1. **Defining the shuffle function:**

|  |  |
| --- | --- |
| 01 | **from** random **import** **\*** |
| 02 | input1 **=** **[**5**,**3**,**8**,**6**,**1**,**9**,**2**,**7**]** |
| 03 | **print(**input1**)** |
| 04 |  |
| 05 | **def** shuff**(**L**):** |
| 06 | '''Randomises order of elements in the taken array''' |
| 07 | **for** i **in** range**(**len**(**L**)):** |
| 08 |  |
| 09 |  |
| 10 |  |
| 11 |  |
| 12 | **return** L |
| 13 |  |
| 14 | **print(**shuff**(**input1**))** |

1. **Counting the trailing zeros in a factorial number:**

|  |  |
| --- | --- |
| 01 | num **=** int**(**input**(**"num: "**))** |
| 02 |  |
| 03 | **def** trailingZero**(**t**):** |
| 04 | '''Returns number of trailing zeros of answer from a factorial int''' |
| 05 | **def** fact**(**n**):** |
| 06 | '''Returns factorial number of argument''' |
| 07 | **if** n **==** 1**:** |
| 08 | **return** n |
| 09 | **else:** |
| 10 | **return** n **\*** fact**(**n**-**1**)** |
| 11 | count **=** 0 |
| 12 | factAns **=** str**(**fact**(**t**))[::-**1**]** # reversed order of (string) integer value |
| 13 | **for** i **in** range**(**len**(**factAns**)):** # counts 0s: stops if no more at trailing end |
| 14 | **if** factAns**[**i**]** **!=** "0"**:** |
| 15 | **break** |
| 16 | **else:** |
| 17 | count**+=**1 |
| 18 | **return** count |
| 19 |  |
| 20 | **print(**trailingZero**(**num**))** |

1. **Returning the highest square number less or equal to the function parameter:**

|  |  |
| --- | --- |
| 01 | **from** math **import** sqrt |
| 02 | num **=** int**(**input**(**"number: "**))** |
| 03 |  |
| 04 | **def** highestSquareNumber**(**x**):** |
| 05 | ''' Returns closest perfect square number ''' |
| 06 |  |
| 07 | psn **=** **[]** ## GENERATING PERFECT SQUARE NUMBERS |
| 08 | **for** i **in** range**(**10000**):** |
| 09 | **if** sqrt**(**i**)** **%** 1 **==** 0**:** |
| 10 | psn**.**append**(**i**)** |
| 11 |  |
| 12 | **for** i **in** range**(**len**(**psn**)):** ## RETURNS CLOSEST PERFECT SQUARE NUMBER |
| 13 | **if** x **<=** psn**[**i**]:** |
| 14 | **return** psn**[**i**]** |
| 15 |  |
| 16 | **print(** highestSquareNumber**(**num**)** **)** |
|  |  |

1. **Pseudocode for questions 1 and 2:**

**Question 1:**

|  |  |
| --- | --- |
| Pseudocode | Time of cost |
| from random import \*  input1 = [5,3,8,6,1,9,2,7]  print(input1)    def shuff(L):  for i in range(len(L)):  iRandom = randint(0,len(L)-1)  temp = L[i]  L[i] = L[iRandom]  L[iRandom] = temp  return L  print(shuff(input1)) | 1  1  n  n  n  n  n  1  1 |
| **Runtime bound:** | f(n) = 1 + 1 + n + n + n + n + n + 1 + 1  = 5n + 4 |
| **Complexity:** | O(n) |

**Question 2:**

|  |  |
| --- | --- |
| Pseudocode | Time of cost |
| num = int(input("num: "))  def trailingZero(t):  def fact(n):  if n == 1:  return n  else:  return n \* fact(n-1)  count = 0  factAns = str(fact(t))[::-1]  for i in range(len(factAns)):  if factAns[i] != "0":  break  else:  count+=1  return count  print(trailingZero(num)) | 1  1  1  n  1  1  n  n  1  n  1  1 |
| **Runtime bound:** | f(n) = 2n + 2n? + 8  = 2n + 2mn + 8  = 2n + 2n + 8  = 4n + 8 |
| **Complexity:** | O(n) |

2. **Function reversing the words in a given sentence:**

|  |  |
| --- | --- |
| Pseudocode | Time of cost |
| String word <- "This is awesome"  WORD\_REVERSE(word):  word <- String [] word  reversedWord <- New String Array []  For i <- a.length-1 Downto 0 {  Append a[i] to reversedWord  }  reversedWord <- joined String reversedWord  RETURN reversedWord  PRINT WORD\_REVERSE(word) | 1  1  1  n  n  1  1  1 |
| **Runtime bound:** | f(n) = 2n + 6 |
| **Complexity:** | O(n) |

