Algos can be measured in terms of:

1) Correctness - whether they perform a specific task

2) Computational resource consumption - how much memory & CPU time an algo needs

3) Ease of understanding - how difficult is it for someone else to understand your algo

Processing & memory resources

1) Processing requirements - measured in # of operations that the CPU must carry out

2) Memory requirements - measured in terms of memory positions needed during execution

Ways to measure time & space requirements

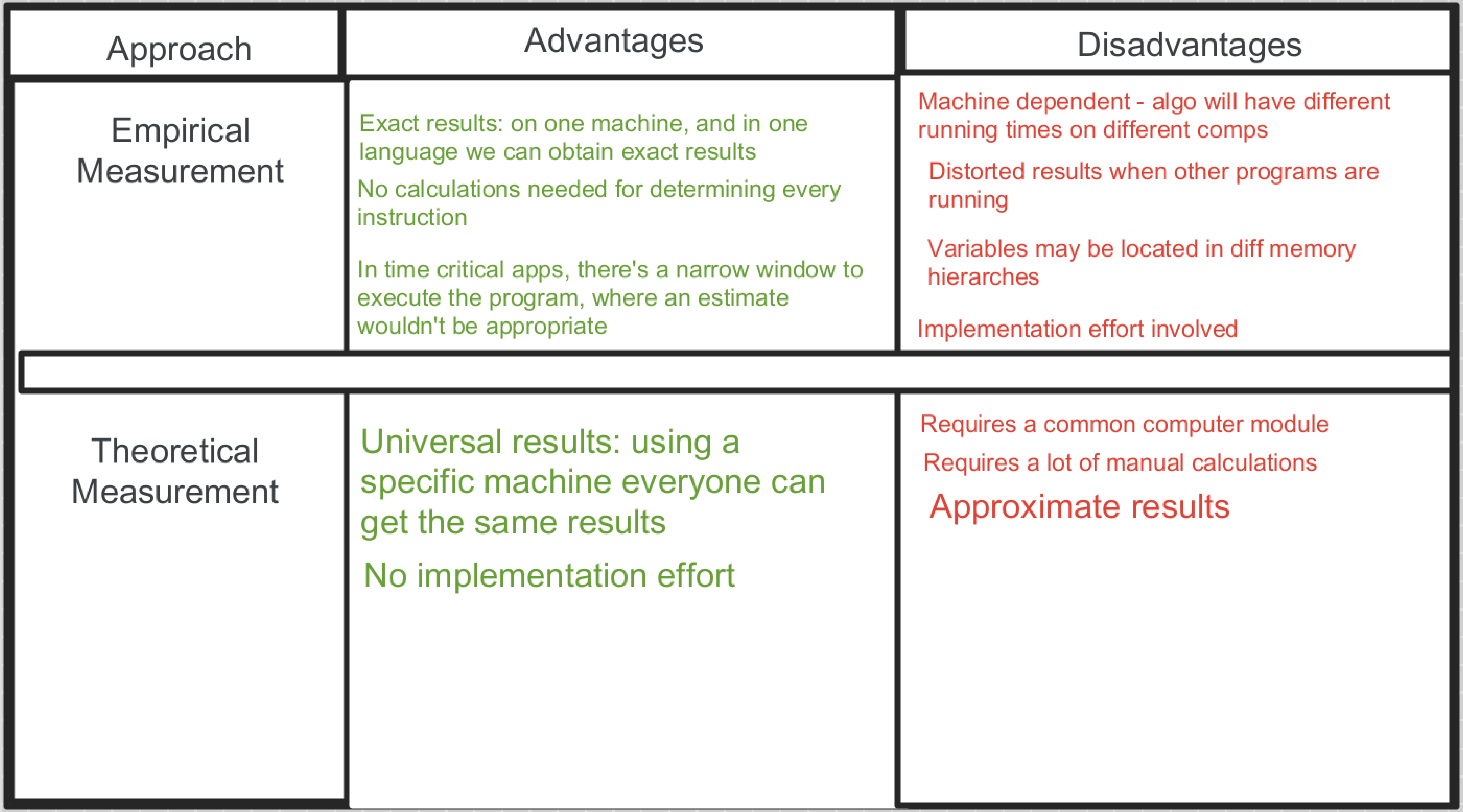
1) Empirical measurement: an implemenatation of an algo in a specific language, run on a specific machine. The execution time is measured.

2) Theoretical measurement: In terms of time requirements: make assumptions about the # of CPU operations to algo performs. Multiply that by the time requiredby each operation.

In terms of space requirements: Examine all new vars created during execution of algos and calculate how much memory they use.

Analyse the algorithm assuming it's run on the model

Use a computer model (define generic computer with with specific capacities)

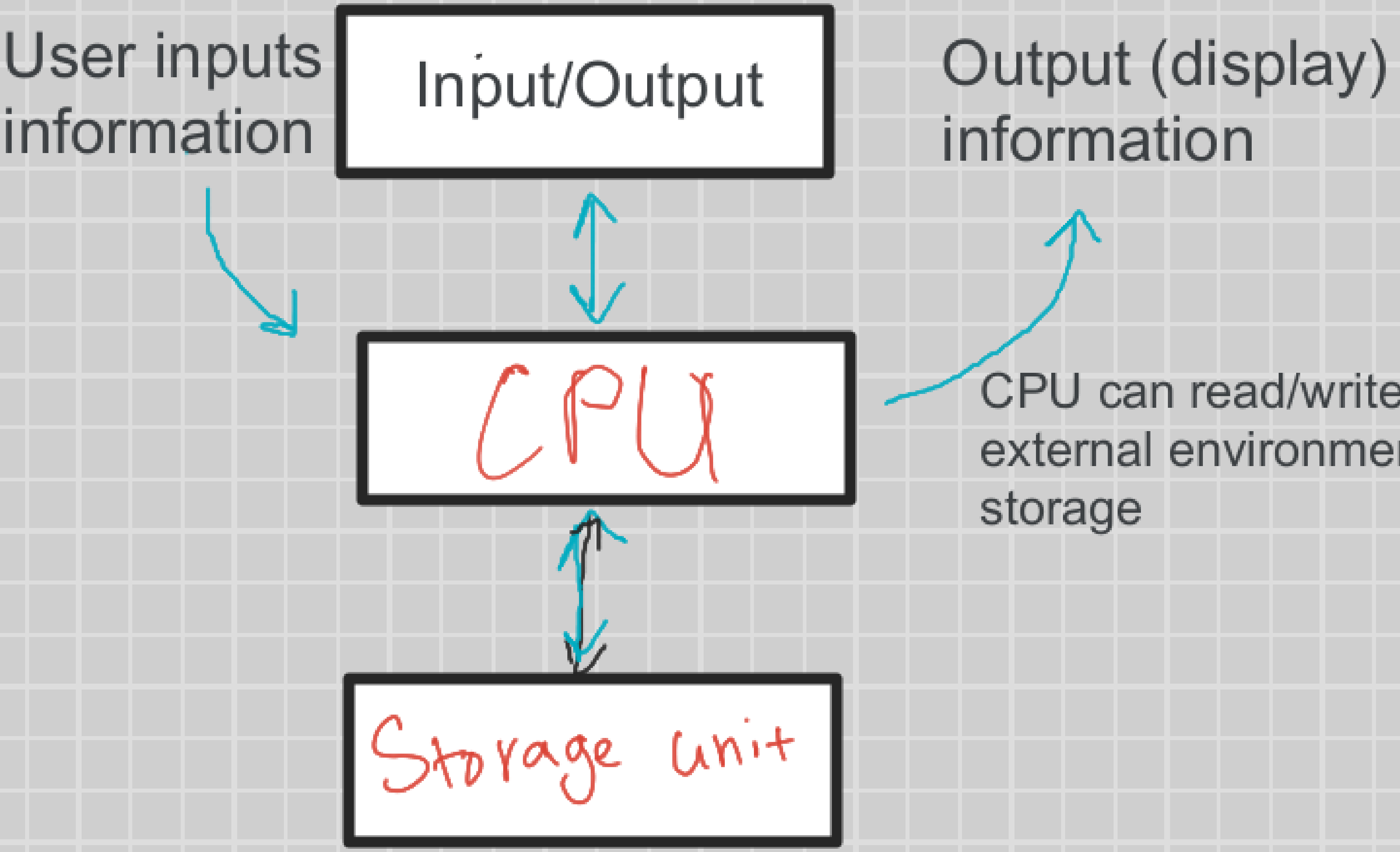


Approach	Advantages	Disadvantages
Empirical Measurement	Exact results: on one machine, and in one language we can obtain exact results No calculations needed for determining every instruction In time critical apps, there's a narrow window to execute the program, where an estimate wouldn't be appropriate	Machine dependent - algo will have different running times on different comps Distorted results when other programs are running Variables may be located in diff memory hierarches Implementation effort involved
Theoretical Measurement	Universal results: using a specific machine everyone can get the same results No implementation effort	Requires a common computer module Requires a lot of manual calculations Approximate results

