AQA Computer Science A-Level

4.3.4 Searching algorithms

Advanced 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

An algorithm is a set of instructions which completes a task in a finite time and always terminates. In the case of a searching algorithm, the task is to find the location of a certain item in a list or to verify if the item is in the list. 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. Hash tables are not searching algorithms, but function in a similar way.

> **Synoptic Link**
>
> A **hash table** is a data structure that creates a mapping between keys and values.
>
> They are covered further in **Hash Tables** under **Fundamentals of Data Structures**.

Linear Search

> **Synoptic Link**
>
> The ideas of time complexity, space complexity and Big-O notation are covered in **Order of Complexity** under **Theory of Computation.**

A linear search can be conducted on any unordered list. It is the most 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). In this algorithm, each item in the list is compared sequentially to the target.

Linear Search Example 1

Here is an array of people:

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

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

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

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

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

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

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

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

Linear Search Example 2

Where is “Hannah” in the array?
The first position of the array is checked.

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

“Hannah” ≠ “Dean”
Check the next position in the array.

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

"Hannah" ≠ "Angelina"
So check the next position in the array

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

"Hannah" ≠ "Seamus"
So check the next position in the array

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

"Hannah" ≠ "Oliver"
So check the next position in the array

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

"Hannah" ≠ "Cho"
So check the next position in the array

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

"Hannah" ≠ "Fred"

Check the next position in the array. There are no more positions in the array, so Hannah is not contained in the array. When correctly programmed, a linear search algorithm should not result in an error when trying to locate an item not in the array.

Pseudocode for a linear search algorithm could be:

```
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
```

Using a For...Next loop in lieu of a Do Until loop would be bad programming practice. The For….Next loop is an example of definite iteration, whereas the Do Until loop is an example of indefinite iteration. For instance, if the target was at the beginning of the array, the Do Until loop would locate the item immediately and then exit the loop, whereas a For...Next loop would still have to search through each piece of data. This makes the Do...Until loop quicker in this scenario (although both loops are O(N) as big O notation looks at the worst case scenario).

Synoptic Link

Iteration is the process of repeating a block of code multiple times.

Iteration is covered under **Programming Concepts** in **Fundamentals of Programming**.

Binary Search

A binary search can be used on any ordered list. If the list is unordered, the data must be sorted by a sorting algorithm. 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.

Synoptic Link

A **merge sort** or a **bubble sort** can be used to order a list.

Further detail on sorting algorithms can be found under **Sorting Algorithms** in **Fundamentals of Algorithms.**

Binary Search Example 1

Here is an array of people:

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

Where is George?

This is an unordered list, so the first step is to use a sorting algorithm. The data can be sorted into ascending or descending order, although each will require a slightly different code.

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

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. I.e. 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 when in alphabetical order. Your programming language can compare strings to determine whether they are higher or lower than one another.

Hence, discard all places in the array beyond “Ginevra”.

Our new array looks like this:

Position	0	1	2
Data	Charles	Fredrick	George

To find George, we must 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 “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.

Binary Search Example 2

Here is an array of names:

Position	0	1	2	3	4	5
Data	Mushu	Zazu	Flounder	Pascal	Gus	Baloo

Where is “Pegasus”?

The first step is to order then with a sorting algorithm.

Position	0	1	2	3	4	5
Data	Baloo	Flounder	Gus	Mushu	Pascal	Zazu

The first step is to find the midpoint. 0 + 5 = 5, 5/2 = 2.5, there is no position 2.5 in the array, so an int calculation is performed on it - this removes the decimal part. Hence, we need to check the data in position 2.

Position	0	1	2	3	4	5
Data	Baloo	Flounder	Gus	Mushu	Pascal	Zazu

“Pegasus” ≠ “Gus”
“Pegasus” > “Gus”, so only positions 3, 4 and 5 will be considered from now on.

Position	3	4	5
Data	Mushu	Pascal	Zazu

To find the midpoint, 3 + 5 = 8, 8/2 = 4.

Position	3	4	5
Data	Mushu	Pascal	Zazu

“Pegasus” ≠ “Pascal”

“Pegasus” > “Pascal”
Positions 3 and 4 are disregarded.

Position	5
Data	Zazu

There is only one piece of data in the array. 5 + 5 = 10, 10/2 = 5

Position	5
Data	Zazu

“Pegasus” ≠ “Zazu”
“Pegasus” > “Zazu”
There is no more data to check; Pegasus isn’t in the array.

