Chapter 19 Student Book Answers

19.1 What you should already know

1 a) Stack – see Key terms 10.1 page 238

b) Queue - see Key terms 10.1 page 238

c) Linked list - see Key terms 10.1 page 238

2 a) Stack – see 10.4.1 page 251

b) Queue – see 10.4.2 page 253

c) Linked list – see 10.4.3 page 255

3

```
DECLARE myList : ARRAY[0:19] OF INTEGER
DECLARE upperBound : INTEGER
DECLARE lowerBound : INTEGER
DECLARE index : INTEGER
DECLARE item : INTEGER
DECLARE found : BOOLEAN
upperBound ← 19
lowerBound ← 0
OUTPUT "Please enter item to be found"
INPUT item
found ← FALSE
index ← lowerBound
REPEAT
    IF item = myList[index]
      THEN
        found ← TRUE
    ENDIF
    index ← index + 1
UNTIL (found = TRUE) OR (index > upperBound)
IF found
  THEN
    OUTPUT "Item found"
  ELSE
    OUTPUT "Item not found"
ENDIF
```

4

```
DECLARE myList : ARRAY[0:19] OF INTEGER
DECLARE upperBound : INTEGER
DECLARE lowerBound : INTEGER
DECLARE index : INTEGER
DECLARE swap : BOOLEAN
DECLARE temp : INTEGER
DECLARE top : INTEGER
upperBound ← 19
lowerBound ← 0
top ← upperBound
REPEAT
    Swap ← FALSE
    FOR index = lowerBound TO top - 1
        IF myList[index] > myList[index + 1]
          THEN
           temp ← myList[index]
          myList[index] ← myList[index + 1]
          myList[index + 1] ← temp
          swap ← TRUE
        ENDIF
    NEXT
    top ← top -1
UNTIL (NOT swap) OR (top = 0)
```

5 Answers given

Activity 19A

Python

```
#Python program for Linear Search
#create array to store all the numbers
myList = [4, 2, 8, 17, 9, 3, 7, 12, 34, 21]

#enter item to search for
item = int(input("Please enter item to be found "))

found = False

for index in range(len(myList)):
   if(myList[index] == item):
         found = True

if(found):
   print("Item found")
else:
   print("Item not found")
```

VB

```
'VB program for Linear Search
Module Module1
    Public Sub Main()
        Dim index As Integer
        Dim item As Integer
        Dim found As Boolean
        'Create array to store all the numbers
        Dim myList() As Integer = New Integer() {4, 2, 8, 17, 9, 3, 7, 12, 34,
21}

        'enter item to search for
        Console.Write("Please enter item to be found ")
        item = Integer.Parse(Console.ReadLine())

        For index = 0 To myList.Length - 1
            If (item = myList(index)) Then
                found = True
            End If
        Next

        If (found) Then
            Console.WriteLine("Item found")
        Else : Console.WriteLine("Item not found")
        End If
        Console.ReadKey()
    End Sub
End Module
```

Java

```
//Java program Linear Search
import java.util.Scanner;
class ACTIVITY19A
{
   public static void main(String args[])
    {
      Scanner myObj = new Scanner(System.in);
      //Create array to store the all the numbers
     int myList[] = new int[] {4, 2, 8, 17, 9, 3, 7, 12, 34, 21};
     int item, index;
     boolean found = false;

     // enter item to search for
      System.out.println("Please enter item to be found ");
      item = myObj.nextInt();

      for (index = 0; index < myList.length - 1; index++)
      {
         if (myList[index] == item)
         {
           found = true;
         }
      }
      if (found)
     {
        System.out.println("Item found");
      }
     else
       {
            System.out.println("Item not found");
         }

    }
}
```

Activity 19B

4 comparisons to find the letter D

A and B take fewer comparisons

Activity 19C

Python

```
#Python program for Binary Search
#create sorted list to store all the numbers
myList = [16, 19, 21, 27, 36, 42, 55, 67, 76, 89]

#enter item to search for
item = int(input("Please enter item to be found "))

found = False
lowerBound = 0
upperBound = len(myList) - 1

while (not found) and (lowerBound <= upperBound):
   index = int((upperBound + lowerBound)/2)
   if(myList[index] == item):
         found = True
   if item > myList[index]:
         lowerBound = index + 1
   if item < myList[index]:
         upperBound = index - 1

if(found):
 print("Item found")
else:
 print("Item not found")
```

VB

```
'VB program for Binary Search
Module Module1
    Public Sub Main()
        Dim index, lowerBound, upperBound As Integer
        Dim item As Integer
        Dim found As Boolean
        'Create array to store all the numbers
        Dim myList() As Integer = New Integer() {16, 19, 21, 27, 36, 42,
55, 67, 76, 89}

        'enter item to search for
        Console.Write("Please enter item to be found ")
        item = Integer.Parse(Console.ReadLine())

        found = False
        lowerBound = 0
        upperBound = myList.Length - 1


        Do
            index = (upperBound + lowerBound) \ 2

            If (item = myList(index)) Then
                found = True
            End If

            If item > myList(index) Then
                lowerBound = index + 1
            End If

            If item < myList(index) Then
                upperBound = index - 1
            End If

        Loop Until (found) Or (lowerBound > upperBound)

        If (found) Then

            Console.WriteLine("Item found")

        Else : Console.WriteLine("Item not found")

        End If
        Console.ReadKey()
    End Sub
End Module
```

