

# Data Copy with Reduced Control Checks

## 1. Problem

The problem is framed around a simulated execution environment with strict instruction timing. In such an environment, the dominant cost is not memory access but **control decisions**.

Checking a loop condition for every element introduces unacceptable overhead. Therefore, the program must:

- Minimize control checks
- Group operations into fixed-size blocks
- Handle misalignment without compact loops
- Perform only explicit element-by-element assignments

## 2. v1 to v4

Several alternative implementations (v1 - v4) appear reasonable at first glance:

- v1 allocates memory but performs no transfer

```
def fastcopy(source,count):
    dest=list("#"*count) #Allocates a destination structure large
    enough to hold count elements.
    return dest
```

- v2 uses slicing for partial copies

```

def fastcopy(source,count):
    dest=list("#"*count) #Allocates a destination structure large
    enough to hold count elements.
    groups=count//8 #Determines how many complete transfer groups
    of eight elements are required.
    rem=count%8
    if rem:
        dest[:rem]=source[:rem]#Performs any necessary initial
        element transfers using structured control flow rather than
        iteration.

    return dest

```

- v3 and v4 use slicing inside loops

```

#v3

def fastcopy(source,count):
    #Allocates a destination structure large enough to hold count
    elements.
    dest=list("#"*count)
    #Determines how many complete transfer groups of eight
    elements are required.
    groups=count//8
    #Performs any necessary initial element transfers using
    structured control flow rather than iteration.
    rem=count%8
    if rem:
        dest[:rem]=source[:rem]
    last=rem
    #Completes the remaining transfers in a repetitive block that
    assigns exactly eight elements per iteration.
    for _ in range(groups):
        buffer=last+8
        dest[last:buffer]=source[last:buffer]
        last=buffer
    return dest #Returns the destination structure containing the
    copied elements.

#SAMPLE IO
source=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]
fastcopy(source,len(source))

```

```
#v4

def fast_copy(source, count):
    # Allocate destination buffer
    dest = [None] * count

    # Number of full 8-element groups
    groups = count // 8
    rem = count % 8

    idx = 0

        # Handle remaining elements first (no loop, structured
        control)
    if rem:
        dest[0:rem] = source[0:rem]
        idx = rem

    # Copy remaining elements in fixed groups of 8
    for _ in range(groups):
        dest[idx:idx + 8] = source[idx:idx + 8]
        idx += 8

    return dest

#SAMPLE IO
source=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]
fastcopy(source,len(source))
```

These solutions violate the constraints because slicing is a **bulk operation implemented in C**. It hides loops, branching, and per-element behavior.

If Python Tutor cannot show the element-by-element execution, then the execution was not explicitly expressed.

This problem demands that *every move be visible*.

### 3. last.py

The correct solution treats the program like a tiny machine:

- Decide how many full 8-element groups exist
- Handle leftover elements first

- Enter a loop where each iteration performs exactly eight assignments
- Perform one loop check per eight transfers

The program controls *where execution begins* instead of *how many times it repeats*.

## Code:

```
def fast_copy(source, count):
    dest = [None] * count

    groups = count // 8
    rem = count % 8
    idx = 0

    if rem >= 1:
        dest[idx] = source[idx]; idx += 1
    if rem >= 2:
        dest[idx] = source[idx]; idx += 1
    if rem >= 3:
        dest[idx] = source[idx]; idx += 1
    if rem >= 4:
        dest[idx] = source[idx]; idx += 1
    if rem >= 5:
        dest[idx] = source[idx]; idx += 1
    if rem >= 6:
        dest[idx] = source[idx]; idx += 1
    if rem >= 7:
        dest[idx] = source[idx]; idx += 1

    for _ in range(groups):
        dest[idx] = source[idx]
        dest[idx + 1] = source[idx + 1]
        dest[idx + 2] = source[idx + 2]
        dest[idx + 3] = source[idx + 3]
        dest[idx + 4] = source[idx + 4]
        dest[idx + 5] = source[idx + 5]
        dest[idx + 6] = source[idx + 6]
        dest[idx + 7] = source[idx + 7]
        idx += 8
```

```
    return dest
```

