



3.4 Searching algorithms Mark Scheme

Mark schemes

Q1.

- (a) (i) **Mark is for AO3 (programming)**

Selection structure with correct condition(s) (9, 23) added in suitable place and value of 4 assigned to two tiles in the dictionary;

R. if any other tile values changed

1

Python 2

```
def CreateTileDictionary():
    TileDictionary = dict()
    for Count in range(26):
        if Count in [0, 4, 8, 13, 14, 17, 18, 19]:
            TileDictionary[chr(65 + Count)] = 1
        elif Count in [1, 2, 3, 6, 11, 12, 15, 20]:
            TileDictionary[chr(65 + Count)] = 2
        elif Count in [5, 7, 10, 21, 22, 24]:
            TileDictionary[chr(65 + Count)] = 3
        elif Count in [9, 23]:
            TileDictionary[chr(65 + Count)] = 4
        else:
            TileDictionary[chr(65 + Count)] = 5
    return TileDictionary
```

Python 3

```
def CreateTileDictionary():
    TileDictionary = dict()
    for Count in range(26):
        if Count in [0, 4, 8, 13, 14, 17, 18, 19]:
            TileDictionary[chr(65 + Count)] = 1
        elif Count in [1, 2, 3, 6, 11, 12, 15, 20]:
            TileDictionary[chr(65 + Count)] = 2
        elif Count in [5, 7, 10, 21, 22, 24]:
            TileDictionary[chr(65 + Count)] = 3
        elif Count in [9, 23]:
            TileDictionary[chr(65 + Count)] = 4
        else:
            TileDictionary[chr(65 + Count)] = 5
    return TileDictionary
```

Visual Basic

```
Function CreateTileDictionary() As Dictionary(Of Char, Integer)
    Dim TileDictionary As New Dictionary(Of Char, Integer) ()
    For Count = 0 To 25
        If Array.IndexOf({0, 4, 8, 13, 14, 17, 18, 19}, Count)
        > -1 Then
            TileDictionary.Add(Chr(65 + Count), 1)
        ElseIf Array.IndexOf({1, 2, 3, 6, 11, 12, 15, 20}, Count)
        > -1 Then
            TileDictionary.Add(Chr(65 + Count), 2)
        ElseIf Array.IndexOf({5, 7, 10, 21, 22, 24}, Count) > -1 Then
            TileDictionary.Add(Chr(65 + Count), 3)
        ElseIf Array.IndexOf({9, 23}, Count) > -1 Then
            TileDictionary.Add(Chr(65 + Count), 4)
        Else
```

```

        TileDictionary.Add(Chr(65 + Count), 5)
    End If
Next
Return TileDictionary
End Function

```

C#

```

private static void CreateTileDictionary(ref Dictionary<char,
int> TileDictionary)
{
    int[] Value1 = { 0, 4, 8, 13, 14, 17, 18, 19 };
    int[] Value2 = { 1, 2, 3, 6, 11, 12, 15, 20 };
    int[] Value3 = { 5, 7, 10, 21, 22, 24 };
    int[] Value4 = { 9, 23 };

    for (int Count = 0; Count < 26; Count++)
    {
        if (Value1.Contains(Count))
        {
            TileDictionary.Add((char)(65 + Count), 1);
        }
        else if (Value2.Contains(Count))
        {
            TileDictionary.Add((char)(65 + Count), 2);
        }
        else if (Value3.Contains(Count))
        {
            TileDictionary.Add((char)(65 + Count), 3);
        }
        else if (Value4.Contains(Count))
        {
            TileDictionary.Add((char)(65 + Count), 4);
        }
        else
        {
            TileDictionary.Add((char)(65 + Count), 5);
        }
    }
}

```

Java

```

Map createTileDictionary()
{
    Map<Character,Integer> tileDictionary = new
HashMap<Character,Integer>();
    for (int count = 0; count < 26; count++)
    {
        switch (count) {
            case 0:
            case 4:
            case 8:
            case 13:
            case 14:
            case 17:
            case 18:
            case 19:
                tileDictionary.put((char)(65 + count), 1);
                break;
            case 1:
            case 2:
            case 3:
            case 6:
            case 11:

```

```

        case 12:
        case 15:
        case 20:
            tileDictionary.put((char)(65 + count), 2);
            break;
        case 5:
        case 7:
        case 10:
        case 21:
        case 22:
        case 24:
            tileDictionary.put((char)(65 + count), 3);
            break;
        case 9:
        case 23:
            tileDictionary.put((char)(65 + count), 4);
            break;
        default:
            tileDictionary.put((char)(65 + count), 5);
            break;
    }
}
return tileDictionary;
}

```

Pascal / Delphi

```

function CreateTileDictionary() : TTileDictionary;
var
    TileDictionary : TTileDictionary;
    Count : integer;
begin
    TileDictionary := TTileDictionary.Create();
    for Count := 0 to 25 do
        begin
            case count of
                0, 4, 8, 13, 14, 17, 18, 19:
                    TileDictionary.Add(chr(65 + count), 1);
                1, 2, 3, 6, 11, 12, 15, 20: TileDictionary.Add(chr(65 +
+ count), 2);
                5, 7, 10, 21, 22, 24: TileDictionary.Add(chr(65 +
count), 3);
                9, 23: TileDictionary.Add(chr(65 + count), 4);
                else TileDictionary.Add(chr(65 + count), 5);
            end;
        end;
    CreateTileDictionary := TileDictionary;
end;

```

(ii) Mark is for AO3 (evaluate)

**** SCREEN CAPTURE ****

Must match code from part (a)(i), including prompts on screen capture matching those in code.

Code for part (a)(i) must be sensible.

Screen captures showing the requested test being performed and the correct points values for J, X, Z and Q are shown; I. order of letters

TILE VALUES

Points for X: 4

Points for R: 1

Points for Q: 5
 Points for Z: 5
 Points for M: 2
 Points for K: 3
 Points for A: 1
 Points for Y: 3
 Points for L: 2
 Points for I: 1
 Points for F: 3
 Points for H: 3
 Points for D: 2
 Points for U: 2
 Points for N: 1
 Points for V: 3
 Points for T: 1
 Points for E: 1
 Points for W: 3
 Points for C: 2
 Points for G: 2
 Points for P: 2
 Points for J: 4
 Points for O: 1
 Points for B: 2
 Points for S: 1

Either:

enter the word you would like to play OR
 press 1 to display the letter values OR
 press 4 to view the tile queue OR
 press 7 to view your tiles again OR
 press 0 to fill hand and stop the game.

1

(b) (i) **All marks for AO3 (programming)**

Iterative structure with one correct condition added in suitable place;

Iterative structure with second correct condition and logical connective;

Suitable prompt displayed inside iterative structure or in appropriate place before iterative structure; **A.** any suitable prompt

StartHandSize assigned user-entered value inside iterative structure;

Max 3 if code contains errors

4

Python 2

```

...
StartHandSize = int(raw_input("Enter start hand size: "))
while StartHandSize < 1 or StartHandSize > 20:
    StartHandSize = int(raw_input("Enter start hand size: "))
...
  
```

Python 3

```

...
StartHandSize = int(input("Enter start hand size: "))
while StartHandSize < 1 or StartHandSize > 20:
    StartHandSize = int(input("Enter start hand size: "))
...
  
```

Visual Basic

```

...
Do
    Console.Write("Enter start hand size: ")
    StartHandSize = Console.ReadLine()
Loop Until StartHandSize >= 1 And StartHandSize <= 20
...

```

C#

```

...
do
{
    Console.Write("Enter start hand size: ");
    StartHandSize = Convert.ToInt32(Console.ReadLine());
} while (StartHandSize < 1 || StartHandSize > 20);
...

```

Java

```

...
do {
    Console.println("&Enter start hand size: &");
    startHandSize = Integer.parseInt(Console.readLine());
} while (startHandSize < 1 || startHandSize > 20);
...

```

Pascal / Delphi

```

...
StartHandSize := 0;
Choice := '';
while (StartHandSize < 1) or (StartHandSize > 20) do
begin
    write('Enter start hand size: ');
    readln(StartHandSize);
end;
...