Java

```
//Java program Binary Search
import java.util.Scanner;
class ACTIVITY19C
{
   public static void main(String args[])
    {
      Scanner myObj = new Scanner(System.in);
      //Create array to store the all the numbers
     int myList[] = new int[] {16, 19, 21, 27, 36, 42, 55, 67, 76, 89};

     boolean found = false;
     int lowerBound = 0;
     int upperBound = myList.length - 1;
     int index;


     // enter item to search for
      System.out.println("Please enter item to be found ");
      int item = myObj.nextInt();

      do
     {
          index = (upperBound + lowerBound) / 2;
         if (myList[index] == item)
         {
           found = true;
         }
          if (item > myList[index])
          {
                lowerBound = index + 1;
          }
          if (item < myList[index])
          {
                upperBound = index - 1;
          }

         }

     while ((!found) && (upperBound >= lowerBound));

      if (found)
     {
        System.out.println("Item found");
      }
     else
       {
           System.out.println("Item not found");
         }

   }
}
```

Activity 19D

Python

```python
#Python program for Insertion Sort
myList = [4,46,43,27,57,41,45,21,14]

lowerBound = 0
upperBound = len(myList)

for index in range(lowerBound + 1, upperBound):
   key = myList[index]
   place = index -1
   if myList[place] > key:
         while place >= lowerBound and myList[place] > key:
               temp = myList[place + 1]
               myList[place + 1] = myList [place]
               myList[place] = temp
               place = place -1
         myList[place + 1] = key

#output the sorted array
print(myList)
```

VB

```vb
'VB program for insertion sort
Module Module1

    Sub Main()

        Dim myList() As Integer = New Integer() {4, 46, 43, 27, 57, 41, 45, 21, 14}
        Dim index, lowerBound, upperBound, place, myKey, temp As Integer
        upperBound = myList.Length - 1
        lowerBound = 0

        For index = lowerBound + 1 To upperBound
            myKey = myList(index)
            place = index - 1
            If myList(place) > myKey Then
                Do While (place >= lowerBound) And myList(place) > myKey
                    temp = myList(place + 1)
                    myList(place + 1) = myList(place)
                    myList(place) = temp
                    place = place - 1
                Loop
                myList(place + 1) = myKey
            End If
        Next

        'output the sorted array
        For index = 0 To myList.Length - 1
            Console.Write(myList(index) & " ")
        Next

        Console.ReadKey() 'wait for keypress

    End Sub

End Module
```

Java

```
public class ACTIVITY19D {

     public static void main(String args[]) {
         int myList[] = {4, 46, 43, 27, 57, 41, 45, 21, 14};

         int upperBound = myList.length;
         int lowerBound = 0;

         for (int index = lowerBound; index < upperBound; index++) {
            int myKey = myList[index];

             int insertPlace = 0;
             for (int place = index-1; place >= lowerBound; place--) {
                 if (myList[place] > myKey) {
                     myList[place+1] = myList[place];
                 } else {
                     insertPlace = place+1;
                     break;
                 }
             }

             myList[insertPlace] = myKey;
         }

         // output the sorted array
         System.out.println(java.util.Arrays.toString(myList));
     }

 }
```

Activity 19E

Python

```
stack = [None for index in range(0,10)]
basePointer = 0
topPointer = -1
stackFull = 10
item = None

def push(item):
   global topPointer
   if topPointer < stackFull - 1:
          topPointer = topPointer + 1
          stack[topPointer] = item
   else:
          print("Stack is full, cannot push")

def pop():
   global topPointer, basePointer, item
   if topPointer == basePointer -1:
          print("Stack is empty,cannot pop")
   else:
          item = stack[topPointer]
          topPointer = topPointer - 1

push(7)
push(32)
print (*stack)

pop()
print(item)

pop()
print(item)
pop()
print(*stack)

push(1)
print(*stack)
push(2)
print(*stack)
push(3)
print(*stack)
push(4)
print(*stack)
push(5)
print(*stack)
push(6)
print(*stack)
push(7)
print(*stack)
push(8)
print(*stack)
push(9)
print(*stack)
push(10)
print(*stack)
push(11)
```

Note:

There is an automated method for printing the stack in Python that has been included in the program. For VB and Java you could write the function that would print the stack, using a loop and use it to show the contents of the stack at each stage.

