end if

**12/10/17**
MERGESORT(A)
MERGESORT(A first half)
MERGESORT(A 2nd half)
MERGE

INORDERTRAVERSAL(T) // T is a tree
* 
INORDERTRAVERSAL(T.left)
output T.data
INORDERTRAVERSAL(T.right)

PRETRAV(T)
* 
output T.data
PRETRAV(T.LEFT)
PRETRAV(T.RIGHT)

POSTTRAV(T)
* 
POSTTRAV(T.left)
POSTTRAV(T.right)
output(T.data)

TRIANGLES(N) // N is an integer >= 0
if N = 0
    return 0
else
    N + TRIANGLES(N-1)
end if

Trace the algorithm with an input of N=6.

<table>
<thead>
<tr>
<th>Recursion Depth</th>
<th>N</th>
<th>N = 0 return</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>F 21</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>F 15</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>F 10</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>F 6</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>F 3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>F 1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>T 0</td>
</tr>
</tbody>
</table>

output = 21

IB Exam Question 6. Consider the following recursive algorithm FUN(X, N), where X and N are two integers.

FUN(X, n)
if N<=0 then
    return X*FUN(X, N-1)
else
    return X*FUN(X, N-1)
end if

The return statement gives the value that the algorithm generates.
   a) Determine how many times multiplication is performed when this algorithm is executed. [1]
   b) Determine the value of FUN(2,3), showing all of your working [3]
   c) State the purpose of this recursive algorithm [1]

<table>
<thead>
<tr>
<th>X</th>
<th>N</th>
<th>return</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Value of FUN(2,3) = 8
   a) N
   c) It finds X^n

---

**Preview from Notesale.co.uk**

Page 2 of 12
### 5.1.18 Define the term dynamic data structure.

*Dynamic data structures* are data structures that can change size during the execution of a program. The size of the structure is determined during run time, which is a very efficient use of memory space.

### 5.1.19 Compare the use of static and dynamic data structures.

<table>
<thead>
<tr>
<th>Static data structures</th>
<th>Dynamic data structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer can allocate space during compilation</td>
<td>Only uses the space needed at any time</td>
</tr>
<tr>
<td>Easy to program</td>
<td>Makes efficient use of memory</td>
</tr>
<tr>
<td>Easy to check for overflow</td>
<td>Storage no longer required can be returned to the system for other use</td>
</tr>
<tr>
<td>An array allows random access</td>
<td></td>
</tr>
<tr>
<td>Programmer has to estimate maximum amount of space needed</td>
<td>Difficult to program</td>
</tr>
<tr>
<td>Can waste space</td>
<td>Can be slow to implement searches</td>
</tr>
<tr>
<td></td>
<td>A linked list only allows serial search</td>
</tr>
</tbody>
</table>

### 5.1.20 Suggest a suitable structure for a given situation.

- **Stacks:** The most important application of a stack is to implement function calls. This provides a technique for eliminating recursion from a program.

- **Queues:**
  - Computing applications: serving requests of a single shared resource (printer, disk, CPU).
  - Buffers: MP3 players and portable CD players, iPod playlist.
  - Playlist for jukebox: add songs to the end, play from the front of the list.
  - Handling interruptions, so the first interruption can be treated first

- **Linked list:**
  - You need constant-time insertions/deletions from the list.
  - You don’t know how many items will be in the list.
  - You don’t need random access to any elements.
  - You want to be able to insert items in the middle of the list.

- **Arrays:**
  - You need indexed/random access to elements
  - You know the number of elements in the array ahead of time
  - You can allocate the exact amount of memory for the array
  - You need speed when iterating through all the elements in sequence.

- **Binary Trees:**
  - Binary Search Tree - Used in many search applications where data is constantly entering/leaving
  - Heaps
  - GGM Trees
  - Syntax Tree