­­COIS 2020H Assignment 2 For Sri

# **Submission boxes**

For each method show your source code and testing of that code. (Note that some of these are trivial, in the lab, or trivial variations on other methods which is why there are so many). If you use a method in main you only need to show it once.

ArrayList: each listed method.

| 1. A constructor: (Public ArrayList (T) ) this can create an empty list  public ArrayList(int size)  {  array = new T[size+1];  count = 0;  } |
| --- |
| 2. Private Grow: this should double the size of the array when it Grows – this will get called by AddFront or AddLast if the array is out of space.  private void Grow()  {  T[] newArray;  if (array.Length != 0)  {  newArray = new T[array.Length \* 2];  }  else  {  newArray = new T[array.Length + 2];  }  for (int i = 0; i < array.Length; i++)  newArray[i] = array[i];  array = newArray;  } |
| 3. AddFront: this should add an item at the front (index 0) of the list (and moves the rest of the list out of the way first)  public void AddFront(T item)  {  if (GetCount() == array.Length-1)  Grow();  for (int i = array.Length-1; i > 0; i--)  array[i] = array[i - 1];  array[0] = item;  count++;  } |
| 4. AddLast: this should add an item after all current items (this isn’t Length -1 though). You’ll probably need GetCount for this |
| 5. GetCount: this should return the number of items in the list (not the length of the array)public int GetCount()  {  int count = 0;  for (int i = 0; i < array.Length; i++)  {  if (array[i] != null)  count++;  }  return count;  } |
| 6. InsertBefore: this takes two parameters, it will insert the first object \*before\* the object in the second parameter.  public void InsertBefore(T item, T beforeItem)  {  int index = Array.IndexOf(array, beforeItem);  if (index == -1)  throw new ArgumentException("Item not found");  if (GetCount() == array.Length)  Grow();  for (int i = count; i > index; i--)  array[i] = array[i - 1];  array[index] = item;  count++;  } |
| 7. InPlaceSort: This should sort the array list in place without making a new Array, use built in array sorting for this for now. (Make sure your Animals implement icomparable, and compare on Name)  public void InPlaceSort()  {  Array.Sort(array, 0, count);  }  Console.WriteLine("Sorting cats => ");  cat.InPlaceSort();  Console.WriteLine(cat.StringPrintAllReverse());  Console.WriteLine("Sorting snakes => ");  snake.InPlaceSort();  Console.WriteLine(snake.StringPrintAllReverse()); |
| 8. Swap(index1, index2): should swap the two elements in the array list  public void Swap(int index1, int index2)  {  T item = array[index1], beforeItem = array[index2];  array[index1] = beforeItem;  array[index2] = item;  } |
| 9. DeleteFirst: this should delete the object at position 0, and shuffle all the elements back one to fill the hole  public void DeleteFirst()  {  if (count == 0)  throw new InvalidOperationException("List is empty");  for (int i = 1; i < count; i++)  {  array[i - 1] = array[i];  }  count--;  } |
| 10. DeleteLast: Should delete the last non null element (and update the count correctly).  public void DeleteLast()  {  if (count == 0)  throw new InvalidOperationException("List is empty");  array[count - 1] = default(T);  count--;  } |
| 11. RotateLeft: Should rotate all elements in the array left. An example of RotateLeft: Given a list {A, B, C, D} rotate all elements left one place becomes {B, C, D, A}  public void RotateLeft()  {  if (count <= 1)  return;  T firstItem = array[0];  for (int i = 0; i < count - 1; i++)  array[i] = array[i + 1];  array[count - 1] = firstItem;  } |
| 12. RotateRight: Reverse of RotateLeft {A, B, C, D} -> {D, A, B, C}  public void RotateRight()  {  if (count <= 1)  return;  T lastItem = array[count - 1];  for (int i = count - 1; i > 0; i--)  array[i] = array[i - 1];  array[0] = lastItem;  } |
| 13. Public Static ArrayList Merge: this should take two ArrayLists, and return a third which is the first two merged in an unsorted order. (Remember to delete the first two if and only if you’re done with them, which you most likely won’t be)  public ArrayList <T> Merge(ArrayList<T> list1, ArrayList<T> list2)  {  ArrayList<T> mergedList = new ArrayList<T>(list1.count + list2.count);  for (int i = 0; i < list1.GetCount(); i++)  {  mergedList.AddLast(list1.array[i]);  }  for (int j = 0; j < list2.GetCount(); j++)  {  mergedList.AddLast(list2.array[j]);  }  return mergedList;  } |
| 14. StringPrintAllForward: This returns a string which is the whole structure printed from beginning to end (calling the object.ToString or similar  public string StringPrintAllForward()  {  string toPrint = "";  if (count != 0)  {  for (int i = 0; i < count; i++)  {  // if(array[i]!=null)  toPrint += array[i].ToString() + "\n";  }  }  else  {  toPrint = default;  }  return toPrint;    } |
| 15. StringPrintAllReverse: Same as above but reverse order.  public string StringPrintAllReverse()  {  string toPrint = "";  if (count != 0)  {  for (int i = count - 1; i >= 0; i--)  {  toPrint = toPrint + array[i].ToString() + "\n";  }  }  else  {  toPrint = default;  }  return toPrint;  } |
| 16. Deleteall  public void DeleteAll()  {  for (int i = 0; i < array.Length; i++)  {  array[i] = default;  }  count = 0;  } |