RAM Model

A model that a simplified representation of computer

A model has several assumptions that it operates upon. It doesn't always need to deal with detail, however, it must be complete enough to model the aspects we need to analyse.



User inputs information

Input/Output

Output (display) information

CPU can read/write data from an external environment. Has access to storage

Running time assumptions:

Single CPU: Instructions are executed sequentially. No concurrency.

1 simple operation = 1 time unit

Simple = numerical ops, e.g. - + * / floor, ceiling

Also control ops e.g. conditions, if-else, calling function, read/ write, return, load, store, copy

Loops and functions are not simple operations but collections of several

No memory hierarchy: Access time to the storage is always a constant equalled to 1 time unit

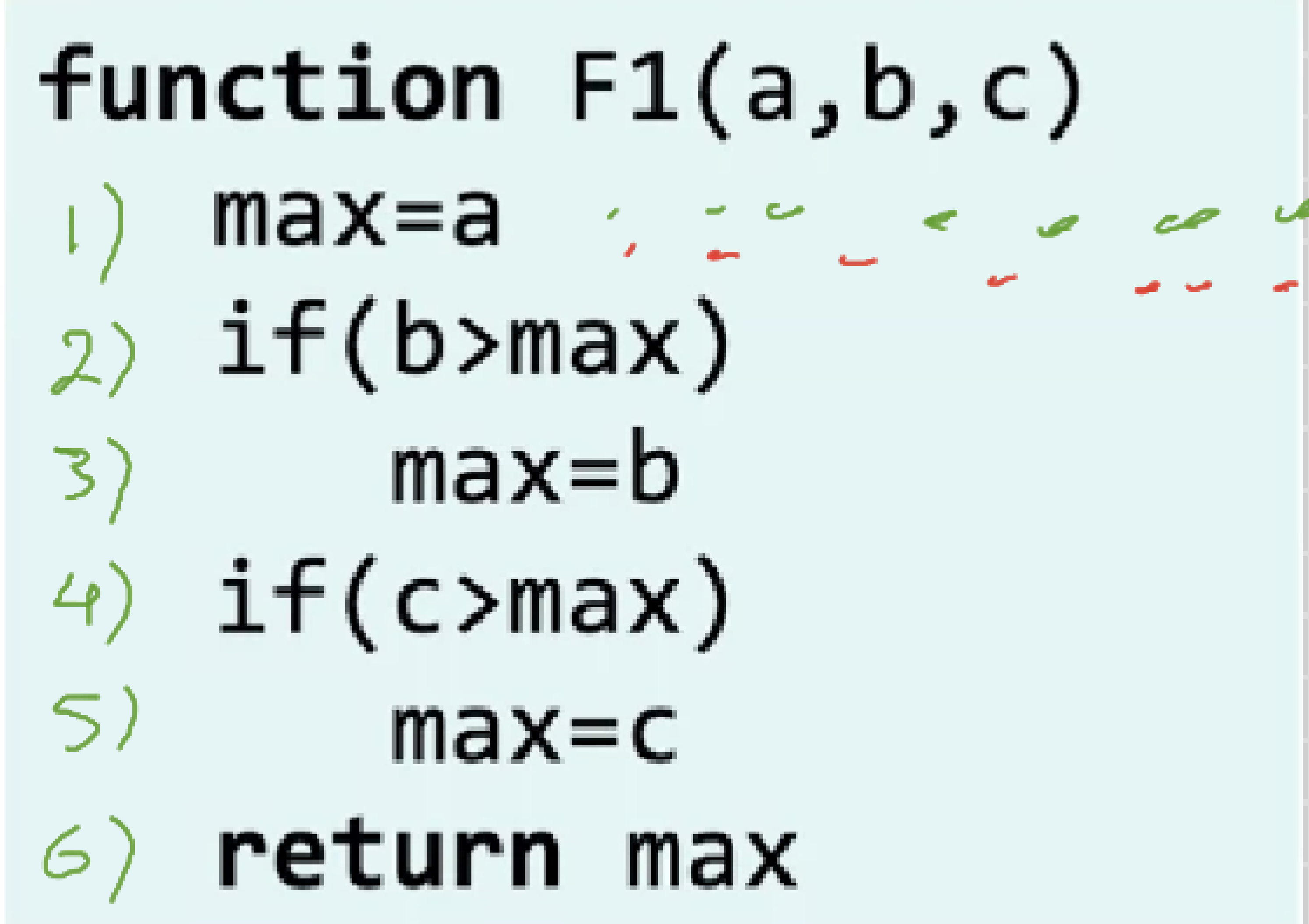
Memory assumption:

Every simple variable uses 1 memory position, no matter its type.

An array of length N uses N memory positions

We assume that memory is unlimited, and we always have as much as we need to execute.

Counting time & space units



```
function F1(a,b,c)
1)  max=a
2)  if(b>max)
3)      max=b
4)  if(c>max)
5)      max=c
6)  return max
```

2 time units (1 mem read(a), 1 mem write(max)

1 space unit (max)

