|  |  |
| --- | --- |
| **Name:** | Siddiqui Hawaiza Sabeel |
| **UID No.** | 2022701010 |
| **Batch:** | D4 CSE DS |
| **Subject:** | DAA |
| **Experiment No.** | 1 |
| **Aim:** | To implement the various functions e.g. linear, non-linear, quadratic, exponential etc. |
| **Objectives:** | 1) To understand time Complexity Notations.  2) How to represent time Complexity using Functions. |

|  |  |
| --- | --- |
| **Theory:** | 1. **Time complexity:** Time complexity is the measure of the amount of time an algorithm takes to run as a function of the size of its input. It's used to analyze the efficiency of algorithms and compare different algorithms for the same problem. 2. **Different notations in time complexities:**    1. **Big O Notation**: It provides an upper bound on the time required for an algorithm to run. It represents the maximum time an algorithm would take for the worst-case scenario.    2. **Big Omega Notation**: It provides a lower bound on the time required for an algorithm to run. It represents the minimum time an algorithm would take for the best-case scenario.    3. **Big Theta Notation**: It provides both upper and lower bounds on the time required for an algorithm to run. It represents the average time an algorithm would take for all cases.    4. **Little o Notation:** It represents a tight upper bound on the time required for an algorithm to run.    5. **Little omega Notation:** It represents a tight lower bound on the time required for an algorithm to run. 3. **Common Time Complexities :-**    1. **O(1)** - Constant time complexity.    2. **O(log n)** - Logarithmic time complexity.    3. **O(n)** - Linear time complexity.    4. **O(n log n)** - Log-linear time complexity.    5. **O(n^2**) - Quadratic time complexity.    6. **O(n^3)** - Cubic time complexity.    7. **O(2^n)** - Exponential time complexity.    8. **O(n!)** - Factorial time complexity.    9. **O(n^k)** - Polynomial time complexity, where k is a constant greater than 1    10. **O(k^n)** - Exponential time complexity, where k is a constant greater than |
| **Program:** | #include<stdio.h>  #include<math.h>  int func1(int n){  return n;  }  double func2(int n){  return pow(2,n);  }  double func3(double n){  return log2(n);  }  double func4(double n){  return n\*log2(n);  }  double func5(double n){  double a = log2(n);  return sqrt(a);  }  double func6(int n){  return round(pow((3.0/2.0),n));  }  int func7(int n){  return pow(n,3);  }  double func8(double n) {  return round(pow(2, log2(n)));  }  double func9(int n){  return n\*(pow(2,n));  }  double func10(int n){  return log2(log2(n));  }  long long factorial(int n) {  if (n == 0) {  return 1;  }  return n \* factorial(n-1);  }  int main(){  printf("\n\nFor function n: \n\n");  for(int n=0; n <= 100; n++){  printf("%d \t", n,func1(n));  }  printf("\n\nFor function 2^n: \n\n");  for(int n=0; n <= 100; n++){  printf("2^%d = %.2f \t\t", n,func2(n));  }  printf("\n\nFor function log2(n): \n\n");  for(int n=0; n <= 100; n++){  printf("log2(%d) = %.2f \t", n,func3(n));  }  printf("\n\nFor function n\*log2(n): \n\n");  for(int n=0; n<=100; n++){  printf("%d.log2(%d) = %.1f \t\t\t\t", n,n,func4(n));  }  printf("\n\nFor function log2(n)^2: \n\n");  for(int n=0; n<=100; n++){  printf("log2(%d)^2 = %.2f \t\t\t\t", n,func5(n));  }  printf("\n\nFor function (3/2)^n: \n\n");  for(int n=0; n <= 100; n++){  printf("(3/2)^%d = %.2f \t\t", n,func6(n));  }  printf("\n\nFor function n^3: \n\n");  for(int n=0; n <= 100; n++){  printf("%d^3 = %d \t\t", n,func7(n));  }  printf("\n\nFor function 2^log2(n): \n\n");  for(int n=0; n <= 100; n++){  printf("2^log2(%d) = %.2f \t", n,func8(n));  }  printf("\n\nFor function n.2^n: \n\n");  for(int n=0; n<=100; n++){  printf("%d.2^%d = %.1f \t\t\t\t", n,n,func9(n));  }  printf("\n\nFor function log2(log2(n)): \n\n");  for(int n=0; n<=100; n++){  printf("log2(log2(%d)) = %.2f \t\t\t\t", n,func10(n));  }  printf("\n\nFor n!: \n\n");  for(int n = 0; n<=20; n++){  printf("%d! = %d \t", n,factorial(n));  }  } |
| **Output:** | Function n:  For function 2^n:      For function log2(n):    For function n\*log2(n):      For function log2(n)^2:      For function (3/2)^n:    For function n^3:    For function 2^log2(n):    For function n.2^n:      For function log2(log2(n)):      For function n!: |
| **Excel Data:** |  |
| **Graphs: 1. Function 2^n** |  |
| **2. Function log2(n)** |  |
| **3. Function n\*log2(n)** |  |
| **4. Function log2(n)^2** |  |
| **5. Function (3/2)^n** |  |
| **6. Function n^3** |  |
| **7. Function 2^log2(n)** |  |
| **8. Function n\*2^n** |  |
| **9. Function log2(log2(n))** |  |
| **10. Square Root** |  |
| **11. n!** |  |
| **For all functions:** |  |
| **Conclusion:** | I learnt about the representation of time complexities using functions and how the the output  values vary when we change one by one the input values |