Main:

(You may not need all the boxes). Show (by hand/equation editor) how you came up with the movement calculation and testing of it.

| Arraylist addfirst  for (int i = 0; i < 3; i++)  {  int id = i + 10;  string name = GetRandomUniqueName(catName, animalList);  Breed breed = (Breed)random.Next(Enum.GetValues(typeof(Breed)).Length);  Position pos = new Position(random.NextDouble() \* 30.0, random.NextDouble() \* 30.0, 0.0, 30, 30, 0.0);  Cat cats = new Cat(i + 1, name, random.NextDouble() \* 10.0, pos, breed);  cat.AddFront(cats);  //cat.AddLast(cats);  } |
| --- |
| Arraylist addlast  for (int i = 0; i < 6; i++)  {  string name = GetRandomUniqueName(snakeName, animalList);  int id = i + 10;  double length = random.NextDouble() \* 10.0;  bool venomous = random.Next(2) == 0;  Position pos = new Position(random.NextDouble() \* 30.0, random.NextDouble() \* 30.0, 0, 30, 30, 0);  Snake snakes = new Snake(i + 1, name, random.NextDouble() \* 10.0, pos, length, venomous);  snake.AddLast(snakes);  } |
| Arraylist merge the first two  //Merge the two lists.  ArrayList<Animal> animals = new ArrayList<Animal>(cat.GetCount() + snake.GetCount());  animals = animals.Merge(cat, snake); |
| PrintAllForward |
| PrintAllReverse |
| Creation of 10 birds  for (int i = 0; i < 10; i++)  {  int id = i;  string name = ((Bird.BirdNames)random.Next(Enum.GetValues(typeof(Bird.BirdNames)).Length)).ToString();  Position pos = new Position(random.NextDouble() \* 100.0, random.NextDouble() \* 70.0, random.NextDouble() \* 10.0, 100.0, 70.0, 10.0);  Bird birds = new Bird(i + 1, name, random.NextDouble() \* 10.0, pos);  bird.AddFront(birds);  } |
| Code for Cats and snakes hunting birds and testing to show they eat the birds  int numRound = 0;  bool birdsExist = true;  ArrayList<Animal> remainingBirds = new ArrayList<Animal>(10);  foreach (Animal b in bird)  {  remainingBirds.AddFront(b);  } // Create a separate list for remaining birds  while (birdsExist)  {  Console.Clear();  Console.SetCursorPosition(0, Console.CursorTop);  DrawSpace(200, 70);  DrawObjects(animals);  numRound++;  Console.WriteLine($"Round {numRound}");  bool anyBirdsEaten = false;  foreach (Animal animal in animals)  {  if (animal is Cat cats && HandleAnimalEating(cats))  {  anyBirdsEaten = true;  }  else if (animal is Snake snakes && HandleAnimalEating(snakes))  {  anyBirdsEaten = true;  }  }  foreach (Animal animal in animals)  {  if (animal is Bird birds)  {  birds.BirdMoveingRandomly();  }  }  if (!anyBirdsEaten)  {  birdsExist = false;  }  // Update the remaining birds list  remainingBirds.DeleteAll();  foreach (Animal animal in animals)  {  if (animal is Bird birdObj)  {  remainingBirds.AddFront(birdObj);  }  }  // Print the current state  Console.WriteLine("Cats:");  foreach (Cat c in cat)  {  Console.WriteLine(c);  }  Console.WriteLine("Snakes:");  foreach (Snake s in snake)  {  Console.WriteLine(s);  }  Console.WriteLine("Birds:");  foreach (Bird b in remainingBirds)  {  Console.WriteLine(b);  }  Console.WriteLine();  }  Console.WriteLine("All birds have been eaten.");  Console.WriteLine($"Total rounds: {numRound}");  bool HandleAnimalEating(Animal animal)  {  ArrayList<Animal> birdsWithinRange = BirdsWithInRange(animal, remainingBirds, animal.Range);  if (birdsWithinRange.GetCount() > 0)  {  foreach (Animal birds in birdsWithinRange)  {  if (birds is Bird birdObj)  {  animal.Eat(birdObj);  remainingBirds.DeleteItem(birdObj); // Update the remaining birds list  return true;  }  }  }  else  {  Animal nearestBird = FindNearestBird(animal, remainingBirds);  MoveAnimalsTowardsBird(animal, nearestBird, animal.Speed);  }  return false;  }  }  private static ArrayList<Animal> BirdsWithInRange(Animal animal, ArrayList<Animal> animals, double range)  {  ArrayList<Animal> birdsWithInRange = new ArrayList<Animal>(animals.GetCount());  foreach (Animal bird in animals.Where(a => a is Bird && Distance(animal.Pos, a.Pos) <= range))  {  birdsWithInRange.AddLast(bird);  }  return birdsWithInRange;  }  //Finding nearest birds within the Distance  private static Animal FindNearestBird(Animal animal, ArrayList<Animal> animals)  {  Animal nearestBird = animals  .OfType<Bird>()  .OrderBy(bird => Distance(bird.Pos, animal.Pos))  .FirstOrDefault();  return nearestBird;  }  //moving Animals Toward the birds  private static void MoveAnimalsTowardsBird(Animal animal, Animal bird, double speed)  {  if (bird == null)  {  return;  }  double dx = bird.Pos.X - animal.Pos.X;  double dy = bird.Pos.Y - animal.Pos.Y;  double dz = bird.Pos.Z - animal.Pos.Z;  double distance = Math.Sqrt(TheSquareDistance(dx, dy, dz));  double moveX = dx / distance \* speed;  double moveY = dy / distance \* speed;  double moveZ = dz / distance \* speed;  animal.Move(moveX, moveY, moveZ);  }  //distance Between the position  private static double Distance(Position animalPosition, Position birdPosition)  {  double dx = birdPosition.X - animalPosition.X;  double dy = birdPosition.Y - animalPosition.Y;  double dz = birdPosition.Z - animalPosition.Z;  return Math.Sqrt(TheSquareDistance(dx, dy, dz));  }  private static double TheSquareDistance(double dx, double dy, double dz)  {  return dx \* dx + dy \* dy + dz \* dz;  } |
| Simulation with a grid + testing  I added a grid.  “Knew that there should be more rounds to eat all the birds But I couldn't make the code properly even I couldn't find out Where I went wrong I needed help but I am stuck outside Peterborough.” |
| Anything else you think needs to be shown with testing or that I missed when writing these boxes |
|  |
|  |
|  |

Big Oh

| a. In the program, the number of operations rises in the first code segment as the size of the input grows. The inner loop also repeats itself a certain number of times for every time the outer loop runs. Because the number of operations is inversely proportional to the size of the input, we say that the time complexity is O(N). |
| --- |
| b. The number of operations increases with the amount of the input in the second code fragment as well, albeit more slowly than in the first. The inner loop also repeats itself a certain number of times for every time the outer loop runs. Although the inner loop runs more frequently, it does so in powers of 2. However, because the total number of operations is proportional to the size of the input, the time complexity is O(N). |
| C.  In this program, in the first loop, i runs from 1 to N, with each run doubling i. In the inner loop j runs from 0 to N-1, with an increment of one.  The runtimes and running total of sum show the program has factor O(Nlog N). |