**High-level language code (Python):**

|  |  |
| --- | --- |
| 01 | word **=** "This is awesome" |
| 02 |  |
| 03 | **def** wordReverse**(**w**):** |
| 04 | '''Returns the string argument reversed by word''' |
| 05 | w **=** w**.**split**(**' '**)** |
| 06 | reversedWord **=** **[]** |
| 07 | ## w = w[::-1] |
| 08 | **for** i **in** range**(**len**(**w**)-**1**,** **-**1**,** **-**1**):** |
| 09 | reversedWord**.**append**(**w**[**i**])** |
| 10 | reversedWord **=** ' '**.**join**(**reversedWord**)** |
| 11 | **return** reversedWord |
| 12 |  |
| 13 | **print(**wordReverse**(**word**)** |

1. **Checking if a given integer is a prime number:**

|  |  |
| --- | --- |
| Pseudocode | Time of cost |
| number <- any given integer value  IS\_PRIME(num, test <- number)  If num ≤ 2 OR test = 1 {  Return TRUE  } Else If num mod test = 0 {  Return IS\_PRIME(num, test-1)  }  Return FALSE  }  PRINT IS\_PRIME(number) | 1  1  1  1  n  1  1 |
| **Runtime bound:** | f(n) = n + 6 |
| **Complexity:** | O(n) |

**High-level language code (Python):**

|  |  |
| --- | --- |
| 01 | number **=** int**(**input**(**"number: "**))** |
| 02 |  |
| 03 | **def** isPrime**(**num**,** test **=** number**-**1**):** |
| 04 | '''Determines whether an integer value is prime''' |
| 05 | **try:** |
| 06 | **if** num **<=** 2 **or** test **==** 1**:** |
| 07 | **return** **True** |
| 08 | **elif** num **%** test **!=** 0**:** |
| 09 | **return** isPrime**(**num**,** test**-**1**)** |
| 10 | **return** **False** |
| 11 | **except** RecursionError**:** |
| 12 | **return** **False** |
| 13 |  |
| 14 | **print(** "%d is a prime number: %s" **%** **(**number**,** isPrime**(**number**))** **)** |

1. **Removing vowels from a given string:**

|  |  |
| --- | --- |
| Pseudocode | Time of cost |
| word <- "beautiful"  REMOVE\_VOWEL(wrd)  wrd <- String [] wrd  For i <- 0 to length(wrd):  If w[i] = "a" OR  w[i] = "e" OR  w[i] = "i" OR  w[i] = "o" OR  w[i] = "u":  Remove wrd[i]  wrd = String wrd  Return wrd  PRINT REMOVE\_VOWEL(word) | 1  1  n  n  n  1  1  1 |
| **Runtime bound:** | f(n) = 3n + 5 |
| **Complexity:** | O(n) |

**High-level language code (Python):**

|  |  |
| --- | --- |
| 01 | word **=** "beautiful" |
| 02 |  |
| 03 | **def** removeVowel**(**wrd**):** |
| 04 | wrd **=** list**(**wrd**)** |
| 05 | **for** i **in** range**(**len**(**wrd**)):** |
| 06 | **if** wrd**[**i**]** **==** "a" **or** wrd**[**i**]** **==** "e" **or** wrd**[**i**]** **==** "i" **or** wrd**[**i**]** **==** "o" **or** wrd**[**i**]** **==** "u"**:** |
| 07 | wrd**[**i**]** **=** '' |
| 08 | wrd **=** ''**.**join**(**wrd**)** |
| 09 | **return** wrd |
| 10 |  |
| 11 | **print(**removeVowel**(**word**))** |

1. **Adapting the binary search (searching by interval):**

|  |  |
| --- | --- |
| 01 | number1 **=** int**(**input**(**"1st number: "**))** |
| 02 | number2 **=** int**(**input**(**"2nd number: "**))** |
| 03 | List **=** **[**4**,** 19**,** 23**,** 36**,** 40**,** 43**,** 61**,** 64**,** 78**,** 95**]** |
| 04 |  |
| 05 | **def** binarySearch**(**num1**,** num2**,** array**):** |
| 06 | '''performs binary search to identify if there is |
| 07 | a number in the List within a given interval''' |
| 08 | mid **=** len**(**array**)//**2 |
| 09 | **try:** |
| 10 | **if** num1 **<=** num2**:** |
| 11 | **if** array**[**mid**-**1**]** **>=** num1 **and** array**[**mid**-**1**]** **<=** num2**:** |
| 12 | **return** **True** |
| 13 | **elif** array**[**mid**-**1**]** **>** num2**:** ##if the value is larger than pivot (to the right of the array) |
| 14 | **return** binarySearch**(**num1**,** num2**,** array**[:**mid**-**1**])** ##calls itself with the array halved to the right side of the pivot |
| 15 | **elif** array**[**mid**-**1**]** **<** num1**:** ##if the value is smaller than pivot (to the left of the array) |
| 16 | **return** binarySearch**(**num1**,** num2**,** array**[**mid**:])** ##calls itself with the array halved to the left side of the pivot |
| 17 | **return** **False** |
| 18 | **else:** |
| 19 | **return** "ERROR! Lower value > upper value" |
| 20 | **except** IndexError **or** RecursionError**:** ##signifying the two common errors during runtime |
| 21 | **return** **False** |
| 22 |  |
| 23 | **print(**"Is there an integer between %d and %d in the list? Answer: %s" **%** **(**number1**,** number2**,** binarySearch**(**number1**,** number2**,** List**)))** |