VB

```
'VB program for stack
Module Module1
    Public stack() As Integer = {Nothing, Nothing, Nothing, Nothing,
                 Nothing, Nothing, Nothing, Nothing, Nothing, Nothing,
Nothing}
    Public basePointer As Integer = 0
    Public topPointer As Integer = -1
    Public Const stackFull As Integer = 10
    Public item As Integer

    Sub Main()

        push(7)
        push(32)
        pop()
        Console.WriteLine(item)
        pop()
        Console.WriteLine(item)
        pop()
        push(1)
        push(2)
        push(3)
        push(4)
        push(5)
        push(6)
        push(7)
        push(8)
        push(9)
        push(10)
        push(11)
        Console.ReadKey() 'wait for keypress
    End Sub

    Sub pop()
        If topPointer = basePointer - 1 Then
            Console.WriteLine("Stack is empty, cannot pop")
        Else
            item = Stack(topPointer)
            topPointer = topPointer - 1
        End If
    End Sub

    Sub push(ByVal item)
        If topPointer < stackFull - 1 Then
            topPointer = topPointer + 1
            Stack(topPointer) = item
        Else
            Console.WriteLine("Stack is full, cannot push")
        End If
    End Sub
End Module
```

Java

```
//Java program for a stack
import java.util.Scanner;
class ACTIVITY19E
{
  public static int stack[] = new int[] {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
  public static int basePointer = 0;
  public static int  topPointer = -1;
  public static final int stackFull = 10;
  public static int item;

  static void push(int item)
  {
     if (topPointer < stackFull - 1)
     {
          topPointer = topPointer + 1;
          stack[topPointer] = item;
     }
     else
          System.out.println("Stack is full, cannot push");
  }

  static void pop()
  {
     if (topPointer == basePointer - 1)
                 System.out.println("Stack is empty,cannot pop");
     else
     {
           item = stack[topPointer];
           topPointer = topPointer - 1;
     }
  }

   public static void main(String args[])
    {
          push(7);
          push(32);
          pop();
          System.out.println(item);

          pop();
          System.out.println(item);
          pop();

          push(1);
          push(2);
          push(3);
          push(4);
          push(5);
          push(6);
          push(7);
          push(8);
          push(9);
          push(10);
          push(11);

    }
}
```

Activity 19F

Python

```
queue = [None for index in range(0,10)]
frontPointer = 0
rearPointer = -1
queueFull = 10
queueLength = 0
def enQueue(item):
    global queueLength, rearPointer
    if queueLength < queueFull:
            if rearPointer < len(queue) - 1:
                    rearPointer = rearPointer + 1
            else:
                    rearPointer = 0
            queueLength = queueLength + 1
            queue[rearPointer] = item
    else:
            print("Queue is full, cannot enqueue")
def deQueue():
    global queueLength, frontPointer, item
    if queueLength == 0:
            print("Queue is empty,cannot dequeue")
    else:
            item = queue[frontPointer]
            queue[frontPointer] = None
            if frontPointer == len(queue) - 1:
                    frontPointer = 0
            else:
                    frontPointer = frontPointer + 1
            queueLength = queueLength -1

enQueue(7)
enQueue(32)
print (*queue)

deQueue()
print(item)

deQueue()
print(item)
deQueue()
print(*queue)

enQueue(1)
print(*queue)
enQueue(2)
print(*queue)
enQueue(3)
print(*queue)
enQueue(4)
print(*queue)
enQueue(5)
print(*queue)
enQueue(6)
print(*queue)
enQueue(7)
print(*queue)
enQueue(8)
print(*queue)
enQueue(9)
print(*queue)
enQueue(10)
print(*queue)
enQueue(11)
```

Note:

There is an automated method for printing the stack in Python that has been included in the program. For VB and Java you could write the function that would print the stack, using a loop and use it to show the contents of the stack at each stage.

VB

```
'VB program for a queue
 Module Module1

    Public Dim frontPointer As Integer = 0
    Public Dim rearPointer As Integer = -1
    Public Const queueFull As Integer = 10
    Public Dim queueLength As Integer =  0
    Public Dim item As Integer

    Public Dim queue() As Integer = {Nothing,  Nothing, Nothing,
Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing}

   Public Sub Main()

         Console.ReadKey()
         enQueue(7)
         enQueue(32)

         deQueue()
         Console.WriteLine(item)
         deQueue()
         Console.WriteLine(item)
         deQueue()
         Console.ReadKey()
         enQueue(1)
         enQueue(2)
         enQueue(3)
         enQueue(4)
         enQueue(5)
         enQueue(6)
         enQueue(7)
         enQueue(8)
         enQueue(9)
         enQueue(10)
         enQueue(11)
         Console.ReadKey()

   End Sub
   Sub enQueue(ByVal item)
         If queueLength < queueFull Then
               If rearPointer < queue.length - 1 Then
                     rearPointer = rearPointer + 1
               Else
                     rearPointer = 0
               End If
               queueLength = queueLength + 1
               queue(rearPointer) = item
```

```
            Else
                  Console.WriteLine("Queue is full, cannot enqueue")

            End if
      End Sub

      Sub deQueue()
            If queueLength = 0 Then
                  Console.WriteLine("Queue is empty, cannot dequeue")
            Else
                  item = queue(frontPointer)
                  If frontPointer = queue.length - 1 Then
                        frontPointer = 0
                  Else
                        frontPointer = frontPointer + 1
                  End if
                  queueLength = queueLength - 1
            End If

      End Sub

   End Module
```