Novel programming algorithms (code for the key algorithm + testing)

| 1.A  using System;  public class A1  {  public static void Main()  {  // Initialize random function  Random rnd = new Random();  // Generate an array of 10 elements  int[] array = new int[10];  // For loop to input 10 random numbers into the array  for (int i = 0; i < array.Length; i++)  {  array[i] = rnd.Next(0, 50);  Console.Write(array[i] + ", ");  }  Console.WriteLine();  // Get local maximum  int localMax = LocalMaximum(array, 0, array.Length - 1, array.Length);  Console.WriteLine($"Maximum is = {localMax}");  Console.ReadLine();  }  // Method to calculate the local maximum and return the result  private static int LocalMaximum(int[] array, int lower, int upper, int num)  {  int mid = lower + (upper - lower) / 2;  if ((mid == 0 || array[mid - 1] <= array[mid]) && (mid == num - 1 || array[mid] > array[mid + 1]))  {  return array[mid];  }  else if (mid > 0 && array[mid - 1] > array[mid])  {  return LocalMaximum(array, lower, mid - 1, num);  }  else  {  return LocalMaximum(array, mid + 1, upper, num);  }  }  }    1.B  using System;  public class B1  {  public static void Main()  {  // Initialize random function  Random rand = new Random();  // Generate an array of 10 elements  int[] array = new int[10];  // For loop to input 10 random numbers into the array  for (int i = 0; i < array.Length; i++)  {  array[i] = rand.Next(0, 50);  Console.Write(array[i] + ", ");  }  Console.WriteLine();  // Get local maximum  int localMax = FindLocalMaximum(array);  Console.WriteLine($"Maximum is = {localMax}");  Console.ReadLine();  }  // Method to calculate the local maximum and return the result  private static int FindLocalMaximum(int[] array)  {  int left = 0;  int right = array.Length - 1;  while (left <= right)  {  int mid = left + (right - left) / 2;  if ((mid == 0 || array[mid - 1] <= array[mid]) && (mid == array.Length - 1 || array[mid] >= array[mid + 1]))  {  int localMax = array[mid];  int i = mid - 1;  while (i >= left && array[mid] >= array[i])  {  if (array[mid] == array[i])  {  localMax = array[i];  }  i--;  }  int j = mid + 1;  while (j <= right && array[mid] >= array[j])  {  if (array[mid] == array[j])  {  localMax = array[j];  }  j++;  }  return localMax;  }  else if (mid > 0 && array[mid - 1] > array[mid])  {  right = mid - 1;  }  else  {  left = mid + 1;  }  }  return -1; // No local maximum found  }  } |
| --- |
| 2.A  using System;  public class A2  {  public static void Main()  {  Random rand = new Random();  int x = 7;  int[] array = new int[10];  for (int i = 0; i < array.Length - 1; i++)  {  array[i] = rand.Next(0, 10);  Console.Write(array[i] + ", ");  }  Console.WriteLine();  Console.WriteLine("x = " + x);  bool check = false;  for (int i = 0; i < array.Length - 1; i++)  {  int sum = array[i];  for (int j = i + 1; j < array.Length; j++)  {  sum += array[j];  if (sum == x)  {  Console.WriteLine("x = " + array[i] + ", " + array[j]);  check = true;  break;  }  }  if (check)  {  break;  }  }  if (!check)  {  Console.WriteLine("No two of the numbers in the array add up to x");  }  Console.ReadLine();  }  }    2.B  using System;  public class B2\_  {  public static void Main()  {  Random rand = new Random();  int x = 7;  int[] array = new int[10];  for (int i = 0; i < array.Length; i++)  {  array[i] = rand.Next(0, 10);  Console.Write(array[i] + ", ");  }  Console.WriteLine();  Console.WriteLine("x = " + x);  bool check = false;  InsertionSort(array);  Console.WriteLine("Sorted array =>");  for (int i = 0; i < array.Length; i++)  {  Console.Write(array[i] + ", ");  }  Console.WriteLine();  int left = 0;  int right = array.Length - 1;  while (left < right)  {  int sum = array[left] + array[right];  if (sum == x)  {  Console.WriteLine("x = " + array[left] + ", " + array[right]);  check = true;  break;  }  else if (sum < x)  {  left++;  }  else  {  right--;  }  }  if (!check)  {  Console.WriteLine("No two of the numbers in the array add up to x");  }  Console.ReadLine();  }  private static void InsertionSort(int[] items)  {  for (int i = 0; i < items.Length - 1; i++)  {  for (int j = i + 1; j > 0; j--)  {  if (items[j - 1].CompareTo(items[j]) > 0)  {  int temp = items[j - 1];  items[j - 1] = items[j];  items[j] = temp;  }  }  }  }  } |
| 3.  using System;  public class A3  {  public static void Main()  {  int[] array = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  int K = 6;  int count = 0;  for (int i = 0; i < array.Length; i++)  {  Console.Write(array[i] + " ");  }  Console.WriteLine();  Console.WriteLine("K = " + K);  Console.WriteLine("Sub arrays =>");  for (int i = 0; i < array.Length; i++)  {  int sum = array[i];  for (int j = i + 1; j < array.Length; j++)  {  sum += array[j];  if (sum < K)  {  count++;  for (int w = i; w <= j; w++)  {  Console.Write(array[w] + " ");  }  Console.WriteLine();  }  }  }  Console.ReadLine();  }  } |