2) 4 time units. 1 if, 2 mem read (b, max), 1 compare

3) 2 time units 1 mem read, 1 mem write

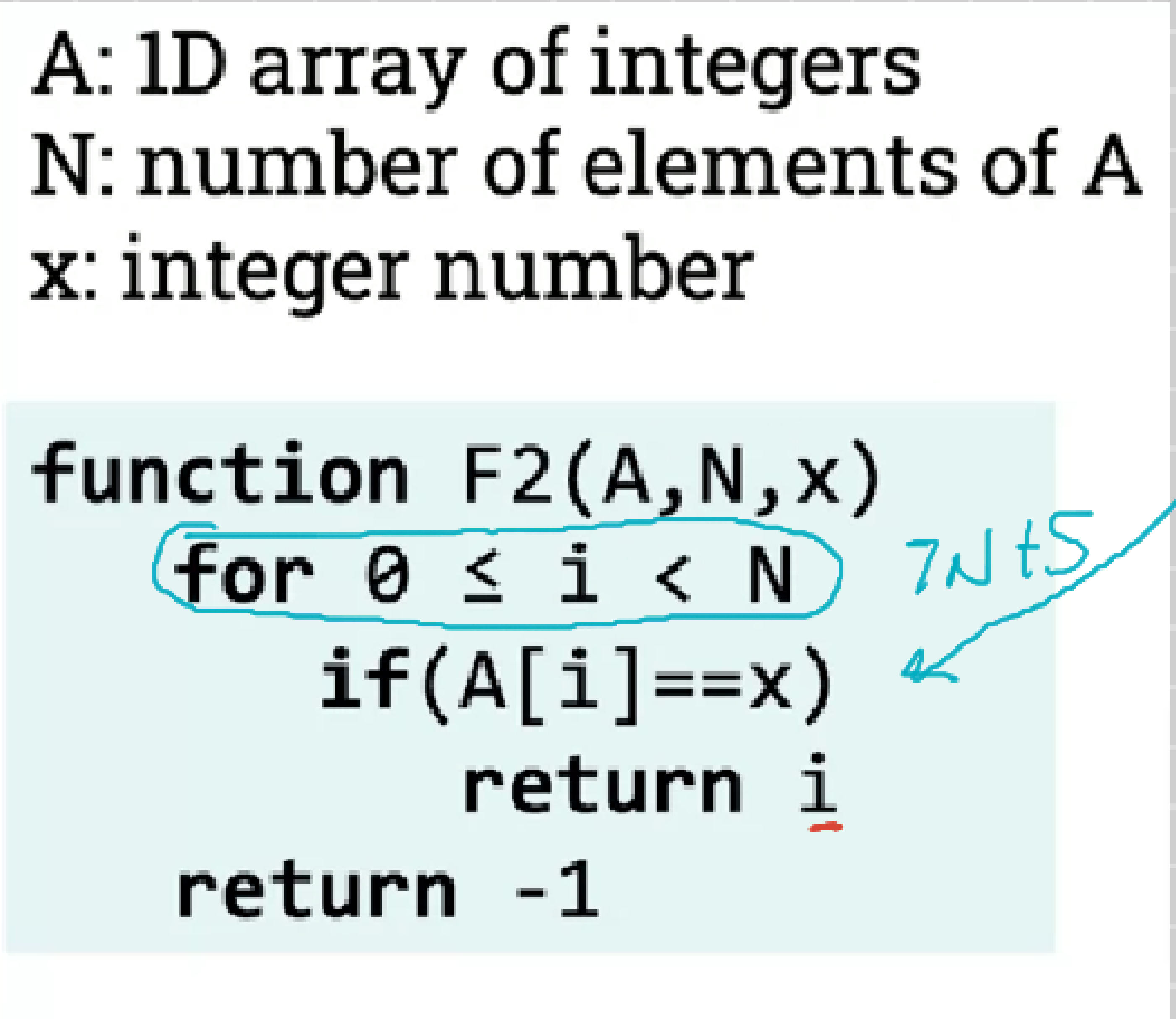
4) 4 time units

5) 2 time units

6) 2 time units

To find the total execution time, we simply add up all time units. Separately add all space units.

This algo takes 16 time units and 1 space unit



A: 1D array of integers
N: number of elements of A
x: integer number

```
1  function F2(A,N,x)
2      for 0 ≤ i < N        7N+5
3          if(A[i]==x)
4              return i
5      return -1
```

for loop:

i=0 : 1 time unit to write, 1 space unit

if(i<N) : 2 mem read, 1 compare, 1 if repeated 4(N+1)

<instruction>

i=i+1 : 1 mem read, 1 mem write, 1 numerical } 3N

N = 3 assumption

0	1	2

executed N+1 times

i	(i<N)	i=i+1
0	T	1
1	T	2
2	T	3
3	F	

2) $4(N+1) + 3N + 1$
$= 4N+4 + 3N + 1$
$= 7N+5$

3) 3 mem read
1 numerical } 5N
1 if
(5 operation, N times)

4) 1 read (2ops)
1 return

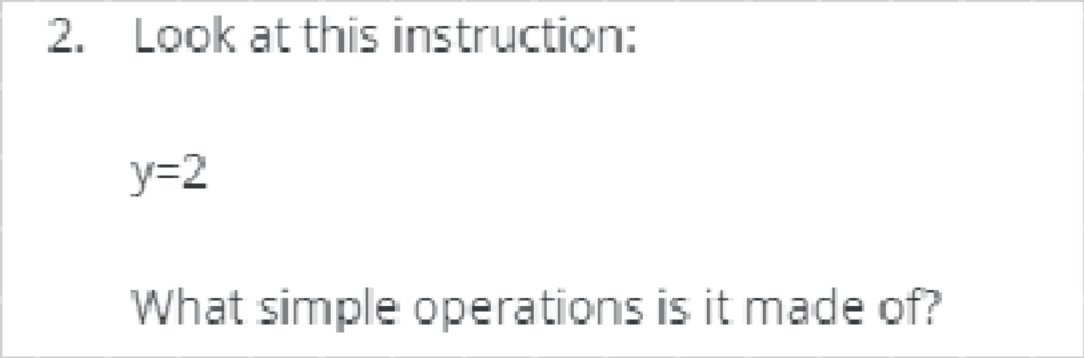
or

5) 1 return

} only 1 line is executed

assuming line 5 runs

$7N+5 + 5N + 1 = 12N+6$ time units

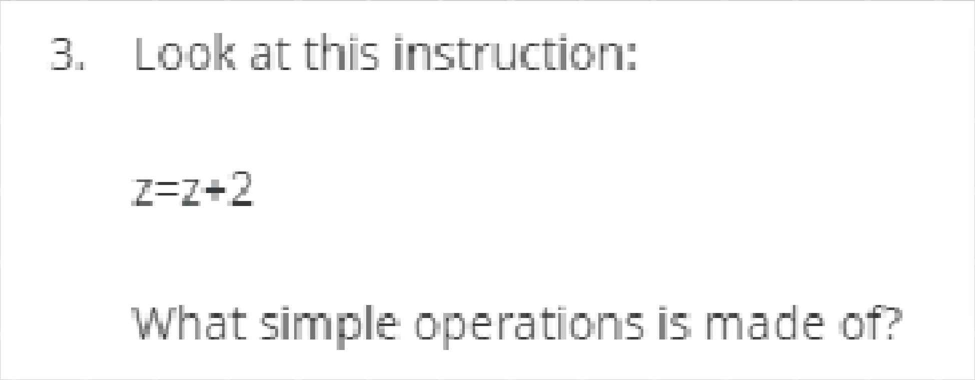


2. Look at this instruction:

   y=2

   What simple operations is it made of?

1 write operation

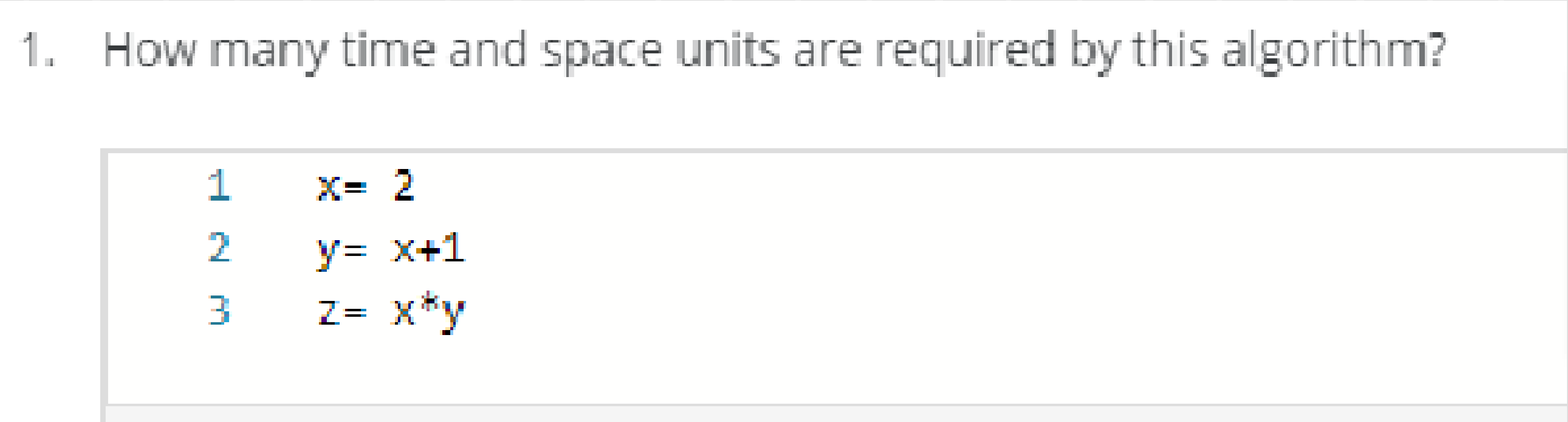


3. Look at this instruction:

   z=z+2

   What simple operations is made of?

