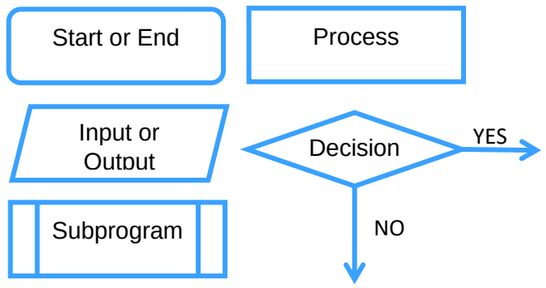
Flowcharts

- Created to represent an algorithm.
- Show the data that is input, and output.
- Show processes that take place.
- Show any decisions and repetitions that take place.
- Lines show flow through the chart.
- Shapes represent different functions



Searching Algorithms

Linear Search

1. Check the first value
2. If it is desired value
   - Stop
3. Otherwise check the second value
4. Keep Going until all elements have been checked or the value is found

Binary Search

1) Put the list in order.
2) Take the middle value.
3) Compare it to the desired value.
   a) If it is the desired value.
      i) Stop.
   b) If it is larger than the desired value.
      i) Take the list to the left of the middle value.
   c) If it is smaller than the desired value.
      i) Take the list to the right of the middle value.
4) Repeat step 3 with the new list.

Trace Tables

- Tests algorithms for logic errors which occur when the algorithm is executed.
- Simulates the steps of algorithm.
- Each stage is executed one at a time allowing inputs, outputs, variables, and processes to be checked for the correct value at each stage.

```
X = 3
Y = 1
while X > 0
      Y = Y + 1
      X = X - 1
print(Y)
```

Stage	X	Y	Output
1	3	1	
2		2	
3	2		
4		3	
5	1		
6		4	
7	0		
8			4

2.1 Algorithms

Key Concepts

- Computational thinking:
  - The use of computers to solve problems.
  - Development of algorithms to solve problems.
  - Using abstraction, decomposition, and algorithmic thinking.

- Abstraction
  - Using symbols and variables. to represent a real-world problem with a computer program.
  - Removing unnecessary elements
  - Example - a program is to be created to let users play chess against the computer.
    - Board is created as an array(s).
    - Pieces are objects that have positions on the board
    - The shape and style of the pieces may not be required.

- Decomposition
  - Breaking down large problems into a set of smaller parts.
    - Smaller problems are easier to solve
    - Each part can be solved independently
    - Each part can be tested independently
    - The parts are combined to produce the full problem.
  - There are usually several different approaches, and not one single right way to do this.

Sorting Algorithms

Bubble Sort

1) Take the first element and second element
2) Compare the two
   a) If element 1 > element 2
      i) Swap then
   b) Otherwise
      i) Do nothing
   c) Move to the next pair in the list
   d) If there are no more elements return to step (1)
   e) Otherwise, return to step (2)
3) Repeat until you have worked through the whole list without making any changes

Merge Sort

1) Split the list into individual elements.
2) Merge the elements together in pairs, putting the smallest element first.
3) Merge two pairs together, putting the smallest first.
4) Keep merging until all pairs are in order.

Insertion Sort

1) Element 1 is a sorted list.
2) The remaining elements are an ‘unsorted’ list.
   a) Compare the first element in the ‘unsorted’ list to each element in the sorted list.
   b) If it is smaller, put it in in front of that element and move the others along.
   c) If it is larger compare it with the next element.
   d) Keep going until you reach the end of the list, at this point put it in the final position.
3) Repeat the above until all elements have been sorted

Exam Reference Language

- Looks like pretend code.
- A more formal way to represent an algorithm for the exam.
- More like a programming language but does not compile.
- Is easy for programmers to read.

```
mark = input("Input mark")
if mark < 50 then
      print("Fail")
elseif mark < 70 then
      print("Pass")
elseif mark < 90 then
      print("Merit")
else
      print("Distinction")
endif
```

Completing An Algorithm

1. Read what the algorithm should do.
2. Note down the steps that should take place.
3. Read the steps of the algorithm you already have.
4. Use your notes to write code to fill in the gaps.

Pseudocode

- Uses short English words and statements to describe an algorithm.
- Generally looks a little more structured than normal English sentences.
- Flexible.
- Less precise than a programming language.

```
IF Age is equal to 14 THEN
Stand up
ELSE Age is equal to 15 THEN
Clap
ElSE Age is equal to 16 THEN
Sing a song
ELSE
Sit on the floor
END
```

Correcting An Algorithm

1. Read what the algorithm should do.
2. Note down the steps that should take place.
3. Read each step of the algorithm.
4. At each step, compare what the algorithm does to your notes about what it should do.
5. Take action to correct the algorithm where it differs from your notes.

Common Error Types

Syntax error

- The code has not been correctly typed, a “typo” in the code.
- For example entering *`print = (Hello`* Instead of *`Print = (“Hello”)`*

Logic error

- The code has been typed, there is an error in the logic used to create it.
- This might be running steps in the correct order, or multiplying instead of dividing

Key Terms

- Algorithmic thinking - identifying the steps involved in solving a problem.
- Algorithm – a series of steps to perform an action or solve a problem.
- Flowchart – a diagram showing inputs, outputs and processes within an algorithm.
- Process – an action that takes place.
- Pseudocode – simplified language used to design algorithms.
- Exam Reference Language – a more formal way of writing algorithms used within the exam.