Linear and Binary Search Algorithms

Question No. 1

(a) *Mark as follows:*

```
High ← 63                                    [1]
X = 0                                        [1]
High ← Middle - 1                            [1]
```

One mark for each correct line

(b) **(i)** ordered / in order [1]

(ii) 6 [1]

(iii) 0 [1]
item not present in array [1]
non zero [1]
position of the item in the array [1]

Question No. 2

4(a)(i)	1 mark for error and correction Error 1 – `IF List[LowerBound] = SearchValue` Correction – `IF List[MidPoint] = SearchValue` Error 2 – `UpperBound ← MidPoint + 1` Correction – `LowerBound ← MidPoint + 1` Error 3 – `IF LowerBound > MidPoint` Correction - `IF LowerBound > UpperBound` Error 4 – `IF ValueFound = FALSE` Correction – `IF ValueFound = TRUE`	**4**
4(a)(ii)	Linear search	**1**

Question No. 3

3	1 mark for each completed statement FUNCTION BinarySearch(ThisArray, LowerBound, UpperBound, SearchItem: INTEGER) RETURNS INTEGER DECLARE Flag : BOOLEAN DECLARE Mid : INTEGER Flag ← -2 WHILE Flag <> -1 Mid ← LowerBound + ((UpperBound - LowerBound) DIV 2) IF **UpperBound** < **LowerBound** THEN RETURN **-1** ELSE IF ThisArray[Mid] > SearchItem THEN UpperBound ← Mid **- 1** ELSE IF ThisArray[Mid] < SearchItem THEN LowerBound ← Mid **+ 1** ELSE RETURN **Mid** ENDIF ENDIF ENDIF ENDWHILE ENDFUNCTION	**6**

Question No. 4

6(a)	One mark each to max 6:
	
	• Function declared taking the string to search for as a parameter, string return (and array)
	• (Declare and) initialise a variable (0) to count number times code occurs
	• Loop through 20 000 elements … // looping until unused element
	• … IF Left 7 from GeoCodeLog[loop counter] = parameter …
	• … if true, increment number times occurs
	• … if true, RIGHT 10 from GeoCodeLog[loop counter] …
	• … if true compare to last date and replace if after // store in variable for last date
	• Return concatenated number times & " " & last date …
	• … converting number to string
	Example 1:
	`FUNCTION SearchLog(SearchGeoCode : STRING) RETURNS STRING`
	`  DECLARE AccessCount : INTEGER`
	`  DECLARE LatestDate : STRING`
	`  DECLARE DateAccess : DATE`
	
	`  LatestDate ← "01/01/1500"`
	`  FOR Index ← 0 TO 19999`
	`  IF LEFT(GeoCodeLog[Index], 7)`
	`  THEN`
	`  AccessCount ← AccessCount + 1`
	`  DateAccess ← (RIGHT(GeoCodeLog[Index],`
	`  10)).TODATE`
	`  IF DateAccess > LatestDate`
	`  THEN`
	`  LatestDate ← DateAccess`
	`  ENDIF`
	`  ENDIF`
	`  ENDFOR`
	`  RETURN NUM_TO_STRING(AccessCount) & " " &`
	`  DATE_TO_STRING(LatestDate)`
	
	`ENDFUNCTION`
		6

Question No. 5

8(a)	`FUNCTION IgnoreWord (ThisWord : STRING) RETURNS BOOLEAN`
	
	`  DECLARE Found : BOOLEAN`
	`  DECLARE Index : INTEGER`
	
	`  Found ← False`
	`  Index ← 1`
	`  ThisWord ← TO_LOWER(ThisWord)`
	
	`  REPEAT`
	`  IF TO_LOWER(IgnoreList[Index]) = ThisWord THEN`
	`  Found ← TRUE`
	`  ENDIF`
	`  Index ← Index + 1`
	`  UNTIL Found = TRUE OR Index > 10`
	
	`  RETURN Found`
	
	`ENDFUNCTION`
	
	1 mark for each of the following:
	
	1 Loop through array elements
	2 Convert both strings to same case
	3 Compare array element with parameter **in a loop**
	4 Set a flag (or similar) if match found (after reasonable attempt at MP3) **in a loop**
	5 Return `TRUE` or `FALSE` in all cases
	
	Note: Max 4 if function declaration incorrect
		5

Question No. 6

3(a)	``` TotalValue ← 0 ZeroCount ← 0 FOR Index ← 1 TO 100 TotalValue ← TotalValue + Result[Index] IF Result[Index] = 0.0 THEN ZeroCount ← ZeroCount + 1 ENDIF ENDFOR OUTPUT "The average is ", (TotalValue / 100) OUTPUT "The number of elements with a zero value is ", ZeroCount ``` One mark for each of the following: 1 Both initialisations 2 Loop 100 times 3 Adding individual element to TotalValue **in a loop** 4 Check if element value is zero **in a loop** 5 If so increment ZeroCount **in a loop** 6 Average is calculated after the loop 7 Both OUTPUT statements, including message and variables	**7**

Question No. 7

3	1 mark for each completed statement
	
	`FUNCTION BinarySearch(ThisArray, LowerBound, UpperBound, SearchItem: INTEGER) RETURNS INTEGER`
	`  DECLARE Flag : BOOLEAN`
	`  DECLARE Mid : INTEGER`
	`  Flag ← -2`
	`  WHILE Flag <> -1`
	`  Mid ← LowerBound + ((UpperBound - LowerBound) DIV 2)`
	`  IF UpperBound < LowerBound`
	`  THEN`
	`  RETURN -1`
	`  ELSE`
	`  IF ThisArray[Mid] > SearchItem`
	`  THEN`
	`  UpperBound ← Mid - 1`
	`  ELSE`
	`  IF ThisArray[Mid] < SearchItem`
	`  THEN`
	`  LowerBound ← Mid + 1`
	`  ELSE`
	`  RETURN Mid`
	`  ENDIF`
	`  ENDIF`
	`  ENDIF`
	`  ENDWHILE`
	`ENDFUNCTION`
		6

Question No. 8

8(a)	`INTEGER`
	`9` // `LENGTH(MyList) - 1`
	`Index + 1`
	`"Value not found"` (or any similar phrase)
		4

8(b)(i)	The list to be searched must be ordered/sorted	**1**
8(b)(ii)	Any **four** from MP1 Find the middle item / index MP2 Check the value of middle item in the list to be searched MP3 If equal item searched for is found MP4 If this is not equal/greater/less than the item searched for MP5 … discard the half of the list that does not contain the search item MP6 Repeat the above steps until the item searched for is found MP7 … or there is only one item left in the list and it is not the item searched for // lower bound > / = upper bound	**4**