# **Programming Question**

In Assignment 1 you created an Array and a Linked List using built in arrays and linked lists of Mobile Objects which you defined yourself.

In this assignment, you are to build your own ArrayList (this is basically the generic array from the labs) (the next assignment will have a doubly linked list).

# **Animals:**

Two types of these are the basically same as assignment one (Cats and Snakes). Generate their starting positions in the range 0 to + 30 in x and y (z=0). Cats and Snakes need a new method Eat, which takes a parameter of type “Bird” (A type of animal) and deletes it. If a cat/snake gets close enough to a bird it will eat it (details discussed later).

You do need to make the objects comparable -for now simply make them comparable by Name (so they can be sorted Alphabetically by name). (<https://docs.microsoft.com/en-us/dotnet/api/system.icomparable?view=net-5.0>)

You are also going to make a new animal subclass: Bird. Birds can have the following names: {“Tweety”, “Zazu”, “Iago”, “Hula”, “Manu”, “Couscous”, “Roo”, Tookie”, “Plucky”, “Kiwi”} Generate their starting positions in the range 0 to + 100 in X and 0 – 70 in Y, and between 0 and 10 in Z, generate their ID’s in order (as in the first object is id 0, the second id 1, etc.).

From their starting positions, birds will move randomly, at most +/-10 in x/y and +/- 2 in Z. (Clamped to edges described above).

An explanation of what these do is in the discussion of main below.

You will need a Collection, most of which you implement yourself. A List built using an array (essentially this is the generic array from the labs).

# **ArrayList (Of Generics) (Based on Lab4):**

17. A constructor: (Public ArrayList (T) ) this can create an empty list

18. Private Grow: this should double the size of the array when it Grows – this will get called by AddFront or AddLast if the array is out of space.

19. AddFront: this should add an item at the front (index 0) of the list (and moves the rest of the list out of the way first)

20. AddLast: this should add an item after all current items (this isn’t Length -1 though). You’ll probably need GetCount for this

21. GetCount: this should return the number of items in the list (not the length of the array)

22. InsertBefore: this takes two parameters, it will insert the first object \*before\* the object in the second parameter.

23. InPlaceSort: This should sort the array list in place without making a new Array, use built in array sorting for this for now. (Make sure your Animals are comparable, and compare on Name)

24. Swap(index1, index2): should swap the two elements in the array list

25. DeleteFirst: this should delete the object at position 0, and shuffle all the elements back one to fill the hole

26. DeleteLast: Should delete the last non null element (and update the count correctly).

27. RotateLeft: Should rotate all elements in the array left. An example of RotateLeft: Given a list {A, B, C, D} rotate all elements left one place becomes {B, C, D, A}

28. RotateRight: Reverse of RotateLeft {A, B, C, D} -> {D, A, B, C}

29. Public Static ArrayList Merge: this should take two ArrayLists, and return a third which is the first two merged in an unsorted order. (Remember to delete the first two if and only if you’re done with them, which you most likely won’t be)

30. StringPrintAllForward: This returns a string which is the whole structure printed from beginning to end (calling the object.ToString or similar)

31. StringPrintAllReverse: Same as above but reverse order.

32. Deleteall

# **Main Program**

Create an ArrayList of Animals containing 3 Cats using Addfront.

Create an ArrayList of Animals containing 3 snakes using Addlast.

Merge the two lists.

Test PrintallForward, PrintAllReverse on the new ArrayList.

Make an array of 10 birds randomly positioned choosing from the names and positioning rules provided.