Java

```
   //Java program for a queue
   import java.util.Scanner;
   class Queue
   {
     public static int queue[] = new int[] {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
     public static int frontPointer = 0;
     public static int  rearPointer = -1;
     public static final int queueFull = 10;
     public static int queueLength = 0;
     public static int item;

     static void enQueue(int item)
     {
        if (queueLength < queueFull)
        {
          if (rearPointer < queue.length - 1)
                  rearPointer = rearPointer + 1;
            else
                  rearPointer = 0;
            queueLength = queueLength + 1;
            queue[rearPointer] = item;
        }
        else
            System.out.println("Queue is full, cannot enqueue");
     }

     static void deQueue()
     {
        if (queueLength == 0)
```

```
                System.out.println("Queue is empty,cannot dequeue");
        else
        {
              item = queue[frontPointer];
              if (frontPointer == queue.length - 1)
                    frontPointer = 0;
              else
                    frontPointer = frontPointer + 1;
              queueLength = queueLength - 1;
        }
    }

    public static void main(String args[])
     {
          enQueue(7);
          enQueue(32);
          deQueue();
          System.out.println(item);

          deQueue();
          System.out.println(item);
          deQueue();

          enQueue(1);
          enQueue(2);
          enQueue(3);
          enQueue(4);
          enQueue(5);
          enQueue(6);
          enQueue(7);
          enQueue(8);
          enQueue(9);
          enQueue(10);
          enQueue(11);

    }
}
```

Activity 19G, 19H and 19I

Python

```
#Python program for finding an item in a linked list

myLinkedList = [27, 19, 36, 42, 16, None, None, None, None, None, None,
None]
myLinkedListPointers = [-1, 0, 1, 2, 3, 6, 7, 8, 9, 10, 11, -1]
heapStartPointer = 5
startPointer = 4
nullPointer = -1

def delete(itemDelete):
   global startPointer, heapStartPointer
   if startPointer == nullPointer:
         print("Linked List empty")
   else:
         index = startPointer
         while myLinkedList[index] != itemDelete and index !=
nullPointer:
               oldindex = index
               index = myLinkedListPointers[index]
         if index == nullPointer:
               print("Item ", itemDelete, " not found")
         else:
               myLinkedList[index] = None
               tempPointer = myLinkedListPointers[index]
               myLinkedListPointers[index] = heapStartPointer
               heapStartPointer = index
               myLinkedListPointers[oldindex] = tempPointer

def insert(itemAdd):
   global startPointer, heapStartPointer
   if heapStartPointer == nullPointer:
         print("Linked List full")
   else:
         tempPointer = startPointer
         startPointer = heapStartPointer
         heapStartPointer = myLinkedListPointers[heapStartPointer]
         myLinkedList[startPointer] = itemAdd
         myLinkedListPointers[startPointer] = tempPointer


def find(itemSearch):
   found = False
   itemPointer = startPointer
   while itemPointer != nullPointer and not found:
         if myLinkedList[itemPointer] == itemSearch:
               found = True
         else:
               itemPointer = myLinkedListPointers[itemPointer]
   return itemPointer
```

```
#enter item to delete
item = int(input("Please enter item to delete from list "))
delete(item)
print(myLinkedList)
print(myLinkedListPointers)

#enter item to add
item = int(input("Please enter item to add to list "))
insert(item)
print(myLinkedList)
print(myLinkedListPointers)



#enter item to search for
item = int(input("Please enter item to be found "))
result = find(item)
if result != -1:
   print("Item found")
else:
   print("Item not found")
```

