Element Uniqueness

→ given single sequence S with n elements and asked if they're distinct

→ first solution uses iterative algorithm

```
def unique1(S):
  for j in range(len(S)):
    for k in range(j+1, len(S)):
      if S[j] == S[k]:
        return False
  return True
```

→ solves element uniqueness by looping thru distinct pairs of indices $j<k$

→ uses 2 nested for loops, where first iter of outer loop has n-1 iterations of inner loop, and second iter of outer loop causes n-2 iters of inner loop

→ thus, worst case is $(n-1)+(n-2)+\ldots+2+1$ which is $O(n^2)$

Sorting as Problem-Solving Tool

→ by sorting the sequence, we are guaranteed that any duplicates will be next to each other

→ thus, we only need to do a single pass

```
def unique2(S):
  temp = sorted(S)                   → worst case runs at O(nlogn)
  for j in range(1, len(temp)):      → once data is sorted, the loop runs in O(n)
    if S[j-1] == S[j]:                 time, so the entire algorithm runs in
      return False                              O(nlogn)
  return True
```

Justification Techniques

→ By Example: to justify, we only need to produce one counterexample to prove its false

→ Contrapositive: to prove using the opposite

→ if p is true, then q is true ⟶ if q is not true, p is not true

→ contradiction: establish a statement q is true by first supposing that q is false and showing this assumption leads to a contradiction