Q. 2. Soln

From the pseudo code, we can say that there two for loops that determine the running time of the algorithm. The first outer loop runs for n times and the inner loop runs for the W times. And, also it is the every case scenario, so has tight bound. Thus, the knapsack problem solution with dynamic programming gives us $\Theta(nW)$ running time complexity.

Q. 3. Soln

The knapsack problem can be solved in brute-force way by using some recursive strategy. In a naive approach, the solution would look like -

```
maxVal ← 0
Algorithm KPSolution (n, currWeight, currVal)

    if n==0 and currWeight <= W and currval > maxVal
            maxVal ← currVal

    if n=0
        return

    KPSolution (n-1, currWeight, currVal)
    KPSolution ( n-1 currWeight + w[n], currVal + v[n])
```

As, these are two recursive calls it has obviously $O(2^n)$ time complexity. So, DP has reduced the exponential time complexity to pseudo-polynomial time.