VB

```
'VB program for a linked list
 Module Module1

    Public Dim heapStartPointer As Integer = 5
    Public Dim startPointer As Integer = 4
    Public Const nullPointer As Integer = -1
    Public Dim item As Integer
    Public Dim index As Integer
    Public Dim itemPointer As Integer
    Public Dim result As Integer

    Public Dim myLinkedList() As Integer = {27, 19, 36, 42, 16, Nothing,
Nothing, Nothing, Nothing, Nothing, Nothing, Nothing}
    Public Dim myLinkedListPointers() As Integer = {-1, 0, 1, 2, 3, 6, 7, 8,
9, 10, 11, -1}

    Public Sub Main()

        'enter item to delete
        Console.Write("Please enter item to remove from list ")
        item = Integer.Parse(Console.ReadLine())
        delete(item)
        For index = 0 To myLinkedList.Length - 1
            Console.Write(myLinkedList(index) & " ")
        Next
        Console.WriteLine()
        For index = 0 To myLinkedListPointers.Length - 1
            Console.Write(myLinkedListPointers(index) & " ")
        Next
        Console.WriteLine()
        'enter item to insert
        Console.Write("Please enter item to add to list ")
        item = Integer.Parse(Console.ReadLine())
        insert(item)
        For index = 0 To myLinkedList.Length - 1
            Console.Write(myLinkedList(index) & " ")
        Next

        For index = 0 To myLinkedListPointers.Length - 1
            Console.Write(myLinkedListPointers(index) & " ")
        Next

    'enter item to search for
        Console.Write("Please enter item to be found ")
        item = Integer.Parse(Console.ReadLine())
        result = find(item)



        If result <> -1 Then
            Console.WriteLine("Item found")
        Else
            Console.WriteLine("Item not found")
        End If
        Console.ReadKey()
    End Sub

    Sub insert (ByVal itemAdd)
        Dim tempPointer As Integer
        If heapStartPointer = nullPointer Then
            Console.WriteLine("Linked List full")
        Else
            tempPointer = startPointer
            startPointer = heapStartPointer
            myLinkedList(startPointer) = itemAdd
            myLinkedListPointers(startPointer) = tempPointer

        End if
    End Sub

    Sub delete (ByVal itemDelete)
        Dim tempPointer, index, oldIndex  As Integer
        If startPointer = nullPointer Then
            Console.WriteLine("Linked List empty")
        Else
            index = startPointer
            While myLinkedList(index) <> itemDelete And index <> nullPointer
                Console.WriteLine( myLinkedList(index) & " " & index)
                Console.ReadKey()
                oldIndex = index
                index = myLinkedListPointers(index)
            End While

            if index = nullPointer Then
                Console.WriteLine("Item " & itemDelete & " not found")
            Else
```

```
                myLinkedList(index) = nothing
                tempPointer = myLinkedListPointers(index)
                myLinkedListPointers(index) = heapStartPointer
                heapStartPointer = index
                myLinkedListPointers(oldIndex) = tempPointer
            End if

         End If
     End Sub


     Function find(ByVal itemSearch As Integer) As Integer
         Dim found As Boolean = False
         itemPointer = startPointer
         While (itemPointer <> nullPointer) And Not found
             If itemSearch = myLinkedList(itemPointer) Then
                 found = True
             Else
              itemPointer = myLinkedListPointers(itemPointer)
             End if
          End While
         Return itemPointer
     End Function

 End Module
```

Java

```
 //Java program for a linked list
 import java.util.Scanner;
 class ACTIVITY19GHI
 {
   public static int myLinkedList[] = new int[] {27, 19, 36, 42, 16, 0, 0,0, 0,
 0, 0};
   public static int myLinkedListPointers[] = new int[] {-1, 0, 1, 2, 3, 6, 7, 8,
 9, 10, 11, -1};
   public static int startPointer = 4;
   public static int  heapStartPointer = 5;
   public static final int nullPointer = -1;

   static void delete(int itemDelete)
   {
       int oldIndex = -1;

       if (startPointer == nullPointer)
         System.out.println("Linked List is empty");
       else
       {

           int index = startPointer;
              while (myLinkedList[index] != itemDelete && index != nullPointer)
              {
                     oldIndex = index;
                     index = myLinkedListPointers[index];
              }
              if (index == nullPointer)
                    System.out.println("Item " + itemDelete + " not found");
              else
```

```
          {
                 myLinkedList[index] = 0;
                 int tempPointer = myLinkedListPointers[index];
                 myLinkedListPointers[index] = heapStartPointer;
                 heapStartPointer = index;
                 myLinkedListPointers[oldIndex] = tempPointer;
          }
   }
}


static void insert(int itemAdd)
{
   if (heapStartPointer == nullPointer)
   {
          System.out.println("Linked List is full");
   }
   else
   {
          int tempPointer = startPointer;
          startPointer = heapStartPointer;
          heapStartPointer = myLinkedListPointers[heapStartPointer];
          myLinkedList[startPointer] = itemAdd;
          myLinkedListPointers[startPointer] = tempPointer;
   }
}

static  int find(int itemSearch)
{
  boolean found = false;
 int itemPointer = startPointer;
 do
 {
        if (itemSearch == myLinkedList[itemPointer])
        {
              found = true;
        }
        else
        {
              itemPointer = myLinkedListPointers[itemPointer];
        }
 }
 while ((itemPointer != nullPointer) && !found);
 return itemPointer;
}
 public static void main(String args[])
  {
  Scanner input = new Scanner(System.in);

  System.out.println("Please enter item to remove from list ");
    int item = input.nextInt();
   delete(item);

   for (int index = 0; index < myLinkedList.length; index++)
          System.out.print(myLinkedList[index] + " ");
   System.out.println();

   for (int index = 0; index < myLinkedListPointers.length; index++)
          System.out.print(myLinkedListPointers[index] + " ");
        System.out.println();

         System.out.println("Please enter item to add to list ");
         item = input.nextInt();
        insert(item);

        for (int index = 0; index < myLinkedList.length; index++)
                System.out.print(myLinkedList[index] + " ");
        System.out.println();

        for (int index = 0; index < myLinkedListPointers.length; index++)
                System.out.print(myLinkedListPointers[index] + " ");
        System.out.println();


        System.out.println("Please enter item to be found ");
         item = input.nextInt();

         int result = find(item);

         if (result != -1 )
        {
           System.out.println("Item found");
         }
        else
          {
                System.out.println("Item not found");
             }

      }
    }
```