1 read
1 numerical (+)
1 write



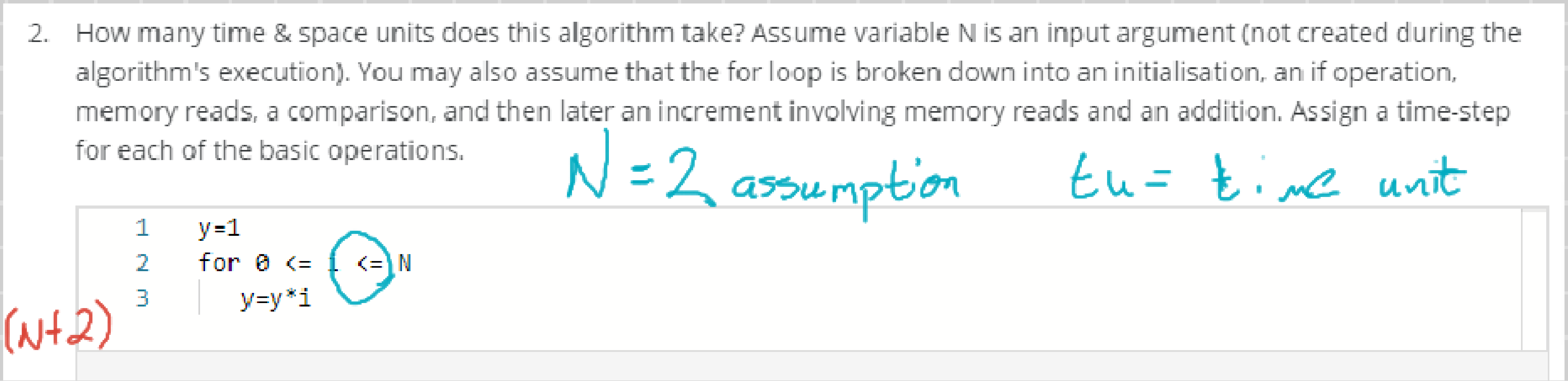
1. How many time and space units are required by this algorithm?

```
1    x= 2
2    y= x+1
3    z= x*y
```

1) 1 write | 1 space
2) 1 read, 1 num(+1), 1 write | 1 space
3) 2 reads(x,y), 1 num (*), 1 write | 1 space

3 space units

8 time units



2. How many time & space units does this algorithm take? Assume variable N is an input argument (not created during the algorithm's execution). You may also assume that the for loop is broken down into an initialisation, an if operation, memory reads, a comparison, and then later an increment involving memory reads and an addition. Assign a time-step for each of the basic operations.

N=2 assumption   tu= time unit

```
y=1
for 0 <= i <= N
    y=y*i
```

2) y=1   1 mem, 1 write   1

for loop:

i=0   1 tu

if(i≤N) 2 reads, 1 num, 1 if   4(N+2)

<instructions>   4(N+1)

i=i+1   3(N+1)

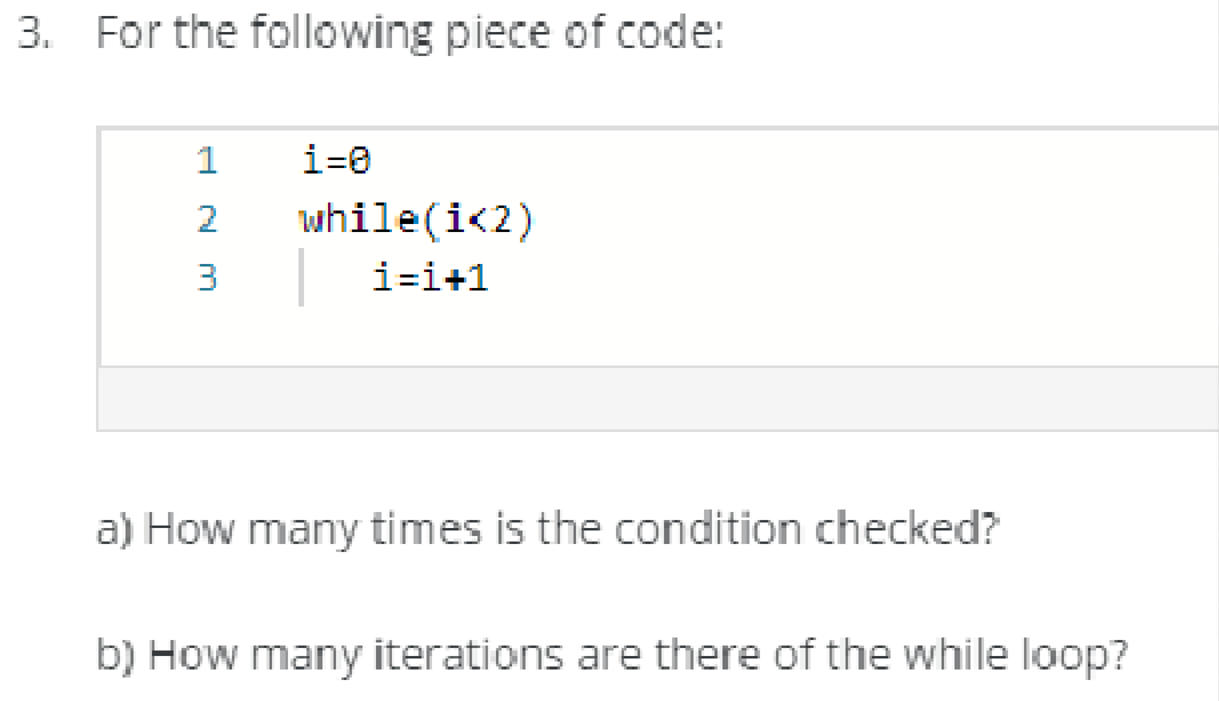
y=y*i   4 time units

i	i≤N	instructions	i=i+1
0	T	✓	1
1	T	✓	2
2	T	✓	3
3	F	✗	
4(N+2)		4(N+1)	3(N+1)

$1+1 + 4(N+2) + 4(N+1) + 3(N+1)$

$= \cancel{2} + \cancel{4N}+\cancel{8} + \cancel{4N}+\cancel{4} + \cancel{3N}+\cancel{3}$

$= 11N + 17$



3. For the following piece of code:

```
i=0
while(i<2)
    i=i+1
```

a) How many times is the condition checked?

b) How many iterations are there of the while loop?

