**Algorithm Efficiency Big O**

|  |
| --- |
| namespace BigO  {  class Program  {  static void Main(string[] args)  {  int n=1000; //where n is the number of elements in a data structure  //=============================================================================  //LINEAR LOOP  //\*\*\*\*\*\*\*\*\*\*\*  for (int i = 0; i < n; i++)  {  //application code  }  ///The number of iterations is directly proportional to n  ///n is the major factor in determining the efficiency of the algorithm  ///the higher n, the higher the efficiency  ///  // Efficiency is n or f(n) = n; where f is the efficiency function  //=============================================================================  //LOGARITHMIC LOOP  //POWER OF 2 LOOP  for (int i = 1; i < n; i= i\* 2)  {  //application code  }  ///for n =1000, the loop will cycle 10 times, since 2 to the power of 10 is 1024  ///so if you take the log of 1000 (base 2) the answer is 10  ///So the efficiency of this loop is log n base 2  ///  /// Efficiency f(n) = log(n)  /// ===============================================================================  ///You can acheive similar efficiency with the following code  ///  for (int i = n; i > 0; i = i/2)  {  //application code  //after each cyle i is divided by 2  //the values of i are: 1000, 500, 250, 125, 62, 31, 15, 7, 3, 1  //this loop cycles just 10 times which is log of 1000  // Efficiency f(n) = log(n)  }  //==================================================================================  //NESTED LOOPS  //LINEAR LOGARITHMIC  for (int i = 0; i < n; i++)  {  for (int j = 1; j < n; j = j \* 2)  {  //application code  }    }  //the inner loop is Logarithmic (log(n)) since j is doubled every cycle  //the outer loop is linear (n)  //the overall is n\*log(n)  //==================================================================================  //QUADRATIC  for (int i = 0; i < n; i++)  {  for (int j = 0; j < n; j++)  {  //application code  }  }  // the total number of cycles is n \* n or n square  // Efficiency = n\*n  //==================================================================================  int[] a = { 3, 2, 7, 9, 4, 1, 12, 22, 8, 11, 25, 20, 32, 6, 5, 14 };  Console.Write("Unsorted array:");  Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  Display(a);  System.Diagnostics.Stopwatch sw = new System.Diagnostics.Stopwatch();  sw.Start();  BubbleSort(a);  sw.Stop();  Display(a);  Console.WriteLine("\n\ntime elapsed: {0} ticks", sw.ElapsedTicks);  Console.ReadKey();  }//end of main    ///The above efficiency are expressed using the Big-Oh notation  /// O(n)=n O(n)=logn O(n)=nlogn O(n)=n\*n  ///  ///There are other efficiency measurement  ///  ///POLYNOMIAL: O(n to the k)  ///EXPONENTIAL: O(c to the n)  ///FACTORIAL: O(n!)  ///  static void BubbleSort(int[] array)  {  for (int i = 1; i < array.Length; i++)  {  for (int j = 0; j < array.Length - i; j++)  {  if (array[j] > array[j + 1])  swap(ref array[j], ref array[j + 1]);  }  }  }  static void swap(ref int a1, ref int a2)  {  int temp = a1;  a1 = a2;  a2 = temp;  }  static void Display(int[] array)  {  Console.WriteLine("Bubble Sorted:");  Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  foreach (int n in array)  Console.Write("{0}, ", n);    }  }  }  ///EXERCISE:  ///\*\*\*\*\*\*\*\*\*  ///Write a pseudocode (or a program for an algorithm that receives an numeric value  ///and then displays the number of digits in the integer and the sum of the digits.  ///Example: 12.345 would display 'there are 5 digits with sum of 15'  ///State its efficiency.  ///  ///Exercise:  ///\*\*\*\*\*\*\*\*\*  ///Write a pseudocode (or a program) for an algorithm that receives a positive integer and then  ///display all the prime numbers up to the integer.  ///State its efficiency |