```

(ii) Mark is for AO3 (evaluate)

**** SCREEN CAPTURE ****

*Must match code from part (b)(i), including prompts on screen capture matching those in code.
Code for part (b)(i) must be sensible.*

Screen capture(s) showing that after the values 0 and 21 are entered the user is asked to enter the start hand size again and then the menu is displayed;

```

+++++
+ Welcome to the WORDS WITH AQA game +
+++++

```

```

Enter start hand size: 0
Enter start hand size: 21
Enter start hand size: 5

```

```

=====
MAIN MENU
=====

```

1. Play game with random start hand
2. Play game with training start hand

9. Quit

Enter your choice: 1

Player One it is your turn.

1

(c) (i) **All marks for AO3 (programming)**

1. Create variables to store the current start, mid and end points; **A.** no variable for midpoint if midpoint is calculated each time it is needed in the code
 2. Setting correct initial values for start and end variables;
 3. Iterative structure with one correct condition (either word is valid or start is greater than end); **R.** if code is a linear search
 4. Iterative structure with 2nd correct condition and correct logic;
 5. Inside iterative structure, correctly calculate midpoint between start and end;
A. mid-point being either the position before or the position after the exact middle if calculated midpoint is not a whole number **R.** if midpoint is sometimes the position before and sometimes the position after the exact middle **R.** if not calculated under all circumstances when it should be
 6. Inside iterative structure there is a selection structure that compares word at midpoint position in list with word being searched for;
 7. Values of start and end changed correctly under correct circumstances;
 8. True is returned if match with midpoint word found and True is not returned under any other circumstances;
- I. missing statement to display current word

Max 7 if code contains errors

Alternative answer using recursion

1. Create variable to store the current midpoint, start and end points passed as parameters to subroutine; **A.** no variable for midpoint if midpoint is calculated each time it is needed in the code **A.** midpoint as parameter instead of as local variable
2. Initial subroutine call has values of 0 for startpoint parameter and number of words in AllowedWords for endpoint parameter;
3. Selection structure which contains recursive call if word being searched for is after word at midpoint;
4. Selection structure which contains recursive call if word being searched for is before word at midpoint;
5. Correctly calculate midpoint between start and end;
A. midpoint being either the position before or the position after the exact middle if calculated midpoint is not a whole number **R.** if midpoint is sometimes the position before and sometimes the position after the exact middle **R.** if not calculated under all circumstances when it should be
6. There is a selection structure that compares word at midpoint position in list with word being searched for and there is no recursive call if they are equal with a value of True being returned;
7. In recursive calls the parameters for start and end points have correct values;

8. There is a selection structure that results in no recursive call and False being returned if it is now known that the word being searched for is not in the list;

Note for examiners: mark points 1, 2, 7 could be replaced by recursive calls that appropriately half the number of items in the list of words passed as a parameter – this would mean no need for start and end points. In this case award one mark for each of the two recursive calls if they contain the correctly reduced lists and one mark for the correct use of the length function to find the number of items in the list. These marks should not be awarded if the list is passed by reference resulting in the original list of words being modified.

I. missing statement to display current word

Max 7 if code contains errors

Note for examiners: refer unusual solutions to team leader

8

Python 2

```
def CheckWordIsValid(Word, AllowedWords):
    ValidWord = False
    Start = 0
    End = len(AllowedWords) - 1
    while not ValidWord and Start <= End:
        Mid = (Start + End) // 2
        print AllowedWords[Mid]
        if AllowedWords[Mid] == Word:
            ValidWord = True
        elif Word > AllowedWords[Mid]:
            Start = Mid + 1
        else:
            End = Mid - 1
    return ValidWord
```

Python 3

```
def CheckWordIsValid(Word, AllowedWords):
    ValidWord = False
    Start = 0
    End = len(AllowedWords) - 1
    while not ValidWord and Start <= End:
        Mid = (Start + End) // 2
        print(AllowedWords[Mid])
        if AllowedWords[Mid] == Word:
            ValidWord = True
        elif Word > AllowedWords[Mid]:
            Start = Mid + 1
        else:
            End = Mid - 1
    return ValidWord
```

Visual Basic

```
Function CheckWordIsValid(ByVal Word As String, ByRef
AllowedWords As List(Of String)) As Boolean
    Dim ValidWord As Boolean = False
    Dim LStart As Integer = 0
    Dim LMid As Integer
    Dim LEnd As Integer = Len(AllowedWords) - 1
    While Not ValidWord And LStart <= LEnd
        LMid = (LStart + LEnd) \ 2
```



```

    Console.WriteLine(AllowedWords(LMid))
    If AllowedWords(LMid) = Word Then
        ValidWord = True
    ElseIf Word > AllowedWords(LMid) Then
        LStart = LMid + 1
    Else
        LEnd = LMid - 1
    End If
End While
Return ValidWord
End Function

```

C#

```

private static bool CheckWordIsValid(string Word,
List<string> AllowedWords)
{
    bool ValidWord = false;
    int Start = 0;
    int End = AllowedWords.Count - 1;
    int Mid = 0;
    while (!ValidWord && Start <= End)
    {
        Mid = (Start + End) / 2;
        Console.WriteLine(AllowedWords[Mid]);
        if (AllowedWords[Mid] == Word)
        {
            ValidWord = true;
        }
        else if (string.Compare(Word, AllowedWords[Mid]) > 0)
        {
            Start = Mid + 1;
        }
        else
        {
            End = Mid - 1;
        }
    }
    return ValidWord;
}

```

Java

```

boolean checkWordIsValid(String word, String[] allowedWords)
{
    boolean validWord = false;
    int start = 0;
    int end = allowedWords.length - 1;
    int mid = 0;
    while (!validWord && start <= end)
    {
        mid = (start + end) / 2;
        Console.println(allowedWords[mid]);
        if (allowedWords[mid].equals(word))
        {
            validWord = true;
        }
        else if (word.compareTo(allowedWords[mid]) > 0)
        {
            start = mid + 1;
        }
        else
        {
            end = mid - 1;
        }
    }
}

```

```

    }
    return validWord;
}

```

Pascal / Delphi

```

function CheckWordIsValid(Word : string; AllowedWords : array
of string) : boolean;
var
    ValidWord : boolean;
    Start, Mid, EndValue : integer;
begin
    ValidWord := False;
    Start := 0;
    EndValue := length(AllowedWords) - 1;
    while (not(ValidWord)) and (Start <= EndValue) do
        begin
            Mid := (Start + EndValue) div 2;
            writeln(AllowedWords[Mid]);
            if AllowedWords[Mid] = Word then
                ValidWord := True
            else if Word > AllowedWords[Mid] then
                Start := Mid + 1
            else
                EndValue := Mid - 1;
        end;
    CheckWordIsValid := ValidWord;
end;

```

(ii) Mark is for AO3 (evaluate)

**** SCREEN CAPTURE ****

*Must match code from part (c)(i), including prompts on screen capture matching those in code.
Code for part (c)(i) must be sensible.*

R. if comparison words not shown in screen capture r

Screen capture(s) showing that the word "jars" was entered and the words "MALEFICIAL", "DONGLES", "HAEMAGOGUE", "INTERMINGLE", "LAGGER", "JOULED", "ISOCLINAL", "JAU KING", "JACARANDA", "JAMBEUX", "JAPONICA", "JAROVIZE", "JASPER", "JARTA", "JARRAH", "JARRINGLY", "JARS" are displayed in that order;

A. "MALEFICIAL", "DONGOLA", "HAEMAGOGUES", "INTERMINGLED", "LAGGERS", "JOULING", "ISOCLINE", "JAUNCE", "JACARE", "JAMBING", "JAPPING", "JAROVIZING", "JASPERISES", "JARVEY", "JARRINGLY", "JARTA", "JARS" being displayed if alternative answer for mark point 5 in part (c)(i) used

ALTERNATIVE ANSWERS (for different versions of text file)

Screen capture(s) showing that the word "jars" was entered and the words "MALEATE", "DONDER", "HADST", "INTERMENDIS", "LAGAN", "JOTTERS", "ISOCHROMATIC", "JASPERS", "JABBING", "JALOUSIE", "JAPANISES", "JARGOONS", "JARRED", "JASIES", "JARUL", "JARS" are displayed in that order;

A. "MALEATE", "DONDERED", "HAE", "INTERMEDIUM", "LAGANS", "JOTTING", "ISOCHROMOSONES", "JASPERWARES", "JABBLED", "JALOUSING", "JAPANIZED", "JARINA", "JARRINGS", "JASMINES",

“JARVEYS”, “JARTAS”, “JARSFUL”, “JARS” being displayed if alternative answer for mark point 5 in part (c)(i) used