3)a) Condition checked 3 times

b) There are 2 iterations

i	i<2	i=i+1
0	T	1
1	T	2
2	F	—

2 executions

3 conditional checks

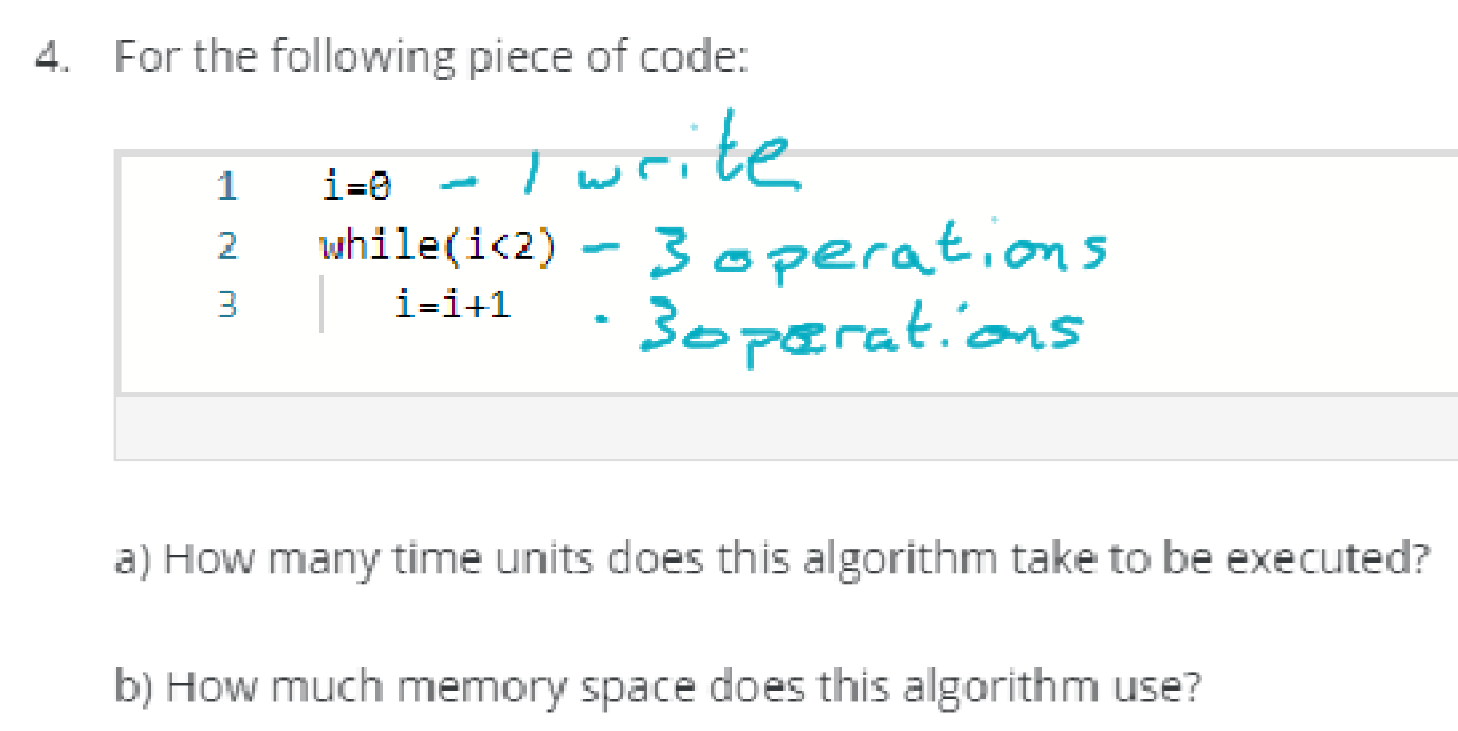
4) i=0 1 write, 1 mem unit

while(i<2) 3 operation X 3 executions=9 t.u.

i=i+1 3operations X 2executions=6 t.u.

This algo takes 16 time units.

1 memory unit.



4. For the following piece of code:

```
i=0          - 1 write
while(i<2)   - 3 operations
    i=i+1    · 3operations
```

a) How many time units does this algorithm take to be executed?

b) How much memory space does this algorithm use?

Growth of functions

We want a method to abstract the calculation of running time. We want to be able to easily compare algos.

size of input

Given $T_1(N) = 21N + 22$

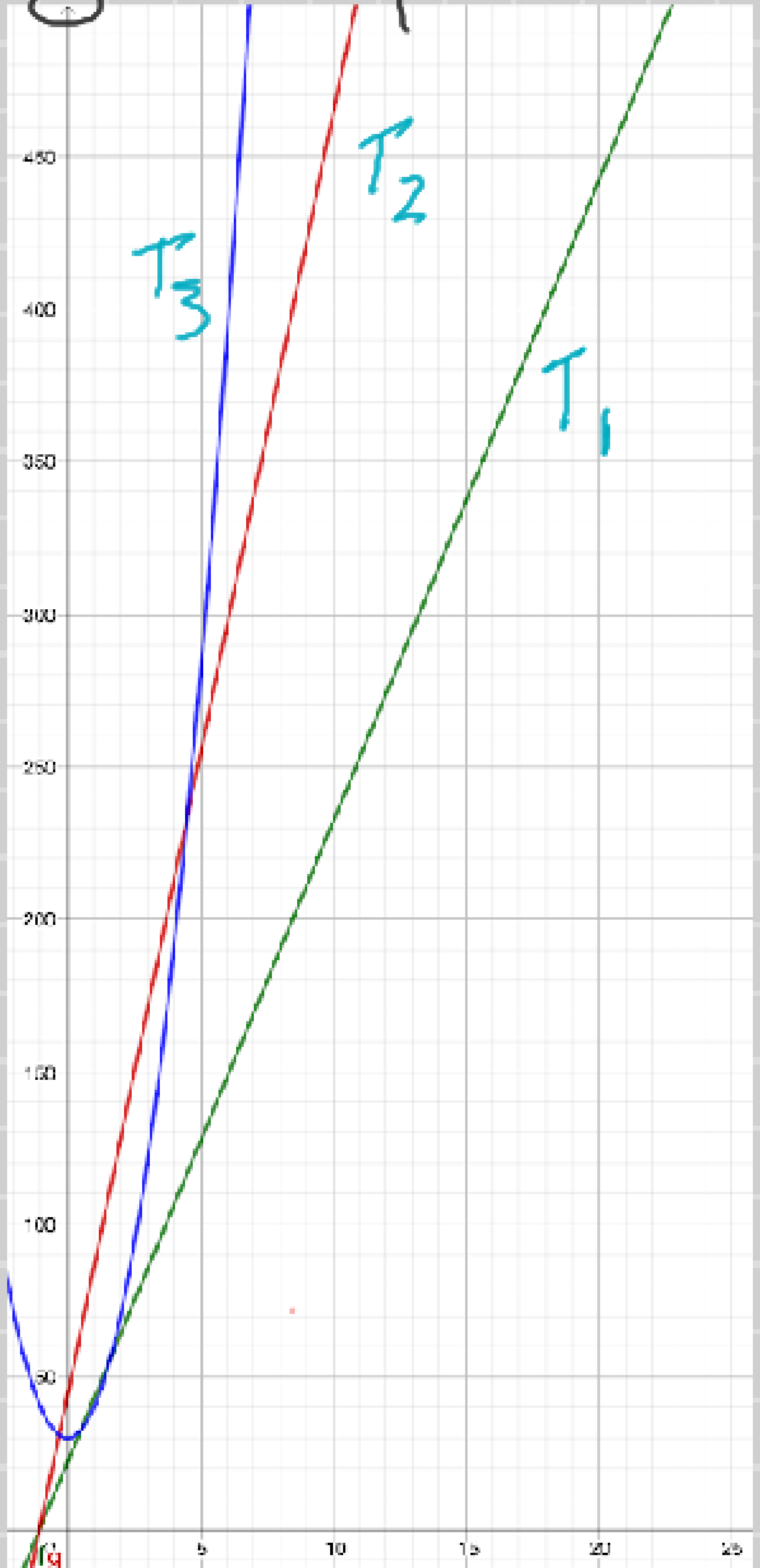
$T_2(N) = 42N + 44$

When plotted, both running times grow in a similar way.

Both grow linearly as the size of N increases.

$T_3(N) = 10N^2 + 3$

There's a point at which the growth of T_3 outpaces T_1 & T_2 for very large inputs



$\exists n_0 : \forall N > n_0$

$T_3(N) > T_2(N)$

* When comparing algos, the growth of the run time is enough. We don't need the coefficients to specified.