For 19H

startPointer	**heapStartPointer**	**itemAdd**	**tempPointer**
Already set to 4	Already set to 5	18	
5	6		4
6	7	25	5

The linked list, `myLinkedList` will now be as shown below.

		mylinkedList	myLinkedListPointers
	[0]	27	−1
	[1]	19	0
	[2]	36	1
	[3]	42	2
	[4]	16	3
	[5]	18	4
startPointer →	[6]	25	5
heapStartPointer →	[7]		8
	[8]		9
	[9]		10
	[10]		11
	[11]		−1

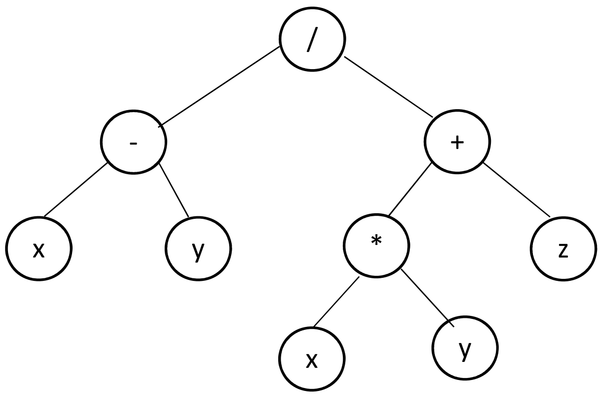
For 19I

startPointer	heapStartPointer	itemDelete	index	oldIndex	tempPointer
Already set to 6	Already set to 7		5	5	
6	4	18	4	4	2
			3		

The linked list, `myLinkedList` will now be as shown below.

		mylinkedList	myLinkedListPointers
	[0]	27	−1
	[1]	19	0
	[2]	36	1
	[3]	42	2
heapStartPointer →	[4]	16	7
	[5]	18	3
startPointer →	[6]	25	5
	[7]		8
	[8]		9
	[9]		10
	[10]		11
	[11]		−1

Activity 19J



Activity 19K

```
TYPE node
   DECLARE item : STRING
   DECLARE leftPointer : INTEGER
   DECLARE rightPointer : INTEGER
ENDTYPE
DECLARE myTree[0 : 49] OF node
DECLARE rootPointer : INTEGER
DECLARE nextFreePointer : INTEGER
```

Activity 19L

rootPointer	itemPointer	itemSearch
0	0	55
	2	
	3	
	5	
	8	

rootPointer	itemPointer	itemSearch
0	0	75
	2	
	3	
	5	
	-1	

Activity 19M

leftBranch	nextFreePointer	itemAddPointer	rootPointer	itemAdd	itemPointer	oldPointer
	10	10	0	25	0	0
	11					
TRUE					1	1
FALSE					6	6
FALSE					-1	

myTree	item	leftPointer	rightPointer
[0]	27	1	2
[1]	19	4	6
[2]	36	-1	3
[3]	42	-1	5
[4]	16	-1	7
[5]	89	8	-1
[6]	21	-1	10
[7]	17	-1	-1
[8]	55	-1	-1
[9]	18	-1	-1
[10]	25	-1	-1
[11]	-1		

Activity 19N

```
Path = {School, Town centre, Shopping centre, Gardens}
Path = {Town centre, School, Train station}
Path = {Town centre, Shopping centre, Gardens, Town centre}
```

Activity 19O

```
// a linked list to store names
TYPE linkedList
    DECLARE name : STRING
    DECLARE pointer : INTEGER
ENDTYPE

// initialise linked list
DECLARE myLinkedList : ARRAY [0:29] of linkedList
DECLARE heapStartPointer : INTEGER
DECLARE startPointer : INTEGER
DECLARE index : INTEGER
DECLARE nameToAdd : STRING
heapStartPointer ← 0
startPointer ← -1 // list empty
FOR index ← 0 TO 28
    myLinkedList[index].pointer ← index
NEXT index
myLinkedList[29].pointer ← -1

// add item to linked list
IF heapStartPointer = -1
  THEN
    OUTPUT "List Full"
  ELSE
    OUTPUT "Name to add"
    INPUT nameToAdd
    myLinkedList[heapStartPointer].name ← nameToAdd
    myLinkedList[heapStartPointer].pointer ← startPointer
    startPointer ← heapStartPointer
    heapStartPointer ← myLinkedList[heapStartPointer].pointer
ENDIF

REPEAT
    index ← startPointer
    OUTPUT myLinkedList[index].name
    Index ← myLinkedList[index].pointer
UNTIL index = -1
```

