# AQA Computer Science A-Level

## 4.3.4 Searching algorithms

Intermediate Notes

**Specification:**

### 4.3.4.1 Linear search:

Know and be able to trace and analyse the complexity of the linear search algorithm. Time complexity is O(n).

### 4.3.4.2 Binary search

Know and be able to trace and analyse the time complexity of the binary search algorithm. Time complexity is O(log n).

### 4.3.4.3 Binary tree search

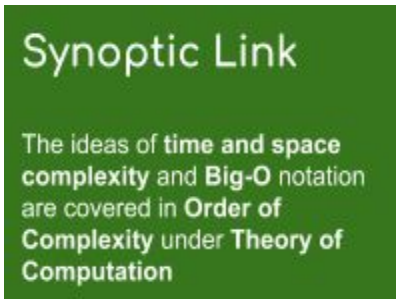
Be able to trace and analyse the time complexity of the binary tree search algorithm. Time complexity is O(log n).

# Searching Algorithms

A searching algorithm is used to find a specified data item within a set of data. This could be an array, a list or even a binary tree. There are several different searching algorithms which can be used in varying circumstances. The three studied below are linear search, binary search and a binary tree search.

## Linear Search

A linear search can be conducted on any list, even if the data isn't in order. It is very simple to program, but it has a comparatively high time complexity, so is rarely used in the real world. It has one loop, and thus has a time complexity of O(N).




Linear search works by inspecting every item in a list one by one until the desired item (the target) is found.

If the target does not exist in the list, the algorithm will check every single item in the list before finishing. If the algorithm has been properly programmed, this will not result in an error.

Linear Search Example

Here is an array of people:

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Data | Dean | Angelina | Oliver | Seamus | Cho | Fred |

Where is "Oliver" in the array?
The first position of the array is checked.

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Data | Dean | Angelina | Oliver | Seamus | Cho | Fred |

"Oliver" ≠ "Dean"
Check the next position in the array.

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Data | Dean | Angelina | Oliver | Seamus | Cho | Fred |

“Oliver” ≠ “Angelina”
So check the next position in the array

| Position | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Data | Dean | Angelina | Oliver | Seamus | Cho | Fred |

“Oliver” = “Oliver”
Hence Oliver is found at position 3 in the array.

Pseudocode

The linear search algorithm could be programmed using the following pseudocode:

```
LinearSearch(Target, ArrayofNames)
Boolean Found
Integer Count
Found ← FALSE
Count ← 0

Do Until Found == TRUE or Count == ArrayofNames Count
     If Target == ArrayofNames(Count)
          Found ← TRUE
     Else
          Count ← Count + 1
     End If
Loop

If Found = TRUE
     Output Target found at Count
Else
     Output Target not found
End if
```

# Binary Search

The binary search algorithm is more efficient than the linear search algorithm, but it can only be used on ordered lists.

A binary search works by looking at the midpoint of a list and determining if the target is higher or lower than the midpoint. The time complexity is O(logN) because the list is halved each search.

Binary Search Example

Here is an array of people:

| Position | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Data | Charles | Fredrick | George | Ginevra | Percy | Ronald | William |

Where is George?

The first step is to take the middle piece of data. To find the midpoint of the data, add the highest position and the lowest position of the array being considered, and divide by 2.

**Note**

**If you calculate a midpoint that isn't a whole number, be sure to always round up**

For example:
0 + 6 = 6, 6 / 2 = 3. Look at position 3 of the array.

| Position | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Data | Charles | Fredrick | George | Ginevra | Percy | Ronald | William |

"George" ≠ "Ginevra"
"George" < "Ginevra" because George is before Ginevra in the list.
Hence we discard all places in the array beyond and including "Ginevra"
Our new array looks like this:

| Position | 0 | 1 | 2 |
|---|---|---|---|
| Data | Charles | Fredrick | George |

Again, we check the middle position. 0 + 2 = 2, 2 / 2 = 1.

| Position | 0 | 1 | 2 |
|---|---|---|---|
| Data | Charles | Fredrick | George |

“George” ≠ “Fredrick”
“George” > “Fredrick”
Hence, everything before and including “Fredrick” does not need to be checked.

| Position | 2 |
|---|---|
| Data | George |

There is only one element in the array. 2 + 2 = 4, 4 / 2 = 2

| Position | 2 |
|---|---|
| Data | George |

“George” = “George”
George is found at position 2 of the array.