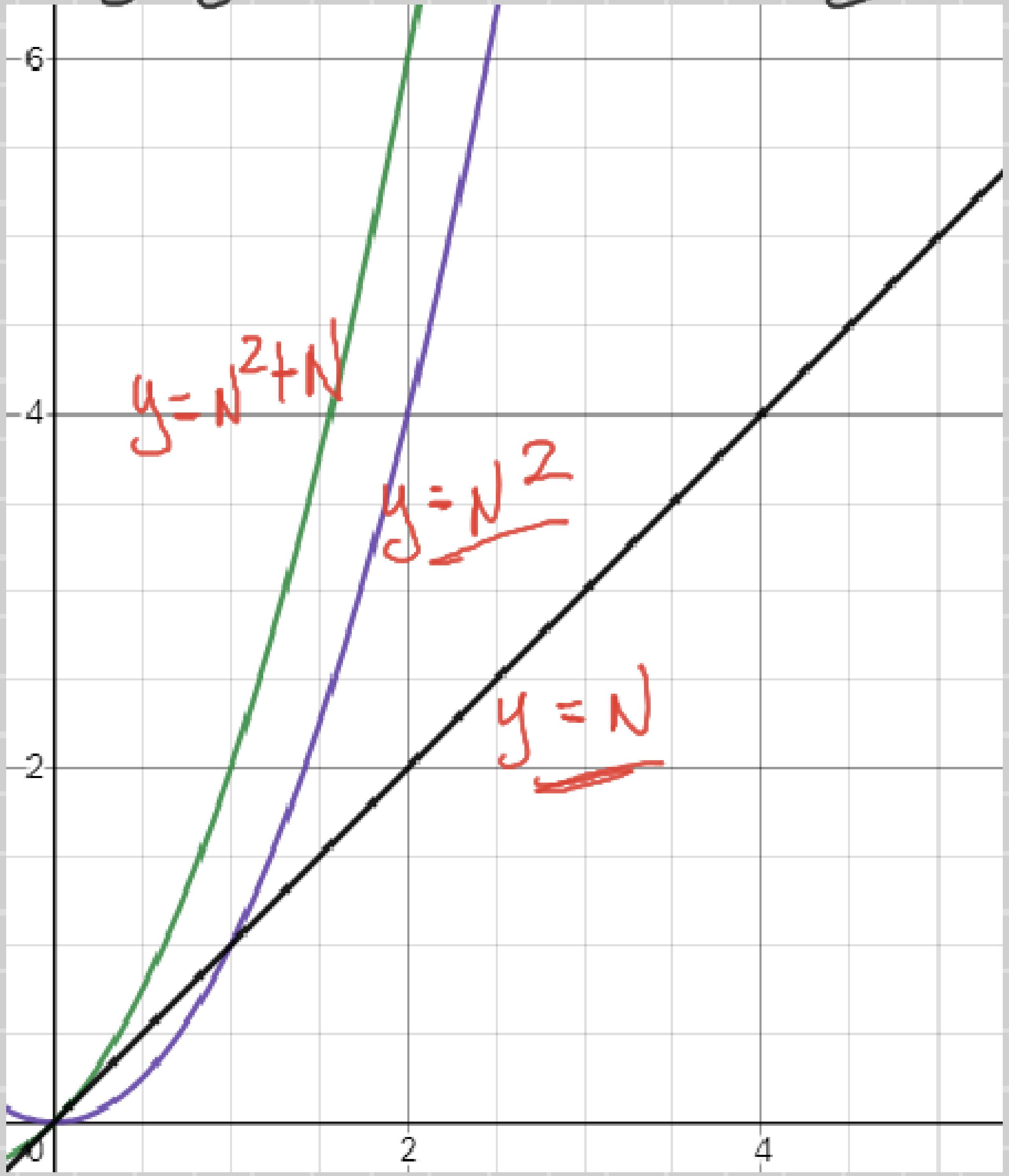
The RAM model is already simplified. we make another simpl

When we analyse asymptotic growth of functions, lower ordered terms of a function **DO NOT** affect the function's growth.

$T(N) = C_1N^2 + C_2N + C_3$

*Discard the coefficients (all constants, are negligible when making estimates)

$N^2 + 1$



To simplify our calculations in comparing running times we will:

1) Use generic constants $T(N) = C_1N + C_2$

2) Ignore constants and lower order terms.

Typical Growth Functions

Slowest

1 constant time

$\log N$

N

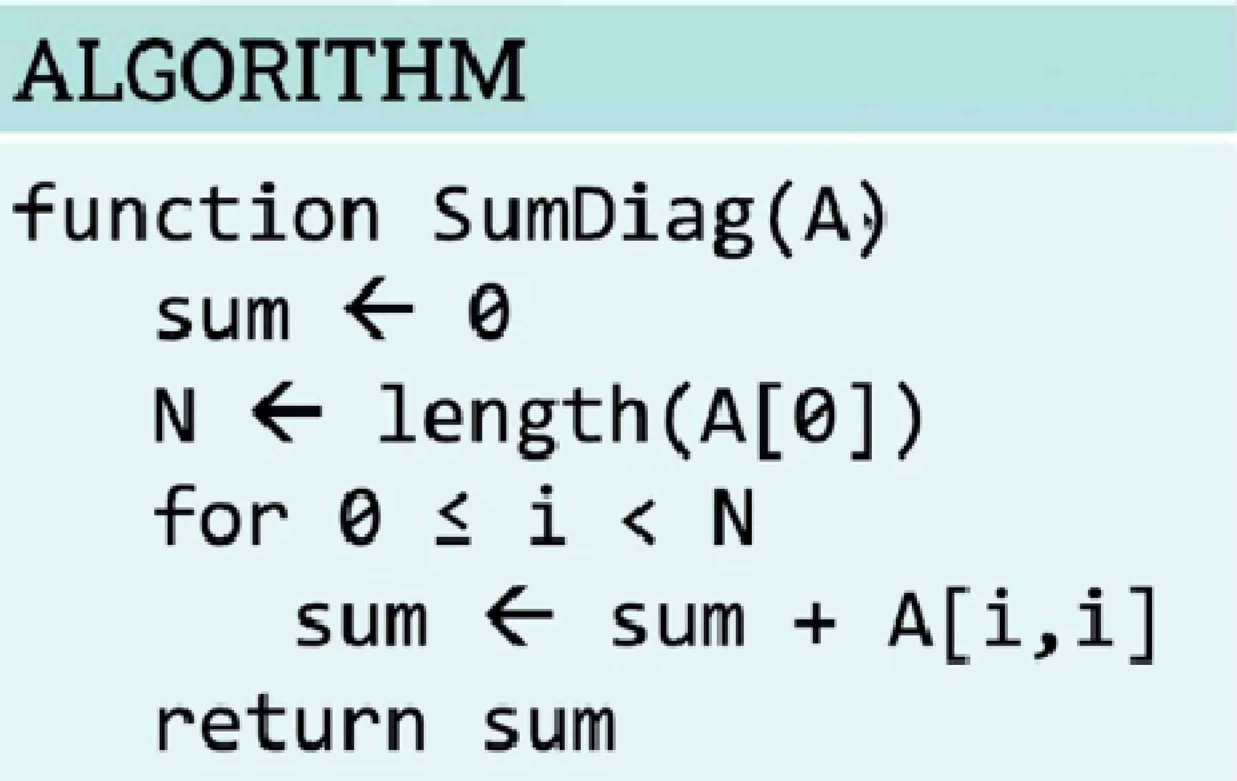
$N \log N$

$$\left.\begin{array}{ll} N^2 & \text{Quadratic} \\ N^3 & \text{Cubic} \end{array}\right\} \text{polynomial}$$