**[CONTINUED]**

**Table 1:**

|  |  |
| --- | --- |
| Pseudocode | Time of cost |
| number1 <- String input converted to Int  number2 <- String input converted to Int  List <- [4, 19, 23, 36, 40, 43, 61, 64, 78, 95]  binarySearch(num1, num2, array) {  midpoint <- len(array)//2  If num1 <= num2 {  If array[midpoint] ≥ num2 AND array[midpoint] ≤ num2 {  Return TRUE  }  ElseIf midpoint > num2 {  Return binarySearch(num1, num2, right half of array)  }  ElseIf midpoint < num2 {  Return binarySearch(num1, num2, left half of array)  Return FALSE  }  Else {  Return “Error! Lower value > upper value”  }  } | 1  1  1  1  1  1  1  1  n  1  n  1  1 |
| **Runtime bound:** | f(n) = 1 + 1 + n + n + n + n + n + 1 + 1  = 5n + 4 |
| **Complexity:** | O(n) |

**Note 1:**

If I specified the except statement as:

except IndexError and RecursionError

then the programme, when executed, would have first encountered an error that would be both an IndexError and RecursionError, by first identifying the IndexError and rendering it mute without the user's awareness and then proceeding to the identifying recursion error as it would also be the case.

As a solution I substituted to above statement to:

except IndexError or RecursionError

The solution is mainly based on a test run of the programme and entering a negative number for the lower range bound. The attempt was carried out with the except statement written before the amendment, as shown in the first instance, and when running the programme, the stack report had returned an IndexError regardless of the written except statement. This led to the second instance as the solution, which indicates that the programme would the code would cause either error instead of both, thus finally displaying the expected result.

**Note 2:**

Initial analysis concluded that the time taken for some of the code is constant, yet when the first condition (base case) is not met and the programme executes the code in lines 14 or 16, the function calls itself thus triggering a recursion. Due to this, the programme re-executes the code before the call until the base case is met, which arguably therefore alters the time complexity of the said code to **O(log n)**. Nevertheless, after much consideration, the worst-case scenario is taken in account so the complexity of the programme algorithm would result to O(log n) regardless.

1. **Extracting the sub-sequence of maximum length (ascending order):**

|  |  |
| --- | --- |
| 01 | array **=** **[**71**,** 41**,** 15**,** 68**,** 49**,** 9**,** 26**,** 46**,** 15**,** 53**,** 96**,** 23**,** 54**,** 17**,** 11**,** 5**]** |
| 02 |  |
| 03 | **print(**array**,** end**=**"\n\n"**)** |
| 04 |  |
| 05 | **def** subSeq**(**list1**):** |
| 06 |  |
| 07 | '''Finds and returns ordered sub-sequence within a given list''' |
| 08 |  |
| 09 | newList **=** **[]** # new list for ordered sub-sequence |
| 10 |  |
| 11 | **while** len**(**list1**)** **>=** 2**:** |
| 12 | **try:** |
| 13 | lowest **=** list1**[**0**]** # lowest value is first value in given list by default (temp) |
| 14 | **if** lowest **>** list1**[**1**]:** |
| 15 | lowest **=** list1**[**1**]** # value in second index of the list is now the lowest |
| 16 | list1 **=** list1**[**1**:]** # list sequence now starts from point of new lowest value |
| 17 | **else:** |
| 18 | **for** i **in** range**(**1**,** len**(**list1**)):** |
| 19 | **if** lowest **>** list1**[**i**]:** # checks for smaller integers than lowest value in given list |
| 20 | lowest **=** list1**[**i**]** # value in index i is now the lowest value |
| 21 | newList**.**append**(**lowest**)** # adds new lowest value to new list |
| 22 | list1 **=** list1**[**i**+**1**:]** # list now starts from index adjacent to that of lowest value |
| 23 | **for** j **in** range**(**len**(**newList**[:**i**])):** # code in for-loop deletes every value before last value of new list (lowest) if a smaller value is found |
| 24 | **if** newList**[-**1**]** **<** newList**[**j**]:** |
| 25 | **del** newList**[:**i**]** |
| 26 | **break** |
| 27 | **break** |
| 28 | **if** len**(**list1**)** **==** 2**:** |
| 29 | **if** newList**[-**1**]** **<** list1**[-**2**]** **and** newList**[-**1**]** **<** list1**[-**1**]:** |
| 30 | # checks if last value in new list is smaller than last two elements in given list; |
| 31 | # if otherwise, following statements are disregards them + returns new list |
| 32 | **if** list1**[**0**]** **<** list1**[**1**]:** # orders remaining values if size length of list is two |
| 33 | newList**.**append**(**list1**[**0**])** |
| 34 | newList**.**append**(**list1**[**1**])** |
| 35 | **else:** |
| 36 | newList**.**append**(**list1**[**1**])** |
| 37 | **break** |
| 38 | **except** IndexError**:** |
| 39 | **break** |
| 40 |  |
| 41 | **return** newList |
| 42 |  |
| 43 | **print(**subSeq**(**array**))** |