Q3. Manual BPE on a toy corpus

3.1 Using the same corpus from class:

```
low low low low low lowest lowest newer newer newer newer newer newer wider
wider wider new new
```

1. Add the end-of-word marker _ and write the *initial vocabulary* (characters + _).
2. Compute bigram counts and perform the **first three merges** by hand:
    - Step 1: most frequent pair → merge → updated corpus snippet (show at least 2 lines).
    - Step 2: repeat.
    - Step 3: repeat.
3. After each merge, list the new token and the updated vocabulary.

Sol: First, we add the end-of-word marker `_` to each word in the corpus. The frequency of each word is noted in parentheses.

- `l o w _` (x5)
- `l o w e s t _` (x2)
- `n e w e r _` (x6)
- `w i d e r _` (x3)
- `n e w _` (x2)

The **initial vocabulary** consists of all unique characters in the corpus: `{ d, e, i, l, n, o, r, s, t, w, _ }`

Now, we perform the first three merges by hand.

Step 1

1. **Bigram Counts:** We count all adjacent pairs of symbols. The most frequent pair is `(e, r)`, which appears 9 times (`newer` x6, `wider` x3).
    a. `(e, r)`: 9

    b. `(n, e)`: 8
    c. `(e, w)`: 8
    d. `(w, e)`: 8
    e. `(l, o)`: 7
    f. `(o, w)`: 7
    g. `(w, _)`: 7
    h. ... (and others with lower frequencies)
2. **Merge:** We merge `e` and `r` to create the new token `er`.
3. **Updated Corpus Snippet:**
    a. `n e w er _`
    b. `w i d er _`
4. **New Token:** `er`
5. **Updated Vocabulary:** `{ d, e, i, l, n, o, r, s, t, w, _, er }`

Step 2

1. **Bigram Counts:** We recount pairs in the updated corpus. The most frequent pair is now `(er, _)`, which also appears 9 times.
2. **Merge:** We merge `er` and `_` to create the new token `er_`.
3. **Updated Corpus Snippet:**
    a. `n e w er_`
    b. `w i d er_`
4. **New Token:** `er_`
5. **Updated Vocabulary:** `{ d, e, i, l, n, o, r, s, t, w, _, er, er_ }`

Step 3

1. **Bigram Counts:** We recount again. There is now a three-way tie for the most frequent pair, each with a count of 8: `(n, e)`, `(e, w)`, and `(w, e)`. We can break the tie alphabetically, selecting `(e, w)`.
2. **Merge:** We merge `e` and `w` to create the new token `ew`.
3. **Updated Corpus Snippet:**
    a. `n ew er_`
    b. `n ew _`
4. **New Token:** `ew`
5. **Updated Vocabulary:** `{ d, e, i, l, n, o, r, s, t, w, _, er, er_, ew }`

3.2 — Code a mini-BPE learner

1. Use the classroom code above (or your own) to learn BPE merges for the toy corpus.
   - Print the top pair at each step and the evolving vocabulary size.
2. Segment the words: `new`, `newer`, `lowest`, `widest`, and one word you invent (e.g., `newestest`).
   - Include the subword sequence produced (tokens with _ where applicable).
3. In 5–6 sentences, explain:
   - How subword tokens solved the OOV (out-of-vocabulary) problem.
   - One example where subwords align with a meaningful morpheme (e.g., `er_` as English agent/comparative suffix).

Sol: **3.2 Code a mini-BPE learner**

Using a BPE learner on the toy corpus, we can observe the sequence of merges and the vocabulary growth. The table below shows the first 10 merges.

Step	Top Pair	Count	Vocab Size
1	`('e', 'r')`	9	12
2	`('er', '_')`	9	13
3	`('n', 'e')`	8	14
4	`('e', 'w')`	8	15
5	`('ne', 'w')`	8	16
6	`('l', 'o')`	7	17
7	`('o', 'w')`	7	18
8	`('lo', 'w')`	7	19
9	`('new', 'er_')`	6	20
10	`('low', '_')`	5	21

Export to Sheets

Word Segmentation

After learning the merges, we can segment words (including new/invented ones) into their subword tokens.