Screen capture(s) showing that the word “jars” was entered and the words “LAMP”, “DESK”, “GAGE”, “IDEAS”, “INVITATION”, “JOURNALS”, “JAMAICA”, “JEWELLERY”, “JEAN”, “JAR”, “JAY”, “JASON”, “JARS” are displayed in that order;

A. “LAMP”, “DESK”, “GAGE”, “IDEAS”, “INVITATIONS”, “JOURNEY”, “JAMIE”, “JEWISH”, “JEEP”, “JAVA”, “JAPAN”, “JARS” being displayed if alternative answer for mark point 5 in part (c)(i) used

Either:

```
enter the word you would like to play OR
press 1 to display the letter values OR
press 4 to view the tile queue OR
press 7 to view your tiles again OR
press 0 to fill hand and stop the game.
```

>jars

```
MALEFICIAL
DONGLES
HAEMAGOGUE
INTERMINGLE
LAGGER
JOULED
ISOCLINAL
JAUING
JACARANDA
JAMBEUX
JAPONICA
JAROVIZE
JASPER
JARTA
JARRAH
JARRINGLY
JARS
```



Valid word **EXAM PAPERS PRACTICE**

Do you want to:

```
replace the tiles you used (1) OR
get three extra tiles (2) OR
replace the tiles you used and get three extra tiles (3) OR
get no new tiles (4)?
```

>

1

(d) (i) **All marks for AO3 (programming)**

1. Creating new subroutine called `CalculateFrequencies` with appropriate interface; **R.** if spelt incorrectly **I.** case
2. Iterative structure that repeats 26 times (once for each letter in the alphabet);
3. Iterative structure that looks at each word in `AllowedWords`;
4. Iterative structure that looks at each letter in a word and suitable nesting for iterative structures;
5. Selection structure, inside iterative structure, that compares two letters;
A. use of built-in functions that result in same functionality as mark points 4 and 5;;

6. Inside iterative structure increases variable used to count instances of a letter;
7. Displays a numeric count (even if incorrect) and the letter for each letter in the alphabet; **A.** is done in sensible place in `DisplayTileValues`
8. Syntactically correct call to new subroutine from `DisplayTileValues`; **A.** any suitable place for subroutine call

Alternative answer

If answer looks at each letter in `AllowedWords` in turn and maintains a count (eg in array/list) for the number of each letter found then mark points 2 and 5 should be:

2. Creation of suitable data structure to store 26 counts.
5. Appropriate method to select count that corresponds to current letter.

Max 7 if code contains errors

8

Python 2

```
def CalculateFrequencies(AllowedWords):
    print "Letter frequencies in the allowed words are:"
    for Code in range (26):
        LetterCount = 0
        LetterToFind = chr(Code + 65)
        for Word in AllowedWords:
            for Letter in Word:
                if Letter == LetterToFind:
                    LetterCount += 1
            sys.stdout.write(LetterToFind + " " + LetterCount)

def DisplayTileValues(TileDictionary, AllowedWords):
    print()
    print("TILE VALUES")
    print()
    for Letter, Points in TileDictionary.items():
        sys.stdout.write("Points for " + Letter + ": " +
            str(Points) + "\n")
    print()
    CalculateFrequencies(AllowedWords)
```

Alternative answer

```
def CalculateFrequencies(AllowedWords):
    for Letter in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
        Count=0
        for Word in AllowedWords:
            NumberOfTimes = Word.count(Letter)
            Count = Count + NumberOfTimes
        sys.stdout.write(Letter + " " + str(Count))
```

Alternative answer

```
def CalculateFrequencies(AllowedWords):
    Counts = []
    for a in range(26):
        Counts.append(0)
    for Word in AllowedWords:
        for Letter in Word:
            Counts[ord(Letter) - 65] += 1
    for a in range(26):
        sys.stdout.write(chr(a + 65) + " " + str(Counts[a]))
```

Python 3

```
def CalculateFrequencies(AllowedWords):
    print("Letter frequencies in the allowed words are:")
    for Code in range(26):
        LetterCount = 0
        LetterToFind = chr(Code + 65)
        for Word in AllowedWords:
            for Letter in Word:
                if Letter == LetterToFind:
                    LetterCount += 1
        print(LetterToFind, " ", LetterCount)

def DisplayTileValues(TileDictionary, AllowedWords):
    print()
    print("TILE VALUES")
    print()
    for Letter, Points in TileDictionary.items():
        print("Points for " + Letter + ": " + str(Points))
    print()
    CalculateFrequencies(AllowedWords)
```

Alternative answer

```
def CalculateFrequencies(AllowedWords):
    for Letter in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
        Count=0
        for Word in AllowedWords:
            NumberOfTimes = Word.count(Letter)
            Count = Count + NumberOfTimes
        print(Letter,Count)
```

Alternative answer

```
def CalculateFrequencies(AllowedWords):
    Counts = []
    for a in range(26):
        Counts.append(0)
    for Word in AllowedWords:
        for Letter in Word:
            Counts[ord(Letter) - 65] += 1
    for a in range(26):
        print(chr(a + 65), Counts[a])
```

Visual Basic

```
Sub CalculateFrequencies(ByRef AllowedWords As List(Of
String))
    Dim LetterCount As Integer
    Dim LetterToFind As Char
    Console.WriteLine("Letter frequencies in the allowed words
are:")
    For Code = 0 To 25
        LetterCount = 0
        LetterToFind = Chr(Code + 65)
        For Each Word In AllowedWords
            For Each Letter In Word
                If Letter = LetterToFind Then
                    LetterCount += 1
                End If
            Next
        Next
        Console.WriteLine(LetterToFind & " " & LetterCount)
    Next
End Sub
```

```
Sub DisplayTileValues(ByVal TileDictionary As Dictionary(Of
```

```

Char, Integer), ByRef AllowedWords As List(Of String))
    Console.WriteLine()
    Console.WriteLine("TILE VALUES")
    Console.WriteLine()
    For Each Tile As KeyValuePair(Of Char, Integer) In
        TileDictionary
        Console.WriteLine("Points for " & Tile.Key & ": " &
            Tile.Value)
    Next
    Console.WriteLine()
    CalculateFrequencies(AllowedWords)
End Sub

```

Alternative answer

```

Sub CalculateFrequencies(ByRef AllowedWords As List(Of
String))
    Dim NumberOfTimes, Count As Integer
    Console.WriteLine("Letter frequencies in the allowed words
are:")
    For Each Letter In "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        Count = 0
        For Each Word In AllowedWords
            NumberOfTimes = Word.Split(Letter).Length - 1
            Count += NumberOfTimes
        Next
        Console.WriteLine(Letter & " " & Count)
    Next
End Sub

```

Alternative answer

```

Sub CalculateFrequencies(ByRef AllowedWords As List(Of
String))
    Dim Counts(25) As Integer
    For Count = 0 To 25
        Counts(Count) = 0
    Next
    Console.WriteLine("Letter frequencies in the allowed words
are:")
    For Each Word In AllowedWords
        For Each Letter In Word
            Counts(Asc(Letter) - 65) += 1
        Next
    Next
    For count = 0 To 25
        Console.WriteLine(Chr(count + 65) & " " & Counts(count))
    Next
End Sub

```

C#

```

private static void CalculateFrequencies(List<string>
AllowedWords)
{
    Console.WriteLine("Letter frequencies in the allowed words
are:");
    int LetterCount = 0;
    char LetterToFind;
    for (int Code = 0; Code < 26; Code++)
    {
        LetterCount = 0;
        LetterToFind = (char)(Code + 65);
        foreach (var Word in AllowedWords)
        {
            foreach (var Letter in Word)

```

```

        {
            if (Letter == LetterToFind)
            {
                LetterCount++;
            }
        }
    }
    Console.WriteLine(LetterToFind + " " + LetterCount);
}

private static void DisplayTileValues(Dictionary<char, int>
TileDictionary, List<string> AllowedWords)
{
    Console.WriteLine();
    Console.WriteLine("TILE VALUES");
    Console.WriteLine();
    char Letter;
    int Points;
    foreach (var Pair in TileDictionary)
    {
        Letter = Pair.Key;
        Points = Pair.Value;
        Console.WriteLine("Points for " + Letter + ": " + Points);
    }
    CalculateFrequencies(AllowedWords);
    Console.WriteLine();
}