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
```

A binary search can also be completed through recursion.

Synoptic Link

Recursion refers to a block of code calling itself in order to complete a task.

Recursion is covered in **recursive techniques** in **fundamentals of programming**.

Binary Tree Search

A binary tree search is the same as a binary search, except it is conducted on a binary tree. A tree is an acyclic, connected graph, and a binary tree is a rooted ordered tree in which each node has 0, 1 or 2 children. Like a binary search, a binary tree search has a time complexity of O(logN).

> **Synoptic Link**
>
> **Graphs** can be used as **visual representations** of **complex relationships**.
>
> Graphs are covered in **Graphs** under **Fundamentals of Data Structures**.

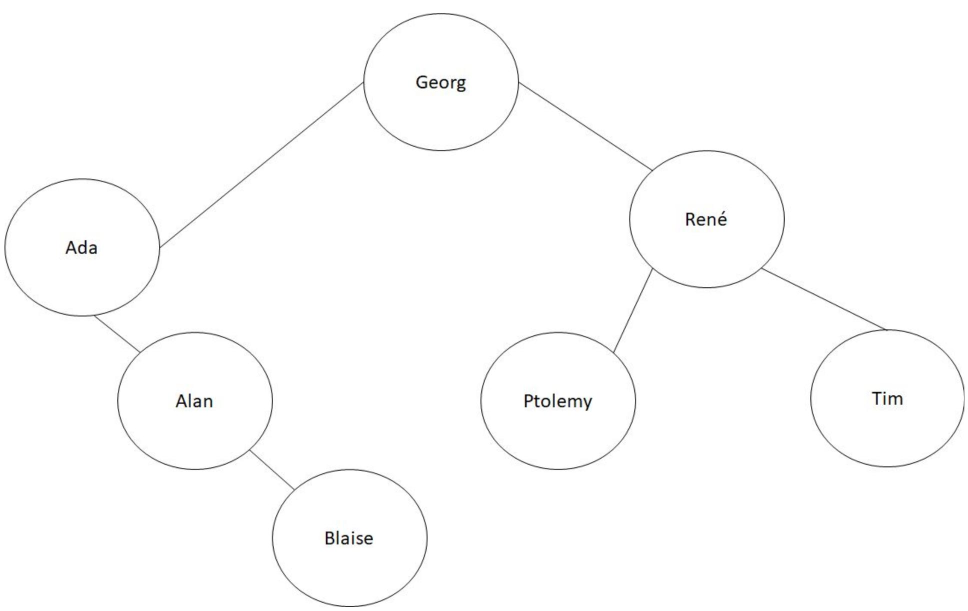
Binary Tree Search Example

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

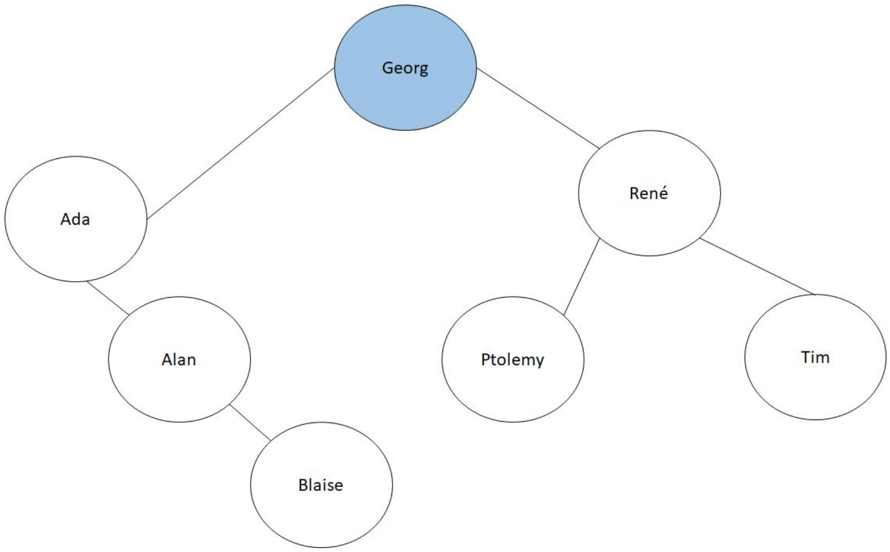
Does the list contain "Alan"?

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.**



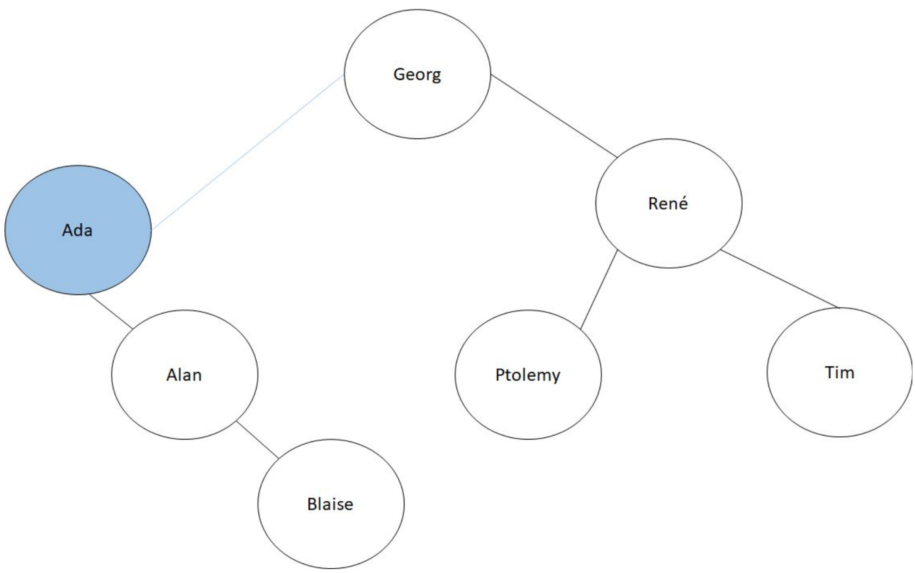
A binary tree search starts at the root.



“Alan” ≠ “Georg”

“Alan” < “Georg”

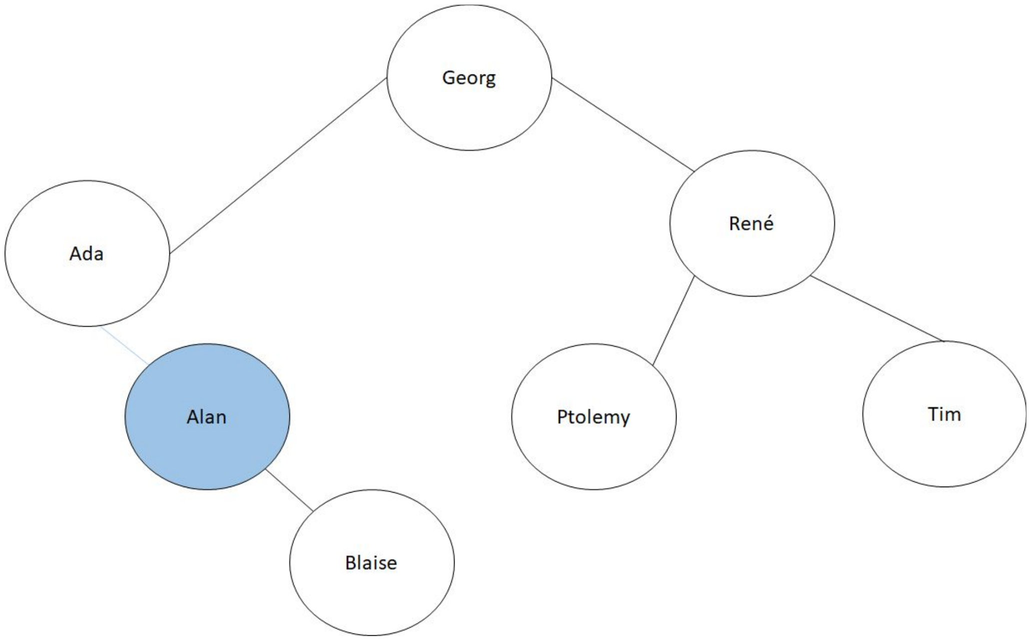
Therefore only items left of the root will be considered further.



“Alan” ≠ “Ada”

“Alan” > “Ada”

Hence only nodes right of Ada will be further considered.



"Alan" = "Alan"

Alan is in the tree.