2^N Exponential

$N!$ Factorial

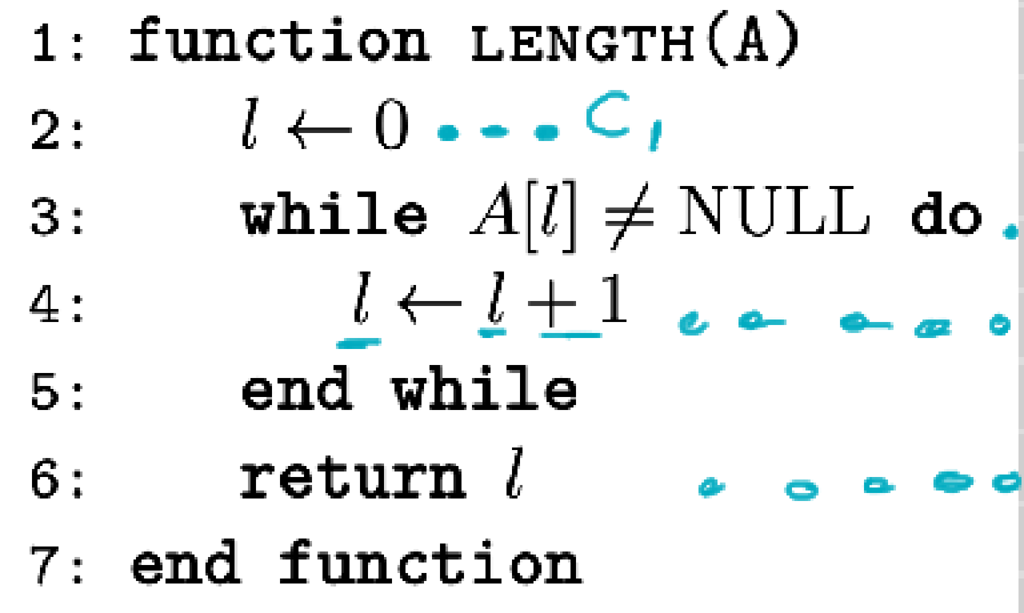
A: square matrix (2D array) with N rows and N columns



ALGORITHM

```
1 function SumDiag(A)
2     sum ← 0
3     N ← length(A[0])
4     for 0 ≤ i < N
5         sum ← sum + A[i,i]
6     return sum
```

How to calculate growth of run time w̄out counting every step



```
1: function LENGTH(A)
2:     $l \leftarrow 0$ ... $C_1$
3:     while $A[l] \neq$ NULL do ... N+1 (N element
4:         $l \leftarrow l + 1$ ...... 3N time ⇒ N
5:     end while
6:     return $l$ ...... Constant
7: end function
```

$(N+1) + \cancel{CN} + C_2$

$C_1 * N + C_2$

2) Constant time operation C_0

3) Length function is not a simple op $C_1 N + C_2$

4) For loop is not a simple operation $C_3 N + C_4$

for loop:

$$\left.\begin{array}{ll} i=0 & \text{Constant} \\ i<N & (N+1)\text{ times} \\ i=i+1 & N\text{ times} \end{array}\right\} C_3 * N + C_4$$

5) $C_5 N$

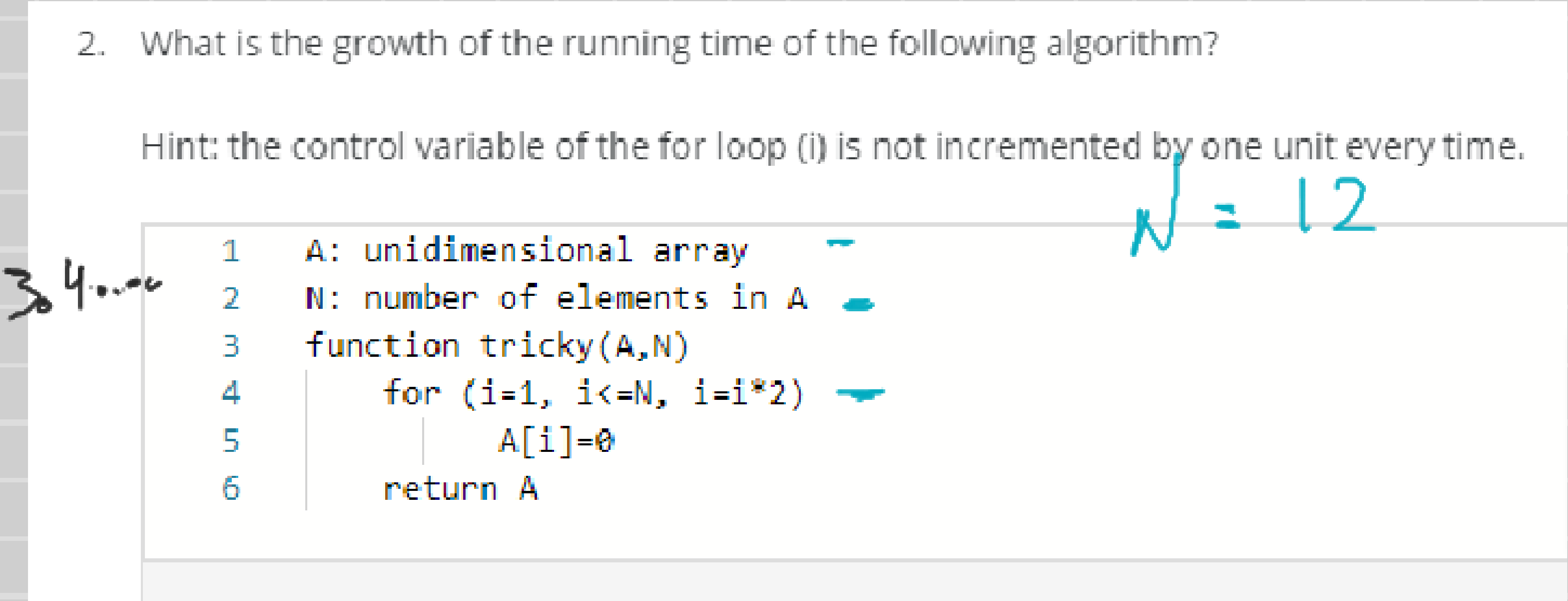
6) C_6

$$T(N) = C_0 + C_1 N + C_2 + C_3 N + C_4 + C_5 N + C_6$$
$$= (C_1 + C_3 + C_5)N + (C_0 + C_2 + C_4 + C_6)$$
$$= C_7 N + C_8 \Rightarrow N \text{ linear function}$$

$N=12$

i	$i \leq N$	$A[i]=0$	$i=i\times 2$
1	T	✓	2
2	T	✓	4
4	T	✓	8
8	T	✓	16
16	F	×	→

$\log(12)=3.4\ldots$



2. What is the growth of the running time of the following algorithm?

Hint: the control variable of the for loop (i) is not incremented by one unit every time.

$N = 12$

```
A: unidimensional array
N: number of elements in A
function tricky(A,N)
    for (i=1, i<=N, i=i*2)
        A[i]=0
    return A
```

We're essentially halving the operations with the loop (approximately)

Whole $\frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} = \left(\frac{1}{2}\right)^k$

logarithmic

log N growth function

We get to the end of array much quicker.

Sum=0	C_0
mult=1	C_1
for $0 \le i \le N$	loop analysis C_2N+C_3
sum=sum+1	C_4N
i=0	C_5
while(i<N*N)	$\boxed{C_6N^2}$
mult=mult*i	C_7N^2
i=i+1	C_8N^2
return (sum+mult)	C_9

loop analysis:

i=0	C_2	$\underbrace{C_3N+C_4N}_{C_5N}+C_2$
$i \le N$	C_3N	
i=i+1	C_4N	C_5N+C_2

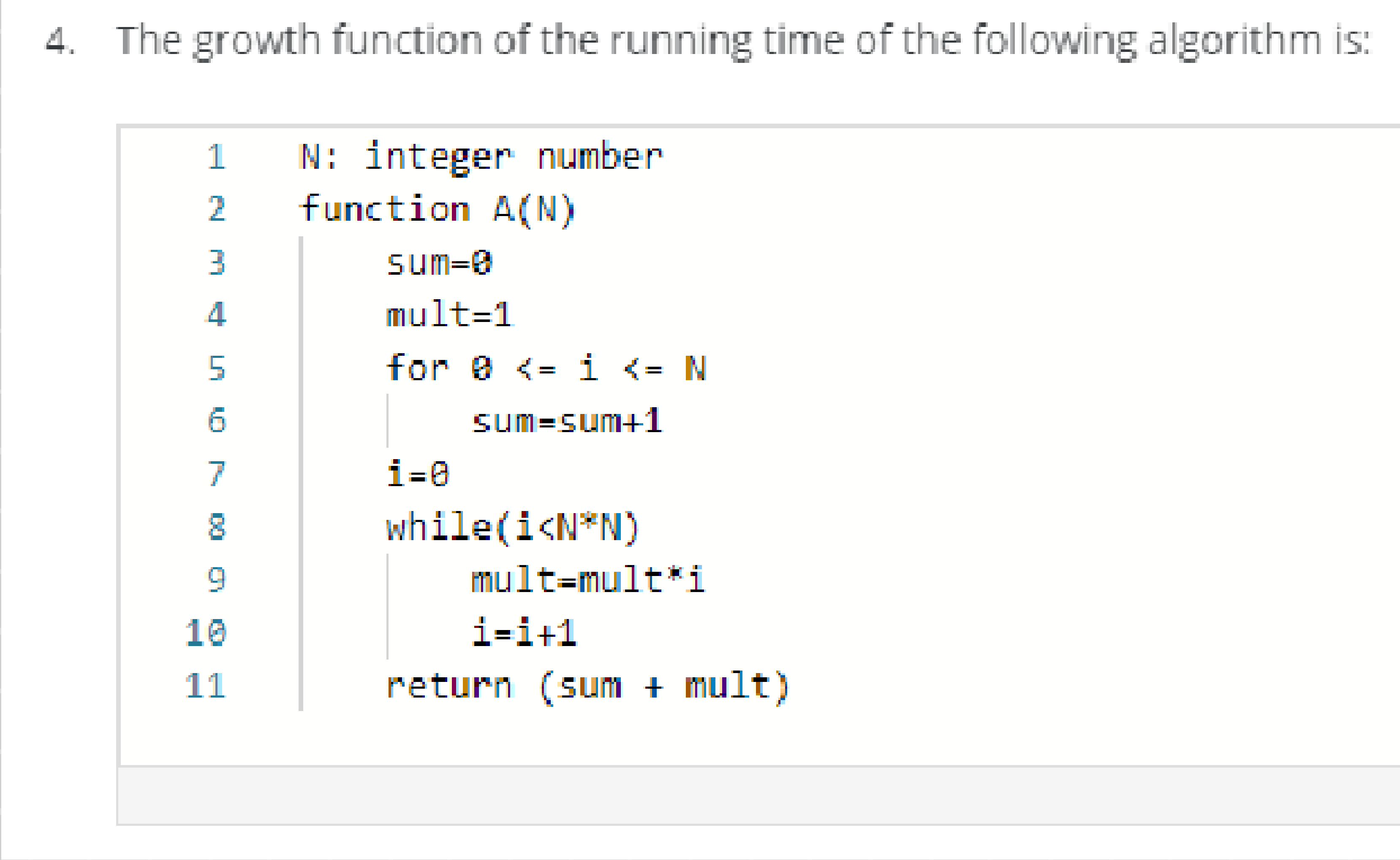
$$T(N) = C_0 + C_1 + \underline{C_2N} + C_3 + \underline{C_4N} + C_5 + \underline{C_6N^2} + \underline{C_7N^2} + \underline{C_8N^2} + C_9$$

$$= (C_2+C_4)N + (C_6+C_7+C_8)N^2 + C_0+C_1+C_3+C_5+C_9$$

$$= C_{10}N + \boxed{C_{11}N^2} + C_{12}$$

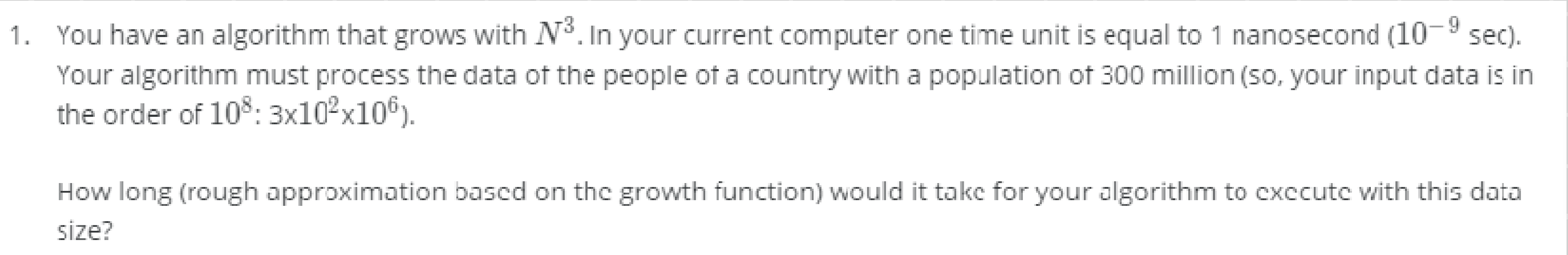
$$\left\{\begin{array}{l} C_2+C_4=C_{10} \\ C_6+C_7+C_8=C_{11} \\ C_0+C_1+C_3+C_5+C_9=C_{12} \end{array}\right\}$$

$$= N^2$$



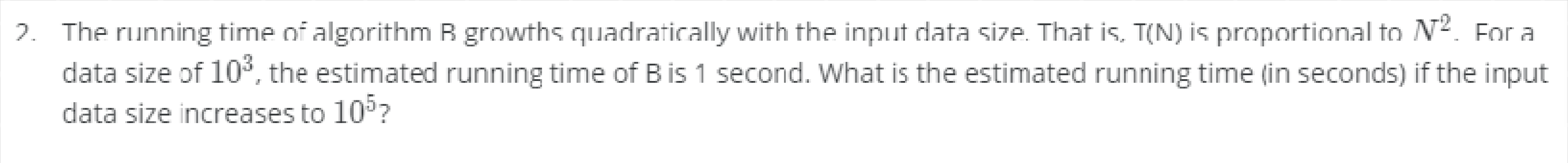
4. The growth function of the running time of the following algorithm is:

```
N: integer number
function A(N)
    sum=0
    mult=1
    for 0 <= i <= N
        sum=sum+1
    i=0
    while(i<N*N)
        mult=mult*i
        i=i+1
    return (sum + mult)
```

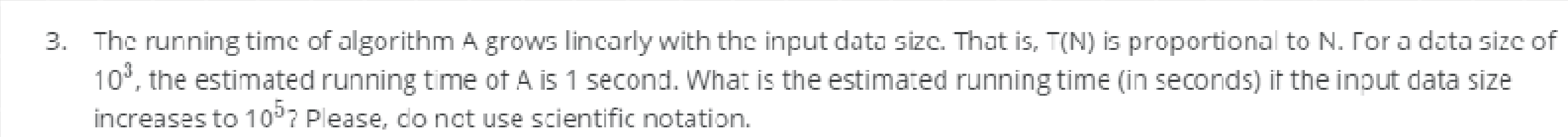


1. You have an algorithm that grows with N^3. In your current computer one time unit is equal to 1 nanosecond (10^{-9} sec). Your algorithm must process the data of the people of a country with a population of 300 million (so, your input data is in the order of 10^8: 3x10^2x10^6).

   How long (rough approximation based on the growth function) would it take for your algorithm to execute with this data size?



2. The running time of algorithm B growths quadratically with the input data size. That is, T(N) is proportional to N^2. For a data size of 10^3, the estimated running time of B is 1 second. What is the estimated running time (in seconds) if the input data size increases to 10^5?



3. The running time of algorithm A grows linearly with the input data size. That is, T(N) is proportional to N. For a data size of 10^3, the estimated running time of A is 1 second. What is the estimated running time (in seconds) if the input data size increases to 10^5? Please, do not use scientific notation.