Your main program should have a while loop. Each iteration over the animals list Each *Cat* or *Snake* will eat *Birds* that are in range (They should print off that they are eating something). *Cats* have a range of 8, snakes have a range of 3. If there is nothing in range to eat, they move towards the nearest bird, cats move at a speed of 16, snakes at a speed of 14. Then birds (that survive) will randomly move as per their rules above. This repeats until all birds have been eaten. Report how many rounds it took.

Note on movement: Speed is a straight-line distance/round, cats and snakes only move in X and Y, it’s up to you to figure out how to do that maths (linear algebra AHHHHH, but they actually aren’t that hard, direction vector, some Pythagorean theorem, nothing to it right?).

Test out each of the methods for your structures (basic test cases: Make sure it doesn’t crash if the list is empty, make sure it works in a normal use case, don’t worry about testing extremely large numbers of objects).

You may need/want to show the output of your debugger in addition to or instead of a String print, that’s fine.

Now, why did I ask for that weird size of 100x70 you might wonder? You’re going to change the console window size to 200x70 (hopefully that works), and then Console.SetCursorPosition(0, Console.CursorTop); you’re going to have your program draw the space and the objects in it – do something like B1 for Bird 1, C1 for Cat 1 etc. (since these are 2 chars you need 100 columns, 2 units wide each) And then every iteration of the loop it will refresh the console display (so you can watch the cats and snakes chase the birds). It should print say B1 at the X, Y position. Note: this is a bit tricky, the maximum console window should be 200 columns and 80 rows, but 80 rows won’t fit on screen usually, hence the 200x70. If for whatever reason you can’t get 200x70 to work, pick a size that works for you and explain how you came up with it. Remember that you need to print off by row (+Y can be down)– meaning you might need/want a single list of all cats, snakes and birds sorted by position in Y and then position in X in one array list….   
  
<https://stackoverflow.com/questions/888533/how-can-i-update-the-current-line-in-a-c-sharp-windows-console-app>

Might help – don’t try and use \r since that’s per line, each iteration of the main loop you want to go back to the top left corner (0,0) and rewrite from there.

Depending on how fast it runs you could use System.Threading and then sleep each iteration for a few milliseconds to make sure you can actually see what’s happening at each step.

You should try and record your program running using [https://www.screentogif.com](https://www.screentogif.com/) or<https://giphy.com/apps/giphycapture> or something that will record a (small) short video of it. Try to avoid OBS as anything but a last resort since that would just produce a huge file for not much benefit.

## **Big Oh Notation Simple problems.**

1. Give the order of growth (as a function of input size ‘*N’* ) of the running times of each of the following code fragments (show your work, including what you initially thought it was and why, and then how you verified it or what you were corrected to, and yes, these are trivial examples, I know you can find solutions on the web or just run the code, which you should do to verify/explain the answer):

**a.**

int sum = 0;

for (int n = N; n > 0; n /= 2)

for(int i = 0; i<n; i++)

sum++;

**b.**

int sum = 0;

for (int i = 1 i<N; i \*= 2)

for (int j = 0; j<i; j++)

sum++;

c.

int sum = 0;

for (int i = 1 i<N; i \*= 2)

for (int j = 0; j<N; j++)

sum++;

# 

# **Program/Algorithm Questions**

The Following three questions are sample interview questions for programmers, students should probably know how to answer them

Write programs for each of these problems but keep them simple – focus on just solving the algorithm problem, don’t worry about making them pretty or advanced error checking weird edge cases or that sort of thing. Show some minimal testing, such as the main test case described.

1. A) Write a program that, given an array a[] of *N* distinct integers, finds a local maximum: an index *i* such that a[i-1] < a[i] > a[i+1]. Your Algorithmshould use ~*2log N* compares in the worst case (this is a binary search variant).

B) Modify your algorithm to account for ‘plateaus’ (e**.**g. {1,2,3,4,5,5,5,5,5,5,4,3,2} as well.

2. A) Given an integer x and an unsorted array of integers, write a program to  determine whether two of the numbers in the array add up to x.

B) Modify your answer to be more efficient when the array is sorted (you should be able to efficiently search in this case).

3. Write a program that for a given array that will find all possible (continuous) sub arrays with a sum less than some input value K. Strictly speaking a sub array can contain only 1 element, but we’re interested in sub arrays of more than one element. E.g. given an array {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} and an input K =6, sub arrays {1,2}, {2,3} are both less than 6. If K = 7 {1,2,3} now becomes a valid sub array as well.