## Execution:

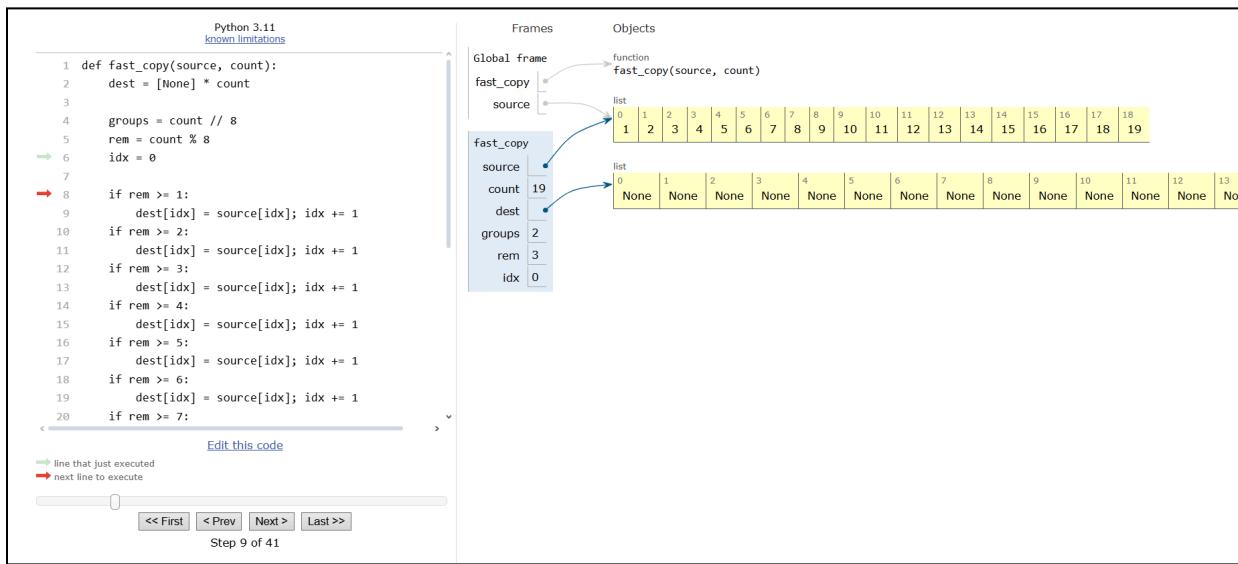
### Stage 1 - Allocation and Planning

Lines executed:

- `dest = [None] * count`
- `groups = count // 8`
- `rem = count % 8`
- `idx = 0`

At this point:

- `dest` is allocated but empty
- `groups` represents full instruction blocks
- `rem` represents the partial block
- `idx` is the instruction pointer