**Given n points on a plane, find pair that is closest**

|  |
| --- |
| class Program  {  static void Main(string[] args)  {  //Exercise 1  ///Give a list of n points in the plane (x0,y0)......(xn,yn)  ///find pair that is closest  ///  Point[] points = GetArrayOfPoints();  Point p1,p2;  double minDistance = GetMinDistance(points, out p1, out p2);  Console.WriteLine("\nmin distance is: {0}",  minDistance.ToString("n4"));  Console.WriteLine("\nbetween Points p1({0},{1}) and p2({2},{3})",  p1.X, p1.Y, p2.X, p2.Y);  Console.ReadKey();  }  static double GetMinDistance(Point[] points, out Point p1, out Point p2)  {  double minDistance = CalculateDistance(points[0], points[1]);  p1 = points[0];  p2 = points[1];  Console.WriteLine("mindistance: {0} p1({0},{1}) and p2({2},{3})",  minDistance, p1.X, p1.Y, p2.X, p2.Y);  DisplayPoints(points);  for (int i = 0; i < points.Length - 1; i++)  {  for (int j = i + 1; j < points.Length; j++)  {  double d = CalculateDistance(points[i], points[j]);  if (d < minDistance)  {  minDistance = d;  p1 = points[i];  p2 = points[j];  // Console.WriteLine("mindistance: {0} p1({0},{1}) and  p2({2},{3})",minDistance, p1.X, p1.Y, p2.X, p2.Y);  }  }  }  return minDistance;  }  static Point[] GetArrayOfPoints()  {  Random rand = new Random();  Point[] points = new Point[20];  for (int i = 0; i < points.Length; i++)  {  points[i] = new Point(rand.Next(10,100), rand.Next(10,100));  }  return points;  }  static double CalculateDistance(Point p1, Point p2)  {  return Math.Sqrt(Math.Pow(p1.X - p2.X,2)+ Math.Pow(p1.Y- p2.Y,2));  }  static void DisplayPoints(Point[] points)  {  Console.WriteLine("\nDisplaying All the Points..../n");  foreach (Point p in points)  {  Console.WriteLine("({0},{1})", p.X, p.Y);  }  }  } |

**Sequential Search / Bubble sort/ Binary Search**

|  |
| --- |
| class Program  {  static Random rand = new Random();  static void Main(string[] args)  {  int[] intArray = new int[50];  InitializeList(intArray);  Console.WriteLine("array before bubble sort: ");  DisplayArray(intArray);  Console.WriteLine("\narray after bubble sort: ");  BubbleSort(intArray);  DisplayArray(intArray);  //search using binary search  Console.Write("\n\nEnter value to search: ");  int n;  int.TryParse(Console.ReadLine(), out n);  int index = BinarySearch(intArray, n);  if (index >= 0)  Console.WriteLine("{0} is found at index: {1}", n, index);  else  Console.WriteLine("{0} was not found", n);  Console.ReadKey();  }  static void InitializeList(int[] array)  {  for(int i=0; i<array.Length; i++)  {  //assign a random value to each element in the array  array[i] = rand.Next(10, 200);  }  }  //method to search for the first ocurrence of x within the array  //and return its index or -1 if not found  static int SequentialSearch(int[] array, int x)  {  for (int i = 0; i < array.Length; i++)  {  if (array[i] == x)  return i;  }  return -1;  }  //sorting the array using Bubble sort  static void BubbleSort(int[] array)  {  for(int i=1; i<=array.Length; i++)  {  for(int j=0; j<array.Length-i; j++)  {  if (array[j] > array[j + 1])  {  swap(array,j,j+1);  }  }  }  }  static void swap(int[] array,int i, int j)  {  int temp = array[i];  array[i] = array[j];  array[j] = temp;  }  //method to search using binary search  //for the binary search to work properly, you need to pass it  //a sorted array  static int BinarySearch(int[] array, int x)  {  int low = 0;  int high = array.Length - 1;  int middle = -1;  while (low <= high)  {  //calculate midpoint  middle = (low + high) / 2;  if (x > array[middle])  low = middle + 1;  else if (x < array[middle])  high = middle - 1;  else  break;  }  if (low <= high)  return middle;  else  return -1;  }  //display array  static void DisplayArray(int[] array)  {  for (int i = 0; i < array.Length; i++)  {  Console.WriteLine("array[{0}] = {1}", i, array[i]);  }  }  } |