Activity 19P

Python

```python
#using a dictionary in Python
studentdict ={
     "Leon":27,
     "Ahmad":78,
     "Susie":64
     }
print (studentdict)

score =  studentdict["Leon"]
print(score)

studentdict["Mo"] = 99
print (studentdict)

del studentdict["Ahmad"]
print (studentdict)
```

VB

```vb
'using a dictionary in VB
Module Module1

    Sub Main()
        Dim studentdict As New Dictionary(Of String, Integer)
        Dim studentScore As Integer
        Dim studentName As String
        studentdict.Add("Leon", 27)
        studentdict.Add("Ahmad", 78)
        studentdict.Add("Susie", 64)
        For Each item As KeyValuePair(Of String, Integer) In studentdict
            studentName = item.Key
            studentScore = item.Value
            Console.WriteLine(studentName + "  " + studentScore.ToString)
        Next

        If studentdict.ContainsKey("Ahmad") Then
            studentScore = studentdict.Item("Ahmad")
            Console.WriteLine(studentScore)
        End If
        Console.WriteLine()

        studentdict.Remove("Ahmad")
        For Each item As KeyValuePair(Of String, Integer) In studentdict
            studentName = item.Key
            studentScore = item.Value
            Console.WriteLine(studentName + "  " + studentScore.ToString)
        Next

        Console.ReadKey()
    End Sub
End Module
```

Java

```
//Java program for a dictionary
import java.util.*;

class ACTIVITY19P
{
   public static void main(String[] args)
   {
         // creating a My HashTable Dictionary
         Hashtable<String, Integer> studentdict = new
Hashtable<String, Integer>();

         // using put method
         studentdict.put("Leon", 27);
         studentdict.put("Ahmad", 78);
         studentdict.put("Susie", 64);


         System.out.println(studentdict);

      // find value at key "Ahmad"
         System.out.println(studentdict.get("Ahmad"));

         // remove value at key "Ahmad"
         studentdict.remove("Ahmad");
         System.out.println(studentdict);
         }
}
```

Activity 19Q

1 a) see Figure 19.10

b) see Figure 10.12

2 a) an ADT that consists of pairs a key and a value, they key is used to find the value

b)

```
TYPE linkedList
  DECLARE item : STRING
  DECLARE Pointer : INTEGER
ENDTYPE
TYPE dictionary
  DECLARE key : myLinkedList : ARRAY [0:19] OF linkedList
  DECLARE value : ARRAY [0:19] OF STRING
ENDTYPE

DECLARE myDictionary : linkedList
DECLARE heapStartPointer : INTEGER
DECLARE startPointer : INTEGER
DECLARE index : INTEGER
```

3 For a linear search the time to perform a search will increase linearly as the number of items in the list increases, the time taken is directly proportional to the number of items in the list. In big O notation, this is O(N) where N is the number of items in the list.

For a binary search the time to perform a search will increase linearly as the number of items in the list increases exponentially, the time taken is logarithmically proportional to the number of items in the list. In big O notation, this is O(log N) where N is the number of items in the list.

Therefore a binary search becomes more time efficient than a linear search as the number of items in the list increases.

Activity 19R

Python

```
#Python program recursive factorial function
def factorial(number):
    if number == 0:
        answer = 1
    else:
        answer = number * factorial(number - 1)
    return answer


print(factorial(0))
print(factorial(5))
```

VB

```
'VB program recursive factorial function
Module Module1
    Sub Main()
        Console.WriteLine(factorial(0))
        Console.Writeline(factorial(5))
        Console.ReadKey()
    End Sub
    Function factorial(ByVal number As Integer) As Integer
          Dim answer As Integer
        If number = 0 Then
            answer = 1
        Else
            answer = number * factorial(number - 1)
        End If
          return answer
    End Function
End Module
```

Java

```
// Java program recursive factorial function
public class Factorial {

    public static void main(String[] args) {
        System.out.println(factorial(0));
        System.out.println(factorial(5));
    }
    public static int factorial(int number)
    {
        int answer;
          if (number == 0)
                 answer = 1;
        else
                 answer = number * factorial(number - 1);
        return answer;
    }
}
```

Call number	Function call	number	answer	RETURN
1	`factorial(5)`	5	`5 * factorial(4)`	
2	`factorial(4)`	4	`4 * factorial(3)`	
3	`factorial(3)`	3	`3 * factorial(2)`	
4	`factorial(2)`	2	`2 * factorial(1)`	
5	`factorial(1)`	1	`1 * factorial(0)`	
6	`factorial(0)`	0	1	1
5 continued	`factorial(1)`	1	`1 * 1`	1
4 continued	`factorial(2)`	2	`2 * 1`	2
3 continued	`factorial(3)`	3	`3 * 2`	6
2 continued	`factorial(4)`	4	`4 * 3 * 2`	24
1 continued	`factorial(5)`	5	`5 * 4 * 3 * 2`	120

