fp, "%d\n", a[i]);  }  fclose(fp);  return totalTime;  }  double insertion(int a[], int n) {  FILE \*fp = fopen("./insertion.csv", "w+");  double totalTime = 0;  if(!fp) {  printf("Error opening file\n");  return -1;  }  fprintf(fp, "n, time\n");    for (int i = 100; i <= n; i+=100)  {  // printf("%d\n", i);  clock\_t start, end;  double cpu\_time\_used;  start = clock();  for(int j = 1; j < i; j++)  {  int k = j;  while(k > 0 && a[k] < a[k-1])  {  swap(&a[k], &a[k-1]);  k--;  }  }  end = clock();  cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;  totalTime += cpu\_time\_used;  fprintf(fp, "%d, %f\n", i, cpu\_time\_used);    printf("Sorted from 0 to %d in %.2fs\n", i, cpu\_time\_used);    }  fclose(fp);  fp = fopen("./insertion.txt", "w+");  for(int i = 0; i < n; i++) {  fprintf(fp, "%d\n", a[i]);  }  fclose(fp);  return totalTime;  }  int main()  {  int n = 100000;  int a[n],b[n];  double timeToPopulate = populate(a, b, n);  printf("Time taken to populate: %f\nSorting...\n", timeToPopulate);  double timeToSortI = insertion(a, n);  double timeToSortS = selection(b, n);  printf("Array sorted by insertion sort in %.2f\n", timeToSortI);  printf("Array sorted by selection sort in %.2f\n", timeToSortS);  printf("Total time taken to sort: %f\n", timeToSortI + timeToSortS);  return 0;  } |
| CONCLUSION: | Successfully studied various functions and sorts in C. |