```

Alternative answer

```

private static void CalculateFrequencies(List<string>
AllowedWords)
{
    Console.WriteLine("Letter frequencies in the allowed words
are:");
    int LetterCount = 0;
    string Alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    foreach (var Letter in Alphabet)
    {
        LetterCount = 0;
        foreach (var Words in AllowedWords)
        {
            LetterCount = LetterCount + (Words.Split(Letter).Length
- 1);
        }
        Console.WriteLine(Letter + " " + LetterCount);
    }
}

```

Alternative answer

```

private static void CalculateFrequencies(List<string>
AllowedWords)
{
    List<int> Counts = new List<int>() ;
    for (int i = 0; i < 26; i++)
    {
        Counts.Add(0);
    }
    foreach (var Words in AllowedWords)
    {
        foreach (var Letter in Words)
        {
            Counts[(int)Letter - 65]++;
        }
    }
}

```

```

    }
}
for (int a = 0; a < 26; a++)
{
    char Alpha =Convert.ToChar( a + 65);
    Console.WriteLine(Alpha + " " + Counts[a] );
}
}

```

Java

```

void calculateFrequencies(String[] allowedWords)
{
    int letterCount;
    char letterToFind;
    for (int count = 0; count < 26; count++)
    {
        letterCount = 0;
        letterToFind = (char) (65 + count);
        for(String word:allowedWords)
        {
            for(char letter : word.toCharArray())
            {
                if(letterToFind == letter)
                {
                    letterCount++;
                }
            }
        }
        Console.println(letterToFind + ", Frequency: " +
letterCount);
    }
}

void displayTileValues(Map tileDictionary, String[]
allowedWords)
{
    Console.println();
    Console.println("TILE VALUES");
    Console.println();
    for (Object letter : tileDictionary.keySet())
    {
        int points = (int)tileDictionary.get(letter);
        Console.println("Points for " + letter + ": " + points);
    }
    calculateFrequencies(allowedWords);
    Console.println();
}

```

Alternative answer

```

void calculateFrequencies(String[] allowedWords)
{
    int letterCount;
    String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    for(char letter: alphabet.toCharArray())
    {
        letterCount = 0;
        for(String word: allowedWords)
        {
            letterCount += word.split(letter + "").length - 1;
        }
        Console.println(letter + ", Frequency: " + letterCount);
    }
}

```


Alternative answer

```
void calculateFrequencies(String[] allowedWords)
{
    int[] counts = new int[26];
    for(String word: allowedWords)
    {
        for(char letter: word.toCharArray())
        {
            int letterPostion = (int)letter - 65;
            counts[letterPostion]++;
        }
    }
    for (int count = 0; count < 26; count++)
    {
        char letter = (char) (65 + count);
        Console.println(letter + ", Frequency: " + counts[count]);
    }
}
```

Pascal / Delphi

```
procedure CalculateFrequencies(AllowedWords : array of
string);
var
    Code, LetterCount : integer;
    LetterToFind, Letter : char;
    Word : string;
begin
    writeln('Letter frequencies in the allowed words are:');
    for Code := 0 to 25 do
    begin
        LetterCount := 0;
        LetterToFind := chr(65 + Code);
        for Word in AllowedWords do
        begin
            for Letter in Word do
            begin
                if Letter = LetterToFind then
                    LetterCount := LetterCount + 1;
            end;
        end;
        writeln(LetterToFind, ' ', LetterCount);
    end;
end;
```

(ii) Mark is for AO3 (evaluate)

**** SCREEN CAPTURE ****

Must match code from part (d)(i), including prompts on screen capture matching those in code.

Code for part (d)(i) must be sensible.

Screen capture(s) showing correct list of letter frequencies are displayed;

I. Ignore order of letter frequency pairs

I. any additional output eg headings like "Letter" and "Count"

Letter frequencies in the allowed words are:

- A 188704
- B 44953
- C 98231
- D 81731
- E 275582

F 28931
G 67910
H 60702
I 220483
J 4010
K 22076
L 127865
M 70700
N 163637
O 161752
P 73286
Q 4104
R 170522
S 234673
T 159471
U 80636
V 22521
W 18393
X 6852
Y 39772
Z 11772

Either:

enter the word you would like to play OR
press 1 to display the letter values OR
press 4 to view the tile queue OR
press 7 to view your tiles again OR
press 0 to fill hand and stop the game.

>

ALTERNATIVE ANSWERS (for different versions of text file)

Letter frequencies in the allowed words are:

A 188627
B 44923
C 98187
D 81686
E 275478
F 28899
G 67795
H 60627
I 220331
J 4007
K 22028
L 127814
M 70679
N 163547
O 161720
P 73267
Q 4104
R 170461
S 234473
T 159351
U 80579
V 22509
W 18377
X 6852
Y 39760
Z 11765

Either:

enter the word you would like to play OR
press 1 to display the letter values OR
press 4 to view the tile queue OR
press 7 to view your tiles again OR
press 0 to fill hand and stop the game.

>

Letter frequencies in the allowed words are:

```
A 5299
B 1105
C 2980
D 2482
E 7523
F 909
G 1692
H 1399
I 5391
J 178
K 569
L 3180
M 1871
N 4762
O 4177
P 1992
Q 122
R 4812
S 4999
T 4695
U 1898
V 835
W 607
X 246
Y 999
Z 128
```

Either:

```
enter the word you would like to play OR
press 1 to display the letter values OR
press 4 to view the tile queue OR
press 7 to view your tiles again OR
press 0 to fill hand and stop the game.
```

>

1

(e) (i) **All marks for AO3 (programming)**

Modifying subroutine `UpdateAfterAllowedWord`:

1. Correct subroutine call to `GetScoreForWordAndPrefix` added in `UpdateAfterAllowedWord`;
2. Result returned by `GetScoreForWordAndPrefix` added to `PlayerScore`;

A. alternative names for subroutine `GetScoreForWordAndPrefix` if match name of subroutine created

Creating new subroutine:

3. Subroutine `GetScoreForWordAndPrefix` created; **R.** if spelt incorrectly **I.** case
4. All data needed (`Word`, `TileDictionary`, `AllowedWords`) is passed into subroutine via interface;
5. Integer value always returned by subroutine;

Base case in subroutine:

6. Selection structure for differentiating base case and recursive case with suitable condition (word length of 0 // 1 // 2); **R.** if base case will result in recursion

7. If base case word length is 0 then value of 0 is returned by subroutine and there is no recursive call // if base case word length is 1 then value of 0 is returned by subroutine and there is no recursive call // if base case word length is 2 the subroutine returns 0 if the two-letter word is not a valid word and returns the score for the two-letter word if it is a valid word;

Recursive case in subroutine:

8. Selection structure that contains code that adds value returned by call to `GetScoreForWord` to score if word is valid; **A.** no call to subroutine `GetScoreForWord` if correct code to calculate score included in sensible place in `GetScoreForWordAndPrefix` subroutine **R.** if no check for word being valid
9. Call to `GetScoreForWordAndPrefix`;
10. Result from recursive call added to score;
11. Recursion will eventually reach base case as recursive call has a parameter that is word with last letter removed;

How to mark question if no attempt to use recursion:

Mark points 1-5 same as for recursive attempt. No marks awarded for mark points 6-11, instead award marks as appropriate for mark points 12-14.

