|  |  |
| --- | --- |
| **Name** | Manish Shashikant Jadhav |
| **UID** | 2023301005 |
| **Subject** | Design and Analysis of Algorithms. |
| **Experiment No.** | 1.A |
| **Aim** | To implement the various functions e.g. linear, non-linear, quadratic, exponential etc. |
| **Code** | #include <stdio.h>  #include <math.h>  // Function prototypes  double linear(int n);                    // Returns n  double powerOfTwo(int n);                // Returns 2^n  double cubic(int n);                     // Returns n^3  double nLogN(int n);                     // Returns n \* log2(n)  double logFactorial(int n);              // Returns (lg n)! (Factorial of logarithm)  double lnLnN(int n);                     // Returns ln(ln(n))  double nTimesTwoToN(int n);              // Returns n \* 2^n  double powerOfTwoLog(int n);             // Returns 2^lg(n)  double lnN(int n);                       // Returns ln(n)  double exponentialTwoTwoNPlusOne(int n); // Returns 2^(2^n+1)  double factorial(int n);                 // Returns n!  // Function implementation of n  double linear(int n)  {    return n;  }  // Function implementation of 2^n  double powerOfTwo(int n)  {    return pow(2, n);  }  // Function implementation of n^3  double cubic(int n)  {    return pow(n, 3);  }  // Function implementation of n \* log2(n)  double nLogN(int n)  {    return n \* log2(n);  }  // Function implementation of (lg n)! (Factorial of logarithm)  double logFactorial(int n)  {    return tgamma(log2(n) + 1); // Using tgamma for gamma function (factorial )  }  // Function implementation of ln(ln(n))  double lnLnN(int n)  {    return log(log(n));  }  // Function implementation of n \* 2^n  double nTimesTwoToN(int n)  {    return n \* pow(2, n);  }  // Function implementation of 2^lg(n)  double powerOfTwoLog(int n)  {    return pow(2, log2(n));  }  // Function implementation of ln(n)  double lnN(int n)  {    return log(n);  }  // Function implementation of 2^(2^n+1)  double exponentialTwoTwoNPlusOne(int n)  {    return pow(2, pow(2, n) + 1);  }  // Function implementation of n!  double factorial(int n)  {    if (n == 0 || n == 1)      return 1;    else      return n \* factorial(n - 1);  }  int main()  {    int n;    // Calculate and display values for n from 0 to 100 with an increment of 1    for (n = 0; n <= 100; n++)    {      printf("n = %d:\n", n);      printf("1. n: %d\n", (int)linear(n));      printf("2. 2^n: %f\n", powerOfTwo(n));      printf("3. n^3: %f\n", cubic(n));      printf("4. n lg n: %f\n", nLogN(n));      printf("5. (lg n)! (Factorial of logarithm): %f\n", logFactorial(n));      printf("6. ln ln n: %f\n", lnLnN(n));      printf("7. n\*2^n: %f\n", nTimesTwoToN(n));      printf("8. 2^lg n: %f\n", powerOfTwoLog(n));      printf("9. ln n: %f\n", lnN(n));      printf("10. 2^(2^n+1): %f\n", exponentialTwoTwoNPlusOne(n));      printf("\n");    }    // Calculate and display values for n from 0 to 20 with an increment of 1 for the factorial function    printf("Factorial Function (n!):\n");    for (n = 0; n <= 20; n++)    {      printf("n = %d: %.0f\n", n, factorial(n));    }    return 0;  } |
| **Graph** |  |
| **Solved Problem:** | . |
| **Conclusion** | Hence, by completing this experiment I came to know about to implement the various functions e.g. linear, non-linear, quadratic, exponential etc. |

|  |  |
| --- | --- |
| **Experiment No.** | 1.B |
| **Aim** | Experiment on finding the running time of an algorithm(Insertion Sort and Selection Sort). |
| **Code** | #include <stdio.h>  #include <stdlib.h>  #include <time.h>  void insertSort(int \*arr, int n)  {    for (int j = 1; j < n; j++)    {      int key = arr[j];      int i = j - 1;      while (i > -1 && key < arr[i])      {        arr[i + 1] = arr[i];        i = i - 1;      }      arr[i + 1] = key;    }  }  void selectSort(int \*a, int n)  {    int temp;    for (int i = 0; i < n - 1; i++)    {      int p = i;      for (int j = i + 1; j < n; j++)      {        if (a[p] > a[j])        {          p = j;        }      }      if (p != i)      {        temp = a[i];        a[i] = a[p];        a[p] = temp;      }    }  }  void main()  {    srand(time(NULL));    FILE \*fileptr;    FILE \*fileptr1;    int n = 100000;    fileptr = fopen("inp.txt", "w");    int arr[n];    for (int j = 0; j < n; j++)    {      arr[j] = rand() % 100000 + 1;      fprintf(fileptr, "%d\n", arr[j]);    }    fclose(fileptr);    fileptr1 = fopen("time.csv", "w");    printf("Block Size\tInsertion Sort\tSelection sort\n");      fileptr = fopen("inp.txt", "r");    for (int p = 99; p < n; p = p + 100)    {      int array[p + 1];      int array1[p + 1];      for (int j = 0; j < p; j++)      {        array[j] = arr[j];        fscanf(fileptr, "%1d", &array1[j]);      }      clock\_t begin = clock();      insertSort(array, p + 1);      clock\_t mid = clock();      selectSort(array1, p + 1);      clock\_t end = clock();      double time\_spent = (double)(mid - begin);      double time\_spent1 = (double)(end - mid);      printf("%d\t%lf\t%lf\n", p + 1, time\_spent, time\_spent1);      fprintf(fileptr1, "%d,%lf,%lf\n", p + 1, time\_spent, time\_spent1);    }    printf("\n");  } |
| **Graph:** |  |
| **Insertion Sort**  **Pseudocode and example:** |  |
| **Selection Sort**  **Pseudocode and example:** |  |
| **Conclusion** | Hence, by completing this experiment I came to know about finding the running time of an algorithm(Insertion Sort and Selection Sort). |