# 309.https://stackoverflow.com/questions/71708142/gas-efficiency-of-totalsupply-vs-a-tokenid-counter-erc-721

**T:**Gas efficiency of totalSupply() vs. a tokenID counter | ERC-721

**Q:**I'm creating a solidity contract for an NFT and in my mint function I'm not sure if a call to totalSupply() vs using a token counter and incrementing it is better practice. Does either variation cost more gas? Is one the more standard practice? I've seen examples of both being used.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Variation 1:  
  
contract MyNFT is ERC721Enumerable, PaymentSplitter, Ownable { using Counters for Counters.Counter; Counters.Counter private currentTokenId;...function mint(uint256 \_count) public payable{ uint256 tokenId = currentTokenId.current(); require(tokenId < MAX\_SUPPLY, "Max supply reached"); for(uint i = 0; i < \_count; ++i){ currentTokenId.increment(); uint256 newItemId = currentTokenId.current(); \_safeMint(msg.sender, newItemId); }}}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Variation 2:  
  
function mint(uint256 \_count) public payable{ uint supply = totalSupply(); require( supply + \_count <= MAX\_SUPPLY, "Exceeds max supply." ); for(uint i = 0; i < \_count; ++i){ \_safeMint(msg.sender, supply + i); }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Both versions seem to work. I just want to be sure I'm using the most efficient / secure. Thanks for any advice!

**C1:**Remix show the gas used when a function is called, just compare the gas usage. Similarly there plugins to measure gas consumption for Truffle and others tools.

**C2:**Looping is not so efficient solution. Without whole contract code base is hard to predict efficiency. You can try to analyze assembly of this contract

**C3:**Thanks for the advice. I just started using remix and it's really helpful

1 **Answer**

**A1:**First off all, you need to show us the underlying implementations. However, I can speculate that these are unmodified openzeppelin implementations for ERC721Enumerable and Counters.  
  
For your case only, using Counter seems a little bit pointless to me.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
 ● It increases your deployment costs(just a little bit) because of redundant code coming from Counter library  
  
 ● You already know the length of your tokens array, why keep it twice? Counters is created for situations where you don't know the number of elements, like a mapping.  
  
I am not guaranteeing correctness of the following analysis  
  
Calling totalSupply (looking from opcode point of view) will:  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
 ● jump to totalsupply (8 gas)  
  
 ● sload tokens.slot (200) gas  
  
However, while using Counter, you sstore (>= 5000 gas) each time you decrement and sload (200 gas) each time you read.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
As long as i am not mistaken about Counter using storage, and therefore sstore and sload opcodes, second variant will use much less gas.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]

**C1:**appreciate the answer, yes I'm using the generic openzeppelin implementations. This makes sense

**C2:**This means totalSupply() is more efficient right