12. Adds the score for the original word to the score once // sets the initial score to be the score for the original word; **A.** no call to subroutine `GetScoreForWord` if correct code to calculate score included in sensible place in `GetScoreForWordAndPrefix` subroutine. **Note for examiners:** there is no need for the answer to check if the original word is valid
13. Iterative structure that will repeat $n - 1$ times where n is the length of the word; **A.** $n - 2$ **A.** n
14. Inside iterative structure adds score for current prefix word, if it is a valid word, to score once; **A.** no call to `GetScoreForWord` if own code to calculate score is correct

Max 10 if code contains errors

Max 8 if recursion not used in an appropriate way

11

Python 2

```
def UpdateAfterAllowedWord(Word, PlayerTiles, PlayerScore,
    PlayerTilesPlayed, TileDictionary, AllowedWords):
    PlayerTilesPlayed += len(Word)
    for Letter in Word:
        PlayerTiles = PlayerTiles.replace(Letter, "", 1)
    PlayerScore += GetScoreForWordAndPrefix(Word,
    TileDictionary, AllowedWords)
    return PlayerTiles, PlayerScore, PlayerTilesPlayed

def GetScoreForWordAndPrefix(Word, TileDictionary,
    AllowedWords):
    if len(Word) <= 1:
        return 0
    else:
        Score = 0
        if CheckWordIsValid(Word, AllowedWords):
            Score += GetScoreForWord(Word, TileDictionary)
```

```

    Score += GetScoreForWordAndPrefix(Word[0:len(Word) - 1],
TileDictionary, AllowedWords)
    return Score

```

Alternative answer

```

def GetScoreForWordAndPrefix(Word,TileDictionary,
AllowedWords):
    Score = 0
    if CheckWordIsValid(Word,AllowedWords):
        Score += GetScoreForWord(Word, TileDictionary)
    if len(Word[:-1]) > 0:
        Score +=GetScoreForWordAndPrefix(Word[:-1],
TileDictionary,AllowedWords)
    return Score

```

Python 3

```

def UpdateAfterAllowedWord(Word, PlayerTiles, PlayerScore,
PlayerTilesPlayed, TileDictionary, AllowedWords):
    PlayerTilesPlayed += len(Word)
    for Letter in Word:
        PlayerTiles = PlayerTiles.replace(Letter, "", 1)
    PlayerScore += GetScoreForWordAndPrefix(Word,
TileDictionary, AllowedWords)
    return PlayerTiles, PlayerScore, PlayerTilesPlayed

```

```

def GetScoreForWordAndPrefix(Word, TileDictionary,
AllowedWords):
    if len(Word) <= 1:
        return 0
    else:
        Score = 0
        if CheckWordIsValid(Word, AllowedWords):
            Score += GetScoreForWord(Word, TileDictionary)
        Score += GetScoreForWordAndPrefix(Word[0:len(Word) - 1],
TileDictionary, AllowedWords)
    return Score

```

Alternative answer

```

def GetScoreForWordAndPrefix(Word,TileDictionary,
AllowedWords):
    Score = 0
    if CheckWordIsValid(Word,AllowedWords):
        Score += GetScoreForWord(Word, TileDictionary)
    if len(Word[:-1]) > 0:
        Score +=GetScoreForWordAndPrefix(Word[:-1],
TileDictionary,AllowedWords)
    return Score

```

Visual Basic

```

Sub UpdateAfterAllowedWord(ByVal Word As String, ByRef
PlayerTiles As String, ByRef PlayerScore As Integer, ByRef
PlayerTilesPlayed As Integer, ByVal TileDictionary As
Dictionary(Of Char, Integer), ByRef AllowedWords As List(Of
String))
    PlayerTilesPlayed += Len(Word)
    For Each Letter In Word
        PlayerTiles = Replace(PlayerTiles, Letter, "", , 1)
    Next
    PlayerScore += GetScoreForWordAndPrefix(Word,
TileDictionary, AllowedWords)
End Sub

```

```

Function GetScoreForWordAndPrefix (ByVal Word As String, ByVal
TileDictionary As Dictionary(Of Char, Integer), ByRef
AllowedWords As List(Of String)) As Integer
    Dim Score As Integer
    If Len(Word) <= 1 Then
        Return 0
    Else
        Score = 0
        If CheckWordIsValid(Word, AllowedWords) Then
            Score += GetScoreForWord(Word, TileDictionary)
        End If
        Score += GetScoreForWordAndPrefix (Mid(Word, 1, Len(Word)
- 1), TileDictionary, AllowedWords)
    End If
    Return Score
End Function

```

Alternative answer

```

Function GetScoreForWordAndPrefix (ByVal Word As String, ByVal
TileDictionary As Dictionary(Of Char, Integer), ByRef
AllowedWords As List(Of String)) As Integer
    Dim Score As Integer = 0
    If CheckWordIsValid(Word, AllowedWords) Then
        Score += GetScoreForWord(Word, TileDictionary)
    End If
    If Len(Word) - 1 > 0 Then
        Score += GetScoreForWordAndPrefix (Mid(Word, 1, Len(Word)
- 1), TileDictionary, AllowedWords)
    End If
    Return Score
End Function

```

C#

```

private static void UpdateAfterAllowedWord(string Word, ref
string PlayerTiles, ref int PlayerScore, ref int
PlayerTilesPlayed, Dictionary<char, int> TileDictionary,
List<string> AllowedWords)
{
    PlayerTilesPlayed = PlayerTilesPlayed + Word.Length;
    foreach (var Letter in Word)
    {
        PlayerTiles =
PlayerTiles.Remove (PlayerTiles.IndexOf (Letter), 1);
    }
    PlayerScore = PlayerScore + GetScoreForWordAndPrefix (Word,
TileDictionary, AllowedWords);
}

```

```

private static int GetScoreForWordAndPrefix(string Word,
Dictionary<char, int> TileDictionary, List<string>
AllowedWords)
{
    int Score = 0;
    if (Word.Length <= 1)
    {
        return 0;
    }
    else
    {
        Score = 0;
        if (CheckWordIsValid(Word, AllowedWords))
        {
            Score = Score + GetScoreForWord(Word, TileDictionary);
        }
    }
}

```

```

    }
    Score = Score +
    GetScoreForWordAndPrefix(Word.Remove(Word.Length - 1),
    TileDictionary, AllowedWords);
    return Score;
}
}

```

Alternative answer

```

private static int GetScoreForWordAndPrefix(string Word,
Dictionary<char, int> TileDictionary, List<string>
AllowedWords)
{
    int Score = 0;
    if (CheckWordIsValid(Word, AllowedWords))
    {
        Score = Score + GetScoreForWord(Word, TileDictionary);
    }
    if (Word.Remove(Word.Length - 1).Length > 0)
    {
        Score = Score +
        GetScoreForWordAndPrefix(Word.Remove(Word.Length - 1),
        TileDictionary, AllowedWords);
    }
    return Score;
}

```

Java

```

int getScoreForWordAndPrefix(String word, Map tileDictionary,
String[] allowedWords)
{
    int score = 0;
    if(word.length() < 2)
    {
        return 0;
    }
    else
    {
        if(checkWordIsValid(word, allowedWords))
        {
            score = getScoreForWord(word, tileDictionary);
        }
        word = word.substring(0, word.length()-1);
        return score + getScoreForWordAndPrefix(word,
        tileDictionary, allowedWords);
    }
}

```

```

void updateAfterAllowedWord(String word, Tiles
playerTiles,
    Score playerScore, TileCount playerTilesPlayed, Map
tileDictionary,
    String[] allowedWords)
{
    playerTilesPlayed.numberOfTiles += word.length();
    for(char letter : word.toCharArray())
    {
        playerTiles.playerTiles =
        playerTiles.playerTiles.replaceFirst(letter+"", "");
    }
    playerScore.score += getScoreForWordAndPrefix(word,
    tileDictionary, allowedWords);
}

```

Alternative answer

```
int getScoreForWordAndPrefix(String word, Map tileDictionary,
String[] allowedWords)
{
    int score = 0;
    if(checkWordIsValid(word, allowedWords))
    {
        score += getScoreForWord(word, tileDictionary);
    }
    word = word.substring(0, word.length()-1);
    if(word.length()>1)
    {
        score += getScoreForWordAndPrefix(word, tileDictionary,
allowedWords);
    }
    return score;
}
```

Pascal / Delphi

```
function GetScoreForWordAndPrefix(Word : string;
TileDictionary : TileDictionary; AllowedWords : array of
string) : integer;
var
    Score : integer;
begin
    if length(word) <= 1 then
        Score := 0
    else
        begin
            Score := 0;
            if CheckWordIsValid(Word, AllowedWords) then
                Score := Score + GetScoreForWord(Word,
TileDictionary);
            Delete(Word,length(Word),1);
            Score := Score + GetScoreForWordAndPrefix(Word,
TileDictionary, AllowedWords);
        end;
        GetScoreForWordAndPrefix := Score;
    end;
end;

procedure UpdateAfterAllowedWord(Word : string; var
PlayerTiles : string; var PlayerScore : integer; var
PlayerTilesPlayed : integer; TileDictionary : TileDictionary;
var AllowedWords : array of string);
var
    Letter : Char;
begin
    PlayerTilesPlayed := PlayerTilesPlayed + length(Word);
    for Letter in Word do
        Delete(PlayerTiles,pos(letter, PlayerTiles),1);
        PlayerScore := PlayerScore +
GetScoreForWordAndPrefix(Word, TileDictionary,
AllowedWords);
    end;
```

(ii) Mark is for AO3 (evaluate)

**** SCREEN CAPTURE ****

Must match code from part (e)(i), including prompts on screen capture matching those in code.

