CSE525 Monsoon 2020 Ashima Garg (PhD19003)
Homework 6 Problem 1 Pramil Panjawani (PhD19008)

You CANNOT consult any other person or online resource for solving the homework problems. You can definitely ask the instructor or TAs for hints and you are encourage to do so (in fact, you will get useful hints if you ask for help at least 1-2 days before the due date). If we find you guilty of academic dishonesty, penalty will be imposed as per institute guidelines.

Solution: **Explanation:** *Scrabble* initialises max_val to -1 that stores the maximum value point obtained so far and calls *ScrabbleHeper*. In *ScrabbleHelper*, we check for all the possible words of length 3 or more, and adds the corresponding value points of that word to the cur_val obtained so far. We update the max_value param if cur_val is more than max_val We prune when the total letters remaining are less than equal to 2.

```
⟨⟨Makes the current sequence length to be 7 if there are elements in $Letter$, and remove those elements from $Letter$⟩⟩
def MakeCurSeqLen7($Letter[1...n], cur\_seq, n, Value$):
  if $n <= 0$: return $Letter, cur\_seq, n$
  $cur\_seq\_len \leftarrow len(cur\_seq)$, $i \leftarrow 1$
  while($i \leq n$ and $cur\_seq\_len \leq 7$):
      Append the new letter from $Letter$ array to $cur\_seq$ and remove that letter from $Letter$ array.
  return $Letter, cur\_seq, (n-i)$ ⟨⟨Returns the updated $Letter$ array, cur_seq, and updated length of $Letter$ array⟩⟩

def AllPossibleWords(sequence, Value):
  ⟨⟨Generates all possible valid words from a string and returns total value points corresponding to that
i.e. $Words[i]$ and $Value\_pts[i]$ represent the Word and Value point corresponding to that word⟩⟩

def Scrabble($Letter[1...n], Value[1...n], n$): ⟨⟨Main function that calls ScrabbleHelper to find max value score.⟩⟩
  if $n = 1$: return $Value[1]$
  $max\_value = -1$, $cur\_seq = []$, $cur\_value = 0$
  ⟨⟨Initializes max_value to -1 and passes reference of this variable to update it whenever next max is observed.⟩⟩
  ScrabbleHelper($Letter, Value, n, cur\_seq, \&max\_value, cur\_value$)
  return $max\_value$ ⟨⟨finally return the updated $max\_value$ obtained from ScrabbleHelper⟩⟩

def ScrabbleHelper($Letter[1...n], Value[1...n], n, cur\_seq, max\_value, cur\_value$):
  if $n \leq 0$ and $len(cur\_seq) \leq 2$: ⟨⟨If $Letter$ array is empty and $cur\_seq$ length is also less than equal to 2⟩⟩
⟨⟨Even if the two letter word is not valid, take the sum of the values of those letters as single letter will always be valid. ⟩⟩
      if $len(cur\_seq) = 2$:
          cur_value += Value[0] + Value[1]
      elif $len(cur\_seq) = 1$: ⟨⟨Single letter is always valid⟩⟩
          $cur\_value += $ Value[0]
      if $max\_value < cur\_value$:
          $max\_value \leftarrow cur\_value$ ⟨⟨Update $max\_value$ variable if another max value is found⟩⟩
      return ⟨⟨Reached the end of branch of search tree⟩⟩
  $Letter, cur\_seq, n \leftarrow$ MakeCurSeqLen7($Letter, cur\_seq, n, Value$) ⟨⟨$n$ is no of remaining elements in $Letter$⟩⟩.
  $Words, Value\_pts \leftarrow$ AllPossibleWords($cur\_seq, Value$)
  if $len(Words) = 0$ : return ⟨⟨If no valid word is found, return⟩⟩
  for($i = 1 ... len(Words)$)
      if $len(Words[i]) \geq 3$ : ⟨⟨Iteratively checks for all the possible Words if the length of the Word is 3 or more⟩⟩
          ⟨⟨Store the remaining letters after removing letters of $Word[i]$ from cur_seq in $remaining\_seq$⟩⟩
          $remaining\_seq \leftarrow cur\_seq - Words[i]$
          ⟨⟨Update the current value by adding the value corresponding to this word⟩⟩
          $cur\_value \leftarrow cur\_value + Value\_pts[i]$
          ScrabbleHelper($Letter, Value, n, remaining\_seq, max\_value, cur\_value$)
  ⟨⟨Backtrack for the next possible solution, remove the $Value\_pts[i]$ added to the $cur\_val$ for the previous $Word[i]$⟩⟩
          $cur\_value \leftarrow cur\_value - Value\_pts[i]$
```

Correctness: ScrabbleHelper returns the updated value of max_val, i.e. maximum total points. If there are no elements in the $Letter$, then it returns -1, if one element is there it returns, the value of that element, if it is 2 letter word it returns the sum of the values of those 2 elements. Whenever the entire $Letter$ gets empty and the remaining elements are true, max_value is also updated with cur_val which stores the total value points obtained in this round. Else, for all the 3 or more valid words, *ScrabbleHelper* recurs for all those words and corresponding maintains the cur_val. It removes the value points of the previous word, when it backtracks.

Complexity: For worst case time complexity we can assume we consider we are making words of length 3 except for the last case, Time Complexity: $T(n) = \binom{7}{3}T(n-3)$(Considering all the words that we make are valid of length)$+O(1)$(AllPossibleWords) $+O(1)$(MakeCurSeqLen7)(Since, at max 4 additional words will be traversed from the $Letter$ array).
$T(n) = \binom{7}{3}T(n-3) + \binom{7}{4}T(n-4) + \binom{7}{5}T(n-5) + \binom{7}{6}T(n-6) + \binom{7}{7}T(n-7) + K.O(1)$
When considering only $T(n) = \binom{7}{3}T(n-3)$ i.e. words of only 3 letters, then, $T(n) = 35^n$
Therefore, the actual time complexity will definitely be greater than this because we need to explore all the possible combinations of length 4, 5, 6, 7 as well.

When all such combinations are considered, $T(n) = O(n^n)$ ■