1. Scaled Dot-Product Attention

```python
def scaled_dot_product_attention(q, k, v, key_padding_mask=None, causal=False):
    d_head = q.size(-1)
    s = (einops.einsum(q, k, "n tl dh, n sl dh -> n tl sl")
         / d_head ** 0.5)

    if key_padding_mask is not None:
        s = s.masked_fill(
            key_padding_mask.unsqueeze(1).to(torch.bool),
            float("-inf")
        )

    if causal:
        attn_mask = future_mask[:s.size(1), :s.size(2)].to(s)
        a = F.softmax(s, dim=-1, dtype=torch.float32).type_as(s)

    return einops.einsum(a, v, "n tl sl, n sl dh -> n tl dh")
```

(a) In scaled-dot product attention, why do we divide pre-softmax attention scores by $\sqrt{d_{\text{head}}}$ (line 5), and what would be the consequence of not doing so. Prove your arguments mathematically, assuming the input tensor elements are i.i.d. and have a mean of 0 and standard deviation of 1?
2. Multi-head Attention

```python
def forward(self, q, k, v, key_padding_mask=None, causal=False):
    q = self.q_proj(q)
    k = self.k_proj(k)
    v = self.v_proj(v)
    q = einops.rearrange(q, "b tl (nh dh) -> (b nh) tl dh", nh=self.n_heads)
    k = einops.rearrange(k, "b sl (nh dh) -> (b nh) sl dh", nh=self.n_heads)
    v = einops.rearrange(v, "b sl (nh dh) -> (b nh) sl dh", nh=self.n_heads)
    if key_padding_mask is not None:
        key_padding_mask = einops.repeat(
            key_padding_mask, "b sl -> (b nh) sl", nh=self.n_heads)
    o = scaled_dot_product_attention(q, k, v, key_padding_mask, causal)
    o = einops.rearrange(o, "(b nh) tl dh -> b tl (nh dh)", nh=self.n_heads)
    return self.o_proj(o)
```

(a) Let’s review the rationale behind multi-head attention. Given that softmax typically exhibits unimodal behavior, it can be approximated by argmax attention. Determine the receptive field size of a node at layer $n$ for the following scenarios:

(i) With a single head.
(ii) With two heads.
(iii) With $k$ heads.

(b) In NLP, a batch of sentences typically contains sequences of varying lengths, requiring padding to match the longest sentence. To prevent these pad tokens from affecting computation, we apply key padding masks and causal masks to attentions. Describe how these masks are applied in each of the following scenarios (applied to which multi-head attention modules in which Transformer stack):

(i) Transformer encoder (e.g., BERT) trained for text classification.
(ii) Transformer decoder (e.g., GPT-3) trained for sequence generation.
(iii) Transformer encoder-decoder (e.g., T5) trained for machine translation.

(c) Determine the asymptotic time complexity of multi-head attention as a function of key/value length $n_s$, query length $n_t$, head dimension $d_{head}$, and the number of heads $h$. Ignore key padding masks and causal masks.

(d) Based on your analysis, identify the computational efficiency bottleneck for the following scenarios:

(i) When $d_{model}$ is large but sequences are short.
(ii) When sequences are long but $d_{model}$ is small.

3. Layer Normalization

Examine the Transformer diagram, which includes an “add and norm” layer after each multi-head attention or feed-forward module. The “add” represents a residual connection, inspired by ResNet. This question serves as a review of layer normalization.
(a) Consider an input tensor $X$ of shape $[B, D]$, where $B$ is the batch size and $D$ is the hidden state dimension. Layer normalization is applied to obtain output tensor $Y$ with the same shape. For an input element $x_{i,j} \in \mathbb{R}$ and its corresponding output $y_{i,j} \in \mathbb{R}$, determine which the value of $y_{i,j}$ depends on (select all that apply):

(i) $x_{i,j}$  
(ii) $x_{i',j}$ where $i \neq i'$  
(iii) $x_{i,j'}$ where $j \neq j'$  
(iv) $x_{i',j'}$ where $i \neq i'$ and $j \neq j'$  

Repeat the same analysis for batch normalization.

(b) **Prove the following:** Given an input vector $x \in \mathbb{R}^d$ and applying layer normalization with scale $\gamma$, bias $\beta$, and $\epsilon = 0$, the output $y$ satisfies

$$||y - \beta 1||_2 = \gamma \sqrt{d}$$