Code for part (e)(i) must be sensible.

Screen capture(s) showing that the word abandon was entered and the

new score of 78 is displayed;

Do you want to:

replace the tiles you used (1) OR

get three extra tiles (2) OR replace the tiles you used

and get three extra tiles (3) OR

get no new tiles (4)?

>4

Your word was: ABANDON

Your new score is: 78

You have played 7 tiles so far in this game.

Press Enter to continue

1

[37]

Q2.

(a) **Mark is for AO1 (knowledge)**

Merge sort;

1

(b) **Mark is for AO1 (understanding)**

4;

1

[2]

Q3.

(a) **Mark is for AO1 (understanding)**

False;

1

(b) **Mark is for AO1 (understanding)**

THEN Failed \leftarrow True;

1

(c) **All marks for AO1 (understanding)**

$L \leftarrow M - 1$;

Mark as follows:

1 mark: L;

1 mark: $\leftarrow M - 1$;

Maximum 1 mark: If not correct

2

(d) **Mark is for AO1 (understanding)**

$O(k^n)$;

A k^n

1

(e) **Mark is for AO1 (knowledge)**

$O(\log n)$;

A $\log n$

1

(f) **Mark is for AO1 (knowledge)**

$O(1)$;

A 1

1

(g) **Mark is for AO1 (knowledge)**

$O(n)$;

A n

1

(h) **All marks AO1 (understanding)**

1 mark: As the size of the list increases the time taken to search for an item increases; at the same rate; //

1 mark: A linear search looks at each item in the list in turn (until it reaches the end of the list or the item being searched for is found); so if there are n items in the list the worst case would be n comparisons;

2

[10]

Q4.

(a)

Position	Value	Order Examined In
8	Philip	1
10	Ravi	3
11	Richard	4
12	Timothy	2

1 mark for row 8 correct

1 mark for row 11 correct

1 mark for both rows 10 and 12 correct

Do not award mark for a particular number if same number is written more than once

EXAM PAPERS PRACTICE

3

(b) 8

1

(c) Order of complexity	Tick one box
$O(\log_2 n)$	<input checked="" type="checkbox"/>
$O(n)$	<input type="checkbox"/>
$O(n^2)$	<input type="checkbox"/>

Do not award mark if more than one box ticked

1

[5]

Q5.

(a) A procedure that is defined in terms of itself;

A A procedure that calls itself

R re-entrant

1

- (b) Store return addresses;
Store parameters;
Store local variables/ return values;

Max 1

(c)

Number	Entry	Output
11	1	
11	2;	
11	3;	
11	4;	4;

4

- (d) A linear search//
To find/output the position/index of Number in Items;

1

- (e) Number is not an entry in Items// Stack overflows;

1

- (f) Test for reaching the end of Items;

1

- (g) Binary Search;
An iterative solution;

Max 1

EXAM PAPERS PRACTICE

[10]

Q6.

(a)

Number	Lower	Upper	Current
12	1	9	
		5	5
	3		3
	4	4	4

Value returned	4
----------------	---

1 mark for 1st row (12, 1, 9)

2 marks for second row (1 mark for each 5)

2 marks for 3rd row (3 and 3)

2 marks for 4th row (1 mark for Lower = 4, 1mark for upper = 4)
1 mark for correct return value

8

- (b) Find the position of 12/ a number in the array// search for 12/ a number in the array;

1

[9]

Q7.

- (a) (i) 271;

1

- (ii) The required item might be the 271st one/last one/ not be present// Every item accessed;

1

- (b) (i) 9;

1

- (ii) Each comparison halves the number of items to be accessed//271 lies between 2^8 and 2^9 ;

1

- (c)



EXAM PAPERS PRACTICE

Count1	Count2	Temp	A				
			[1]	[2]	[3]	[4]	[5]
-	-	-	23	45	16	12	31
1	1						
	2	45		16	45		
	3	45			12	45	
	4	45				31	45
2	1	23	16	23			
	2	23		12	23		
	3						
	4						
3	1	16	12	16			
	2						
	3						
	4						
4	1						
	2						
	3						
	4						

E

1 mark for Count1
1 mark for Count2
1 mark for Temp

5

(ii) (bubble) sort the items into ascending order;

1

(iii) Reduce the number of tests each pass// stop when no swaps occur during a pass//Add a flag No Swaps to indicate when no swaps occur// change loop control to Repeat until no swaps// sort variable sized array;

1

[11]

Q8.

- (a) (i) • poorly structured code;
 • uses GoTo statements;
 • the flow of control jumps out of a loop;
 • nothing reported to the user when no matching name found;
 • abbreviated variable for 'position' variable;
 • ReadLn is better than Read;
 • Program only iterates once / considers only the first array element;
 • (if duplicates) only the first matching surname is found;
 • (loop terminates at 20) does not allow for additional array /name entries;

A poor layout - excessive indentation used;

I. variable declaration // reference to the syntax

Max 2

- (ii) All statements must have correct identifier name correct data type (String / Text // Integer / Byte / Word / Int / Shortint / Short as appropriate)

In addition, either array must have brackets to indicate an 'array' 19/20 to indicate a range;

Max 2

- (b) Initialisation of counter or Boolean variable
 P := 1 / P := 0 / For P := 1 to 20 // IsFound := False;

Looping

LOOP UNTIL // DO WHILE // WHILE DO // REPEAT UNTIL and used at the beginning/end of a code block as appropriate;

Some loop condition is met

(P = 20/21) OR IsFound = TRUE / P = 20/21 // IsFound = TRUE / IsFound;

IF with use of the array

IF NoOfClaims [P];

Selection condition

>4 / >=5;

Loop counter incremented

P = P+1

Final output

Correct logic followed with OUTPUT 'Yes'

A multiple times

Final output

Correct logic followed with OUTPUT 'No'

R Multiple times

R 'Prose' scores 0

5

[9]

Q9.

Compare Pascal with middle item of list / Lisp;

Compare Pascal with middle item of upper sublist / Prolog;

Compare Pascal with Pascal // compare only item in this sublist to get a match;

Lose 1 mark if Pascal not explicit in comparison

Stop marking from time it goes wrong

OR

List[4] = Pascal? False;

A [4] = Pascal

R 4 = Pascal

List[6] = Pascal? False;

List[5] = Pascal? True;

If formula explicit, follow through on formula

[3]

Q10.

(a) A procedure/routine which calls itself//is defined in terms of itself;

R re-entrant

A function instead of procedure

R program iteration Talked Out (no mark)

1

(b) (i)

E	L	H	M	List[M]	Printed Output
6502	1	11 ;	6	5789 ;	
6502	7	11 ;	9	8407 ;	
6502	7	8 ;	7	6502 ;	
					True;

Accept True in row 3

Marks in each row for all three/two parts correct

Accept empty cell to mean: same as in previous row.

Stop marking when logic goes wrong

7

(ii) Binary search;;

Search;

R any other type of search

2

[10]

Q11.

Compare Newcastle with (middle item of list), Manchester;

Compare Newcastle with (middle item of upper sublist), Sheffield;

Compare Newcastle with Newcastle // compare only item (in lower sublist of this upper sublist) to get a match;

Lose 1 mark if Newcastle not explicit in comparison stop marking from time it goes wrong

OR

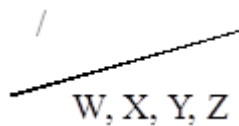
List[4] = Newcastle? False;
A [4] = Newcastle
R 4 = Newcastle
List[6] = Newcastle? False;
List[5] = Newcastle? True;

If formula explicit, follow through on formula

[3]

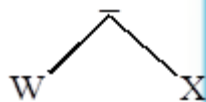
Q12.

- (a) Root, (1)
Branch (1)
Leaf node(1)
Must circle!



3

- (b) Left sub-tree (1)



Right sub-tree(1)



EXAM PAPERS PRACTICE

2

- (c) $W-X / Y+Z$
1 1 1

A column vector
Spurious punctuation (1)

3

[8]

Q13.