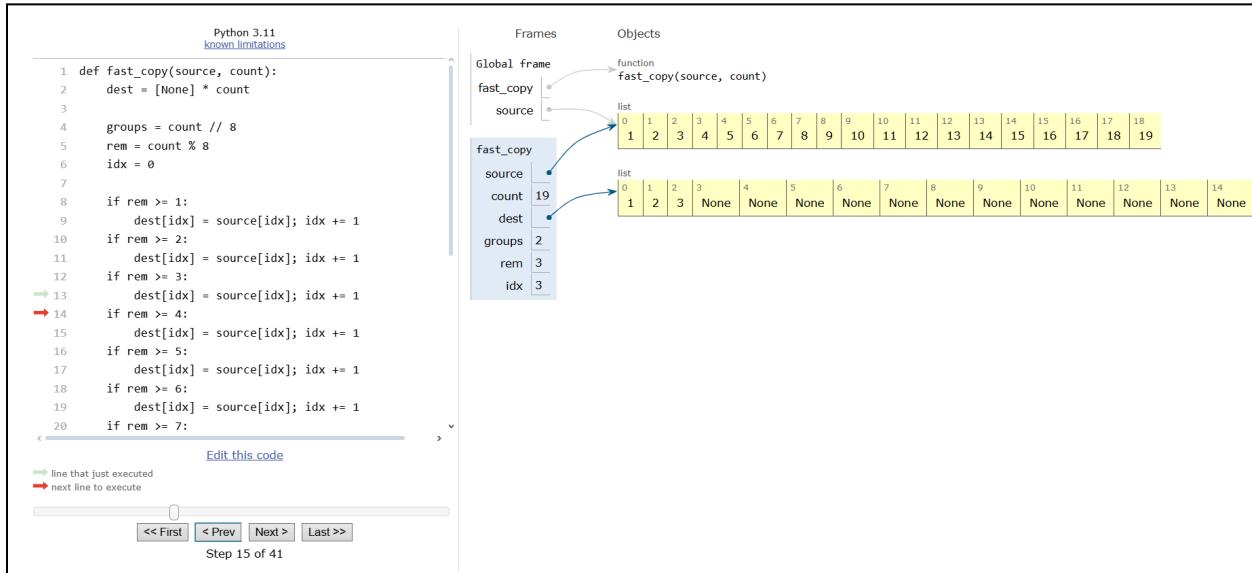


### Stage 2 - Partial Group Handling

The `if rem >= N` sequence is not a loop. It is a **manual fall-through structure**.

Only the required number of assignments execute. There is no counting loop and no repeated condition check per element.

This simulates jumping into the middle of an unrolled instruction block.



## Stage 3 - Fixed-Width Repeating Block

The loop that follows is the repeating portion of the routine.

Inside the loop:

- Exactly eight assignments occur
- No branching or decisions exist
- Control overhead happens once per eight transfers

This is the performance-critical structure.

```

Python 3.11
known_limitations

13     dest[idx] = source[idx]; idx += 1
14 if rem >= 4:
15     dest[idx] = source[idx]; idx += 1
16 if rem >= 5:
17     dest[idx] = source[idx]; idx += 1
18 if rem >= 6:
19     dest[idx] = source[idx]; idx += 1
20 if rem >= 7:
21     dest[idx] = source[idx]; idx += 1
22
23 for _ in range(groups):
24     dest[idx] = source[idx]
25     dest[idx + 1] = source[idx + 1]
26     dest[idx + 2] = source[idx + 2]
27     dest[idx + 3] = source[idx + 3]
28     dest[idx + 4] = source[idx + 4]
29     dest[idx + 5] = source[idx + 5]
30     dest[idx + 6] = source[idx + 6]
31     dest[idx + 7] = source[idx + 7]
32     idx += 8
33
34 return dest
35
36 source=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]
37 fast_copy(source,len(source))

```

line that just executed  
next line to execute

<< First < Prev Next > >>

Step 29 of 41

## Stage 4 - Completion

When the loop finishes:

- `idx == count`
- All elements have been copied
- No extra checks were performed

```

Python 3.11
known_limitations

18 if rem >= 6:
19     dest[idx] = source[idx]; idx += 1
20 if rem >= 7:
21     dest[idx] = source[idx]; idx += 1
22
23 for _ in range(groups):
24     dest[idx] = source[idx]
25     dest[idx + 1] = source[idx + 1]
26     dest[idx + 2] = source[idx + 2]
27     dest[idx + 3] = source[idx + 3]
28     dest[idx + 4] = source[idx + 4]
29     dest[idx + 5] = source[idx + 5]
30     dest[idx + 6] = source[idx + 6]
31     dest[idx + 7] = source[idx + 7]
32     idx += 8
33
34 return dest
35
36 source=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]
37 fast_copy(source,len(source))

```

line that just executed  
next line to execute

<< First < Prev Next > >>

Step 41 of 41

This implementation satisfies all constraints:

- Control checks are amortized across 8 operations
- Transfers occur in fixed-width groups

- Partial groups are handled without compact loops
- Every assignment is explicit and visible
- Execution order is deterministic