Pseudocode

A binary search can be conducted in many different ways. Here is pseudocode for one solution:

```
BinarySearch(Target, ArrayofNames)
     Integer TopPointer
     Integer BottomPointer
     Integer Midpoint
     Boolean Found

     Found ← FALSE
     BottomPointer ← 0
     TopPointer ← ArrayofNames Count - 1

     Do Until Found = TRUE or TopPointer < BottomPointer
          Midpoint = int mid TopPointer, BottomPointer
          If ArrayofNames(Midpoint) = Target
               Found = TRUE
          ElseIf ArrayofNames(Midpoint) > Target
               TopPointer = Midpoint - 1
          ElseIf ArrayofNames(Midpoint) < Target
               BottomPointer = Midpoint + 1
               End If
     Loop

     If Found = TRUE
          Output Target found at Midpoint
     Else
          Output Target not found
     End if
```

# Binary Tree Search

**Synoptic Link**

**Graphs** can be used as **visual representations** of **complex relationships**.

Graphs are covered in **Graphs** under **Fundamentals of Data Structures**.

A binary tree search is the same as a binary search, except it is conducted on a binary tree rather than a list. A tree is an type of connected graph that has no cycles.

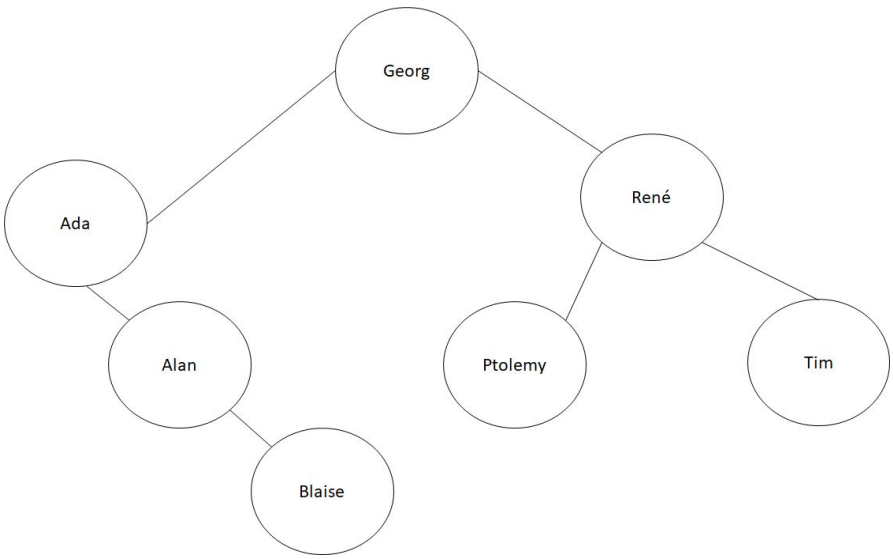
A binary tree is a rooted, ordered tree in which each node has no more than 2 children. Just like the binary search algorithm, the binary tree search algorithm has a time complexity of O(logN).

## Binary Tree Search Example

Here is a list of names:
`Georg, René, Ada, Alan, Blaise, Ptolemy, Tim.`

Does the list contain “Alan”?

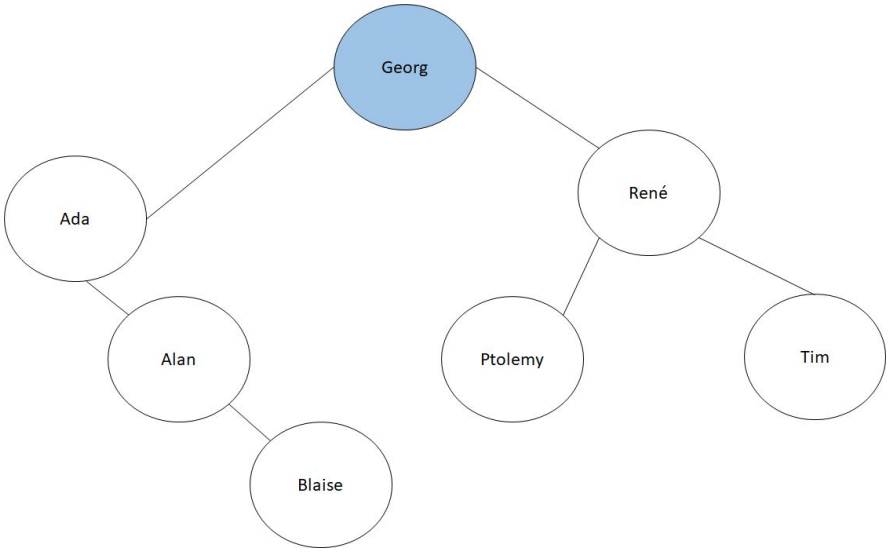
The first stage in a binary tree search is to put the list into a binary tree.