- (a) (i) 8;

1

- (ii) Each time a comparison is made in a binary search the number of items to be searched / list is halved;

// 137 lies between 2^7 and 2^8 ;

Could give (ii) even if (i) incorrect

1

(b) (i) 137;

1

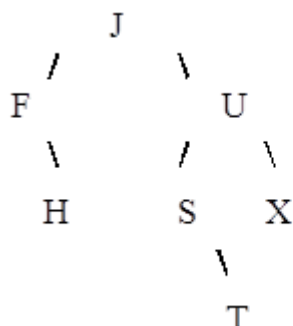
(ii) In a linear search of 137 items, the required item might be the 137th one;
Need a termination – must explain why 137 is the maximum

1

[4]

Q14.

(a)



J 1 mark

F and U 1 mark

H 1 mark

S, X, T 1 mark

A mirror image (this time)

4

(b) 'T' 4; 'U';
'T' 5; 'S';
'T' 7; 'T';

No penalty if candidate gets 'item' wrong
Ignore 'item' column

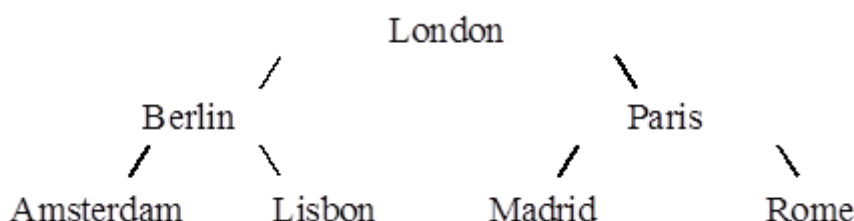
6

[10]

EXAM PAPERS PRACTICE

Q15.

(a)



1 mark for correct position of London,
1 mark for correct position of Berlin and Paris,
1 mark for Amsterdam and Lisbon correct,
1 mark for Madrid and Rome correct
No follow through in this part of the question
If consistent mirror image give marks
Note (b) and (c) must follow on

4

(b) Root node marked correctly

Tick by question

- (c) London, Paris Madrid; *in correct order*

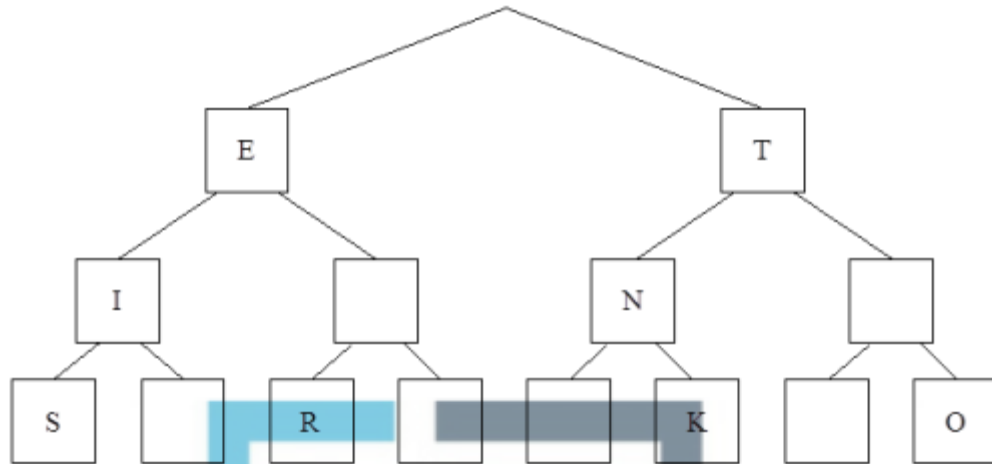
1

1

[6]

Q16.

- (a) 1 mark for each letter correctly placed:



1+1

2

- (b) I;T;

1+1

2

[4]

Q17.

- (a) Array must be sorted (1), on the field being used as the search key (1)

2

- (b) Description must include the following points: Find median record of array (1) Compare key field of record at median position with required search key, exit if found (1) If search key lower (i.e. required record in first half), discard second half, else discard first half (1) Repeat process (1) until either found, or no further division possible so record does not exist (1)

5

- (c) On each iteration, half the possible matches are eliminated, compared with only one for the linear search (2)
Linear search on average scans $n/2$ records, compared with $\log_2 n$ which is smaller "Looks at fewer records" without further explanation (1)

2

[9]

Q18.

- (i) File is sorted on appropriate key (1), and direct access to records is possible

(1)

2

- (ii) Binary chop - number of comparisons is of the order of $\log_2 n$ instead of $n/2$
(verbal description OK – e.g. in binary chop, number of records left to examine is halved each iteration instead of being reduced by one)

2

[4]

Q19.

(a)

Low	High	Middle	Found
		5	
6		8	
	7	6	
7		7	true

1 mark for each entry above (as far as first incorrect entry)

Mark row by row

Max 7

- (b) Binary search/chop
Iterative (no synonyms)
(Specific searches not on AS syllabus - search sufficient for mark)

1

[8]

EXAM PAPERS PRACTICE

Examiner reports

Q1.

- (a) This was the first of the questions that required modifying the Skeleton Program. It was a simple question that over 80% of students were able to answer correctly. When mistakes were made this was normally because tiles other than just J and X were also changed to be worth 4 points.
- (b) Like question (a), this question was normally well-answered with almost all student getting some marks and about 75% obtaining full marks. Where students didn't get full marks this was normally due to the conditions on the loop being incorrect which prevented the values of 1 and / or 20 from being valid.
- (c) For this question students had to replace the linear search algorithm used to check if a word is in the list of allowed words with a binary search algorithm. An example of how a binary search algorithm works was included on the question paper but if a similar question is asked in the future that may not be done. A mixture of iterative and recursive solutions were seen. The most common error made by students who didn't get full marks but made a good attempt at answering the question was to miss out the condition that terminates the loop if it is now known that the word is **not** in the list.
- (d) Students found question (d) easier than questions (c) and (e). Better answers made good use of iteration and arrays / lists, less efficient answers which used 26 variables to store the different letter counts could also get full marks. Some students added code in their new subroutine to read the contents of the text file rather than pass the list as a parameter to the subroutine; this was not necessary but was not penalised.
- (e) Question (e) asked students to create a recursive subroutine. If students answered the question without using recursion they could still get 9 out of the 12 marks available.

It was disappointing that many students did not include any evidence of their attempt to answer the question. Good exam technique would be to include some program code that answers some part or parts of the question. For instance, in question (e) students could get marks for creating a subroutine with the specified name and calling that subroutine – even if the subroutine didn't do anything. There are many examples of subroutines and subroutine calls in the Skeleton Program that students could have used to help them obtain some marks on this question.

A number of very well-written subroutines were seen that made appropriate use of recursion and string handling. Some good recursive answers did not get full marks because they did not include a check that the word / prefix passed as a parameter was valid before the tile points included in the word were used to modify the score, this meant that all prefixes would be included in the score and not just the valid prefixes. Another frequent mistake came when students wrote their own code to calculate the score for a prefix rather than use the existing subroutine included in the Skeleton Program that calculated the score for a word – if done correctly full marks could be obtained by doing this but a number of students made mistakes when writing their own score-calculating code.

Q2.

This was the Section A question that students found hardest, with very few getting full

marks. Not many students were able to identify the time complexity of either merge sort or (to a lesser extent) bubble sort and a significant number of students thought that the binary search and/or the post-order tree traversal would not be used to solve tractable problems.

When students could state the time complexity of the bubble sort algorithm they were rarely able to clearly explain why $O(n^2)$ was the correct answer.

Q4.

Part (a): The binary search method was well understood and the majority of candidates were able to correctly label the sequence of four items that would be checked to search for the name "Richard". Some candidates missed out on the final mark by not realising that there would be a fourth comparison, i.e. "Richard" compared with "Richard". A minority of candidates applied the linear search method instead, labelling each of the names from Adam down to Richard consecutively from 1 to 11.

Part (b): This part was not tackled as well as part (a). Most candidates seemed to realise that the answer involved logarithms or powers of two, but the most common response was seven rather than eight, the correct answer. When calculating the number of comparisons required to search a list of n items, $\log_2(n)$ should be calculated and the result then rounded up. So $\log_2(137) = 7.10$ to 2 decimal places, which rounds up to 8.

Part (c): This question part was well answered, with over half of the candidates correctly identifying the complexity of the binary search method as $O(\log_2 n)$.

Q5.

