Part C

Focusing on the main part of the algorithm, Depth-First Search, given that n represents the dimension of the board ($n \times n$) and k represents the maximum length of a word which is limited by the chars in the alphabet (even if the alphabet expands in size), this algorithm has a time-complexity of $O(A\, n^2 8^k)$, where 8 represents the maximum number of possible adjacent cells to an individual cell. With the restraint that prevents letter duplicates though, there are fewer letter choices every time another adjacent cell has been chosen when trying to match the letters in a word. Since there are only k, letters in the alphabet, this restricts the path significantly. Whereas in the original algorithm, allowing duplicates would allow k, in essence, to be infinitely large. Thus, the restricted algorithm has a worst-case time-complexity of $O(n^2 8^{26})$ in comparison to the worst-case time-complexity of the original algorithm, $O(n^2 8^{\infty})$. This is the approach I took in Part D, which should restrict duplicate paths such that it prevents retracing with repeated letters. It disabled intermediary child nodes which have the same characters, where the starting node e.g. “O” and the destination node (grandchild node) e.g. “E” is the same. However, another approach which I didn’t manage to implement on time for PartD works by disabling cells on the board which aren’t adjacent to the next letter in a word being analysed. By going through the prefix trie using DFS, every word would be checked for. If A represents the number of words in the dictionary, this would lead to a time complexity of $O(n^2 kA)$, which is significantly better when scaling in comparison.