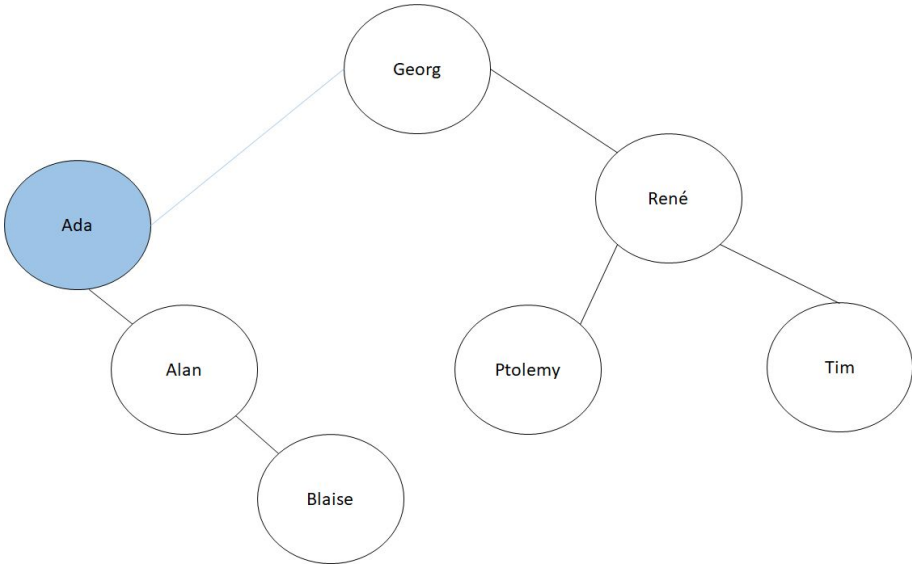

**Synoptic Link**

Information on how to create **binary trees** can be found under **Trees** in **Fundamentals of Data Structures**

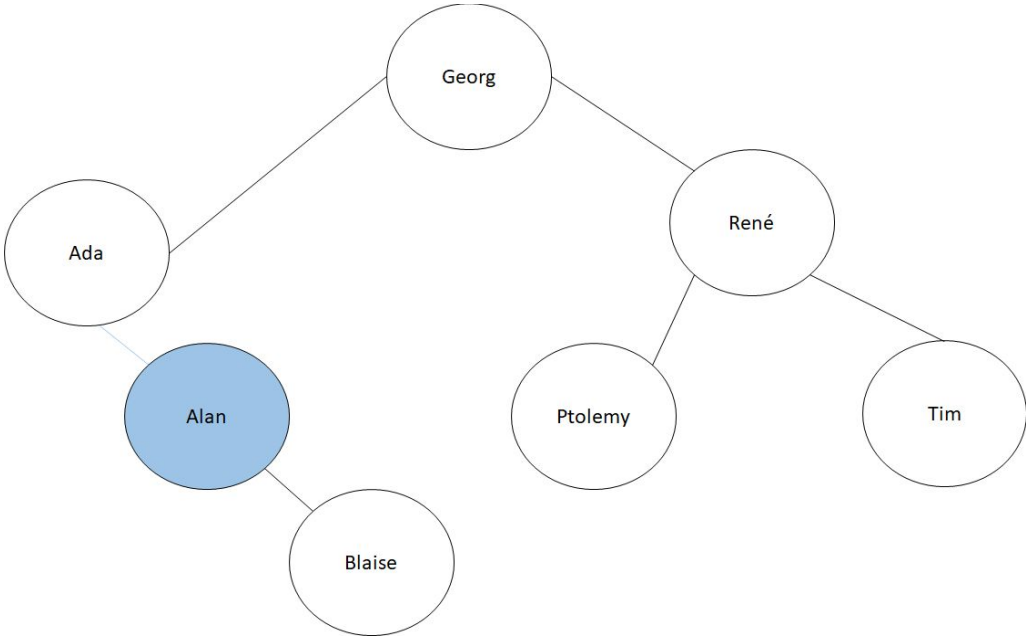
A binary tree search starts at the root.




“Alan” ≠ “Georg”
“Alan” < “Georg”
Therefore only items left of the root will be considered further.




“Alan” ≠ “Ada”
“Alan” > “Ada”
Hence only nodes right of Ada will be further considered.




“Alan” = “Alan”
So Alan is in the tree.