Candidates generally scored well on this question. Recursively-defined was well understood although many candidates were unable to describe the use of the stack well enough. It was pleasing to see the majority of candidates obtaining most of the marks on part (c). Candidates often failed to obtain the mark for part (d) due to inadequate descriptions. Although many candidates provided a situation where the algorithm will fail, fewer were able to suggest a suitable modification. Once again this was often due to an inability to express themselves well. A wide range of answers were supplied for part (g) but a substantial number of correct responses were given.

Q6.

Very few candidates did not get some marks for the trace and many returned full marks for this part of the question. Some candidates who did achieve full marks on part (a) could not say what the algorithm does. Many candidates seemed to make a wild guess.

Q7.

It was pleasing to see many good answers to parts (a) and (b) although a number of candidates failed to obtain full marks through inadequate explanations. Part (c) was disappointing with few candidates completing the trace table correctly. Nevertheless it was pleasing to see a greater number of candidates able to partially complete a dry run. A surprising number of candidates were able to state that this was a bubble sort even though they failed to complete the trace table. Fewer were able to give a suitable improvement. The most common incorrect suggestion was to "make the algorithm recursive".

Q8.

- (a) (i) The use of GoTo statements has not previously been examined on this paper and most candidates struggled to suggest a single reason why this was poorly designed code, despite a large number of acceptable answers. The most common correct answers were that the use of GoTo statements gives rise to code which is difficult to follow and trace; there is no output produced when the SearchName value is not found; when there is more than one occurrence of SearchName in the PolicyHolder array, the program will output the number of claims value for the first occurrence of the name only.
- (ii) Few marks were obtained here with most candidates failing to give the bounds of the array for PolicyHolder or NoOfClaims, or omitting a data type for the identifier.
- (b) Candidates should be able to write small amounts of program code in a unit that has the word 'programming' in its title. Knowledge of loops other than a For loop was rare. It was hoped that candidates would have constructed a Repeat – Until or While loop which terminated when a NoOfClaims value of 5 or more was found. Candidates who used a For loop were, however, still able to score the maximum 5 marks.

Examiners were not looking for the correct use of exact syntax for the language as stated by the candidate.

The use of IF statements was better understood, but this often did not extend to using an array index for the NoOfClaims as part of the IF statement. Very many candidates used the maths operator incorrectly, e.g. \geq or more usually $=>$. Quite a few candidates reversed the logic testing for <5 and gave appropriate output for which they gained marks. Most popular languages seen were Pascal and Visual Basic but the candidates that used C on the whole answered the question very well indeed.

Q9.

Many candidates scored full marks on this question, but a significant minority do not appear to know how a binary search algorithm operates and could not write down the actual comparisons needed: Pascal is first compared with the middle item, Lisp. Then Pascal is compared with the middle item of the second sub-list, Prolog. Many candidates failed to state that a third comparison has to be made, i.e. comparing Pascal with Pascal before it can be stated that Pascal has been found.

Q10.

- (a) Most candidates could correctly state that recursively defined means that a procedure is defined in terms of itself or that it calls itself. Some candidates failed to gain marks because they could not express this clearly enough. A common misconception was that the procedure was in a loop.
- (b) Many candidates managed to gain full marks for completing the trace table:

E	L	H	M	List[M]	Printed Output
6502	1	11 ;	6	5789 ;	
6502	7	11 ;	9	8407 ;	

6502	7	8 ;	7	6502 ;	
					True;

A common mistake was to have False as printed output in the first 2 rows until it changed to True.

Some candidates who correctly completed the trace table could not see that the process was a binary search and some who did not complete the table correctly did manage to identify the process correctly.

Candidates should be aware of the need to fill in trace tables carefully, showing how values change chronologically. This may mean leaving some cells empty if no values are assigned to variables initially.

Q11.

Many candidates scored full marks on this question, some lost marks for not explicitly stating that each middle value had to be compared with the value searched for. Especially at the last step, when Newcastle is the only element left in the list, a comparison still has to be done to check that it is the required value.

Q12.

This was well answered by many candidates, although here again, marks were lost through lack of care in carrying out the clearly written instructions. The instructions in part (a) were to circle and label the three parts of the tree. An un-circled, or (more rarely) unlabelled part was not credited. Very few candidates were unable to draw the left and right sub-trees correctly, and most could perform in-order traversal.

Q13.

This question involved comparing the number of items which would be accessed when searching a list of 137 items using a binary search as opposed to a linear search. For the binary search, answers varied from 1 (the search goes directly to required item) to 256 or 28, although the reason for this was not made clear. 68 or 69 were common errors. The correct answer was 8. For the reason, candidates frequently had the idea of discarding the unwanted half, but the concept of this being repeated in each repetition was required for credit. A correct reason was credited even if the candidate had miscalculated and given an answer of 7 or, occasionally, 9.

For the linear search, the answers were more often correct, although 138 or 136 were sometimes given. A good reason here was 'Although most searches will take less, if the required item is the last in the list then all items will be accessed before it is reached'. Too many candidates missed this mark by stating that every item needed to be looked at; some even stating that the list would be un-sorted. A linear search is performed on a sorted list. Reference to looking at every file resulted in the answer being talked out.

Q14.

- (a) The tree was generally well created. A few candidates produced mirror images, which were accepted this time. However, candidates need to be aware that a binary search tree stores lower values in the left sub-tree and higher values in the right sub-tree. A few candidates produced a balanced binary tree with each node having two sub nodes, which was not correct.

- (b) Only the better candidates seemed to score full marks here. Dry-running an algorithm does not seem to have received sufficient attention in the preparation of a number of candidates. The fact that 'Item' did not change its value seemed to be noticed by only a few candidates. Candidates did not appear to appreciate that the algorithm was using the binary search tree, which the candidates had drawn in part (a).

Q15.

This was usually answered correctly, with a few candidates getting some of the nodes wrong. Some candidates did not answer part (c) correctly by knowing about traversal of binary trees (which is not in the specification of CPT I) and then applying this rather than the straightforward binary search. Many candidates did not mention the final node in their list of data items accessed. This was accepted this time, but will not gain credit in future years. Clearly, the final node has to be accessed to see that the search has been successful.

Q16.

This was usually answered correctly.

Q17.

Knowledge of sort procedures seems good, but the ability to express it with anything like the precision of language expected at A-level is not. In particular, far too many candidates treat file, record and field as interchangeable terms, which is unacceptable at this level.

For part (a), nearly all candidates pointed out that an array needed to be sorted for a binary chop search to work, but very few realised that, say, an array with ten fields could be sorted in ten different orders and only one would work - that in which the sort key was the field being searched.

For part (b), most scored reasonably well, although as usual descriptions were full of waffle. Candidates seemed to be trying to paraphrase an algorithm they had been taught - while formal algorithms are not in the syllabus for this paper, if a candidate wants to present an answer in pseudocode or even flowchart form it will receive full credit. A common mistake was to say that the search key was compared to "the middle record" (rarely the more accurate "key field of the median record") and one half or the other discarded, ignoring the possibility that it might be matched enabling the search to end. Many candidates failed to indicate how the search could indicate failure if the desired key did not exist in the array. Another mistake is to describe the process by describing the progress of a search of specimen data - rarely adequate because it misses many of the things that can happen with different data, and does not bring out the iterative nature of the process because the list is so short.

In part (c), it was necessary to indicate something about the workings of both techniques to gain both marks (preferably referring back to part (b)) - the bald (and common) "it looks at fewer records" shows little understanding. Not many appreciated the significance of the word "normally" - a linear search is actually much faster than a binary chop in finding a key such as "aardvark"!

Q18.

Most candidates knew that a binary search only works on a sorted file, but hardly any also realised that direct access to records is also essential. The last part also showed weaknesses in exam technique: candidates did not read carefully what the question was

asking for. A very large number laboured mightily for half a page or more describing the binary chop in detail: this was unnecessary, the question merely wanted an explanation of why it is faster than a linear search. “The binary chop eliminates at least half the possible records in each iteration, whereas a linear search can only eliminate at most one” would have scored both marks.

Q19.

Those candidates who read this question carefully gained full marks, and most of those were then able to identify the routine as a binary search routine. Candidates were given credit for simply saying “a search routine” as particular types of search routine are not specified on the AS syllabus. Although the function *Int* was explained, many candidates gave the first middle value as 5.5 or rounded up to 6. A very common error was to confuse the subscripts with the actual data. Surprisingly, many calculated the first three values correctly, and then made an error on the fourth.