**Dynamic Array**

|  |
| --- |
| namespace DynamicArray  {  public class MyList  {  //internal array  private int[] array;  //internal pointer. points to the last added element  private int cindex; //current index  //constructors  public MyList()  {  cindex = -1;  //create the array  array = new int[4];  }  public MyList(int capacity)  {  cindex = -1;  array = new int[capacity];  }  //indexer  public int this[int i]  {  get  {  if (i >= 0 && i < array.Length)  return array[i];  else  throw new IndexOutOfRangeException("Invalid Index");  }  }  //property Count  public int Count  { get { return cindex + 1; } }    //property Capacity  public int Capacity  {  get{return array.Length;}  }  //Methods  public void Add(int n)  {  if (cindex < array.Length - 1)  {  //inc cindex  cindex++;  //assign a value to the array at cindex  array[cindex] = n;  }  else  {  //double the capacity of the array  //create a temp array  int[] temp = new int[2 \* Capacity];  //copy the internal array to temp  for (int i = 0; i < array.Length; i++)  {  temp[i] = array[i];  }  //reposition the array to reference temp  array = temp;  //add the element to the array  //inc cindex  cindex++;  //assign a value to the array at cindex  array[cindex] = n;  }  }  public bool RemoveAt(int index)  {  if(index < 0 || index > cindex)  return false;  //shift all the elements below index up one position  for(int i=index+1; i <= cindex; i++)  {  array[i-1] = array[i];  }  //update cindex  cindex--;  return true;  }  public int IndexOf(int n)  {  //return the index of n within the array  //or -1 if not found  for (int i = 0; i < Count; i++)  {  if (array[i] == n)  return i;  }  //not found  return -1;  }  public bool Contains(int n)  {  for (int i = 0; i < Count; i++)  {  if (array[i] == n)  return true;  }  //not found  return false;  }  public void RemoveAll()  {  cindex = -1;  //give up array memory  array = new int[4];  }  public void BubbleSort()  {  int max = Count;  for (int i = 0; i < max-1; i++)  {  for (int j = 0; j < max - i; j++)  {  if (array[j] > array[j + 1])  {  int temp = array[j];  array[j] = array[j + 1];  array[j + 1] = temp;  }  }  }  }  public int IterativeBinarysearch(int n)  {  int low = 0;  int high = Count - 1;  int mid = -1;  while (low <= high)  {  mid = (low + high) / 2;  if (n > array[mid])  low = mid + 1;  else if (n < array[mid])  high = mid - 1;  else  break;//implied equal so exit the loop  }  if (low <= high)  return mid;  else  return -1;  }  }  } |

**Get number of digits and their sum in an input string**

|  |
| --- |
| class Program  {  static void Main(string[] args)  {  ///Given a string that contains integers  ///write an algorithm that display the numbers of digits in the  String and display the sum of all of its digits  ///  //request a string of digits  Console.Write("\n Enter a list of digits: ");  string digits = Console.ReadLine();  //check if digits contains valid data  long n;  bool valid = long.TryParse(digits, out n);  if (!valid)  {  Console.WriteLine("\n Invalid Data");  Console.ReadKey();  return;  }  //get the number of digits  int numOfDigits = digits.Length;  //add all the digits  long sum = 0;  for (int i = 0; i < numOfDigits;i++ )  {  //convert character to byte  byte d = byte.Parse(digits[i].ToString());  sum += d;  }  //OR  //foreach (char c in digits)  //{  //convert character to byte  // byte d = byte.Parse(c.ToString());  // sum += d;  //}  Console.WriteLine("\n Number of digits: {0}", numOfDigits);  Console.WriteLine("\n Sum of all digits: {0}", sum);  Console.ReadKey();  }  } |