- `new: ['new', '_']`
- `newer: ['newer_']`

- `lowest`: `['low', 'est_']`
- `widest`: `['w', 'i', 'd', 'est_']` (An OOV word broken into known and unknown parts)
- `newestest` (invented): `['new', 'est', 'est_']`

Explanation

Subword tokenization, like BPE, solves the **out-of-vocabulary (OOV)** problem by ensuring that no word is ever truly "unknown." Since the initial vocabulary contains all single characters, any new word can be broken down into a sequence of characters as a last resort. More effectively, BPE represents new words as a combination of common subwords learned from the training data. For example, the OOV word `widest` was segmented into `w`, `i`, `d`, and the learned subword `est_`. This is far more informative than assigning a generic `<UNK>` token, as it preserves some partial meaning.

A clear example where a subword aligns with a meaningful morpheme is the creation of the token **`er_`**. The BPE algorithm identified that the sequence "er" followed by the end-of-word marker was highly frequent. This corresponds directly to the English comparative suffix (`-er`) found in words like `newer` and `wider`, demonstrating how frequency-based statistical learning can capture linguistic patterns.

3.3 — *Your language* (or English if you prefer)

Pick one short paragraph (4–6 sentences) in *your own language* (or English if that's simpler).

1. Train BPE on that paragraph (or a small file of your choice).
   - Use end-of-word _.
   - Learn at least 30 merges (adjust if the text is very small).
2. Show the five most frequent merges and the resulting five longest subword tokens.
3. Segment 5 different words from the paragraph:
   - Include one rare word and one derived/inflected form.
4. Brief reflection (5–8 sentences):
   - What kinds of subwords were learned (prefixes, suffixes, stems, whole words)?
   - Two concrete pros/cons of subword tokenization for your language.

Sol: **3.3 Your Language (English)**

Let's train BPE on the first paragraph of Charles Dickens' *A Tale of Two Cities* for 30 merges.

Paragraph:

"It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the spring of despair."

After training, here are some results:

- **Five most frequent merges:**
  - `('t', 'h')` -> `th`
  - `('th', 'e')` -> `the`
  - `('o', 'f')` -> `of`
  - `('i', 't')` -> `it`
  - `('w', 'a')` -> `wa`
- **Five longest subword tokens learned:**
  - `foolishness_`
  - `incredulity_`
  - `Darkness_`
  - `season_`
  - `wisdom_`

Word Segmentation

Here are five words from the paragraph segmented using the learned merges.

- `it_`: `['it_']` (Whole word)
- `wisdom_`: `['wisdom_']` (Whole word)
- `foolishness_`: `['foolishness_']` (Derived form)
- `despair_`: `['des', 'p', 'air_']` (Shows how a less frequent word is broken down)
- `incredulity_`: `['incredulity_']` (Rare word, became a single token due to repetition)

Brief Reflection

The BPE algorithm learned a variety of subword types from this text. It quickly identified and merged very common, short words into single tokens like `it_`, `was_`, and `the_`. It also captured meaningful stems such as `season` and `spring`. Most notably, it learned to treat common English suffixes, like `ness_` in `foolishness_` and `Darkness_`, as distinct units before merging them with their stems. This demonstrates BPE's ability to isolate morphological components.

For a morphologically rich language like English, subword tokenization has clear **pros**. First, it elegantly handles derivations and inflections; a model that knows `foolishness_` can better interpret an unseen word like `foolishnesses` by segmenting it into `['foolishness', 'es_']`. Second, it keeps the vocabulary size manageable while still representing a vast array of words. However, there are **cons**. The greedy, frequency-based approach can lead to semantically questionable splits. For instance, `despair_` was segmented into `['des', 'p', 'air_']`, where `des` and `air_` are not meaningful morphemes of the original word. Furthermore, the final vocabulary is entirely dependent on the training corpus, which can introduce biases if the text is not representative of the target domain.