Activity 19T

1 Recursion – see 19.2.1

2 Use of stack for recursive procedures – see 19.2.2

Activity 19S

```
FUNCTION fibonacci (number : INTEGER) RETURNS INTEGER
    IF number = 0 OR number = 1
      THEN
        answer ← 1 // base case
      ELSE
        answer ← Fibonacci(number - 1) + fibonacci (number - 2)
        // recursive call with general case
    ENDIF
    RETURN answer
ENDFUNCTION
```

Call number	Function call	number	answer	RETURN
1	`fibonacci(4)`	4	`fibonacci(3) + fibonacci(2)`	
2	`fibonacci(3)`	3	`fibonacci(2) + fibonacci(1)`	
3	`fibonacci(2)`	2	`fibonacci(1) + fibonacci(0)`	
4	`fibonacci(1)`	1	1	1
5	`fibonacci(0)`	0	1	1
3 continued	`fibonacci(2)`	2	1 + 1	2
2 continued	`fibonacci(3)`	3	1 + 2	3
1 continued	`fibonacci(4)`	4	3 + 2	5

End of chapter questions

1 a) i)

```
FOR ThisPointer ← 2 TO 10
   // use a temporary variable to store item which is
   to // be inserted into its correct location
   Temp ← NameList[ThisPointer]
   Pointer ← ThisPointer – 1
   WHILE (NameList[Pointer] > Temp) AND Pointer > 0
      // move list item to next location
      NameList[Pointer + 1] ← NameList[Pointer]
      Pointer ← Pointer - 1
   ENDWHILE
   // insert value of Temp in correct location
   NameList[Pointer] ← Temp
ENDFOR
```

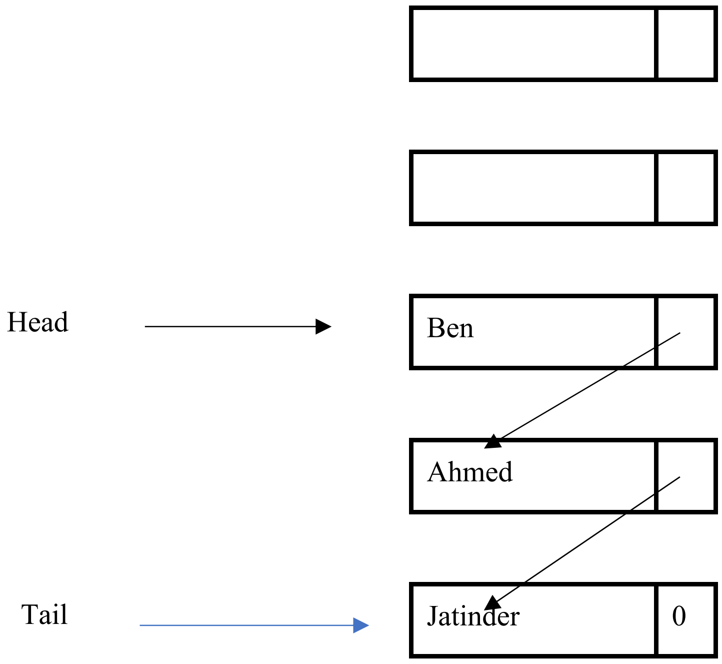
b) ii) The outer loop is always executed 9 times and the inner loop is never executed as `Temp` is always larger.

c) i) Both loops are always executed 9 times.

ii)

```
REPEAT
    NoMoreSwaps ← TRUE
    FOR Pointer ← 1 TO NumberOfItems – 1
        IF NameList[Pointer] > NameList[Pointer + 1]
        THEN
            NoMoreSwaps ← FALSE
            Temp ← NameList[Pointer]
            NameList[Pointer] ← NameList[Pointer + 1]
            NameList[Pointer + 1] ← Temp
        ENDIF
    ENDFOR
    NumberOfItems ← NumberOfItems - 1
UNTIL NoMoreSwaps
```

2 a)



b) i)

		Name	Pointer
HeadPointer	[1]		2
0	[2]		3
	[3]		4
TailPointer	[4]		5
0	[5]		6
	[6]		7
FreePointer	[7]		8
1	[8]		9
	[9]		10
	[10]		0

ii)

```
PROCEDURE RemoveName()
    //Report error if Queue is empty
    IF HeadPointer = 0
  THEN
    Error
  ELSE
        OUTPUT Queue[HeadPointer].Name
   //current node is head of queue
   CurrentPointer ← HeadPointer
        // update head pointer
        HeadPointer ← Queue[CurrentPointer].Pointer
       //if only one element in queue, then update tail pointer
  IF HeadPointer = 0
   THEN
     TailPointer ← 0
  ENDIF
       // link released node to free list
       Queue[CurrentPointer].Pointer ← FreePointer
       FreePointer ← CurrentPointer
   ENDIF
ENDPROCEDURE
```