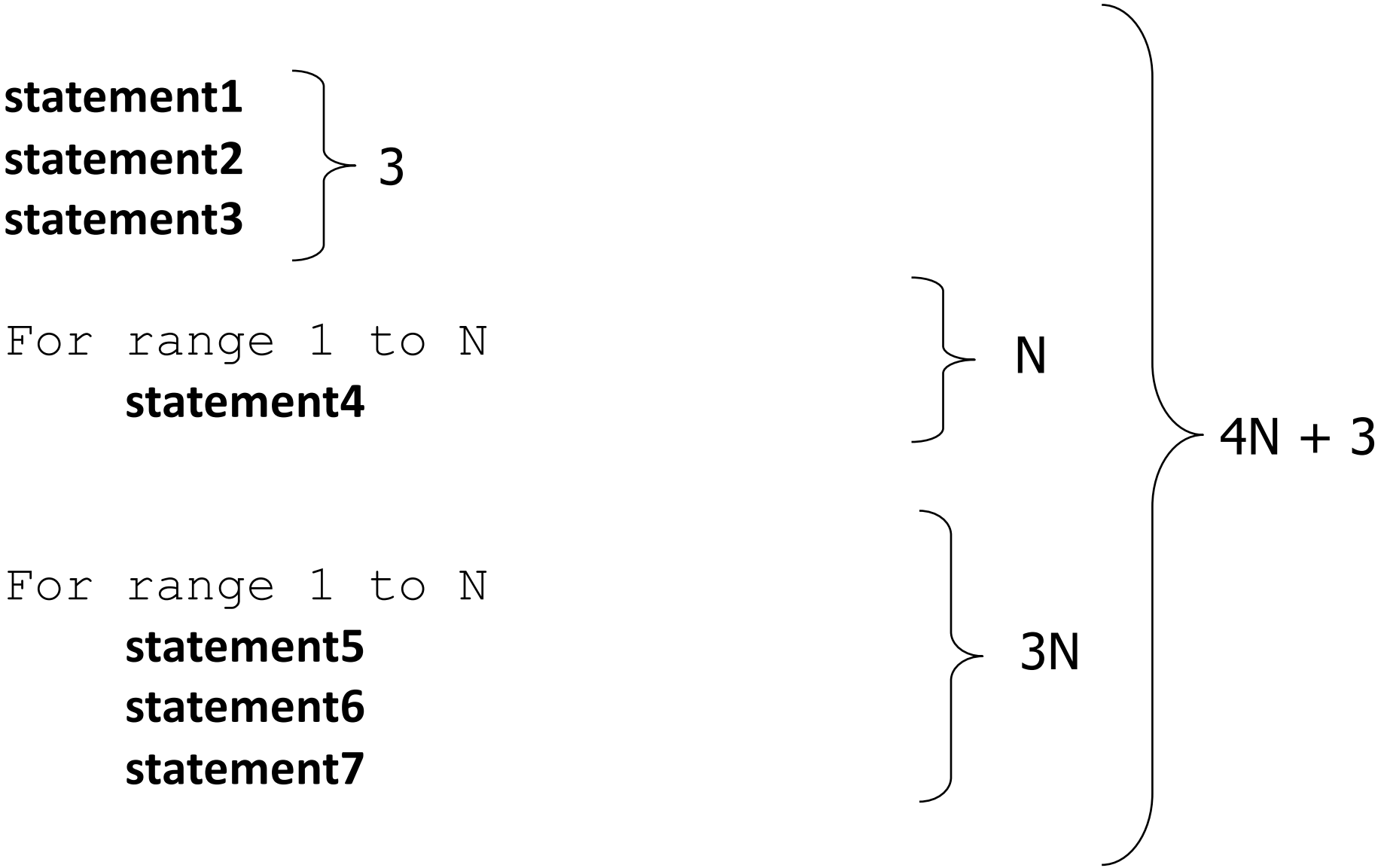
Efficiency

Runtime Efficiency

- **efficiency**: A measure of the use of computing resources by code.
  - can be relative to speed (time), memory (space), etc.
  - most commonly refers to run time

- Assume the following:
  - Any single program statement takes the same amount of time to run.
  - A function runtime is measured by the total of the statements inside the function body.
  - A loop's runtime, if the loop repeats N times, is N times the runtime of the statements in its body.

Efficiency examples



Efficiency examples 2

```
For range 1 to N
    For range 1 to N
        statement1
```

N^2

```
For range 1 to N
    statement2
    statement3
    statement4
    statement5
```

4N

$N^2 + 4N$

- How many statements will execute if N = 10? If N = 1000?

Algorithm growth rates

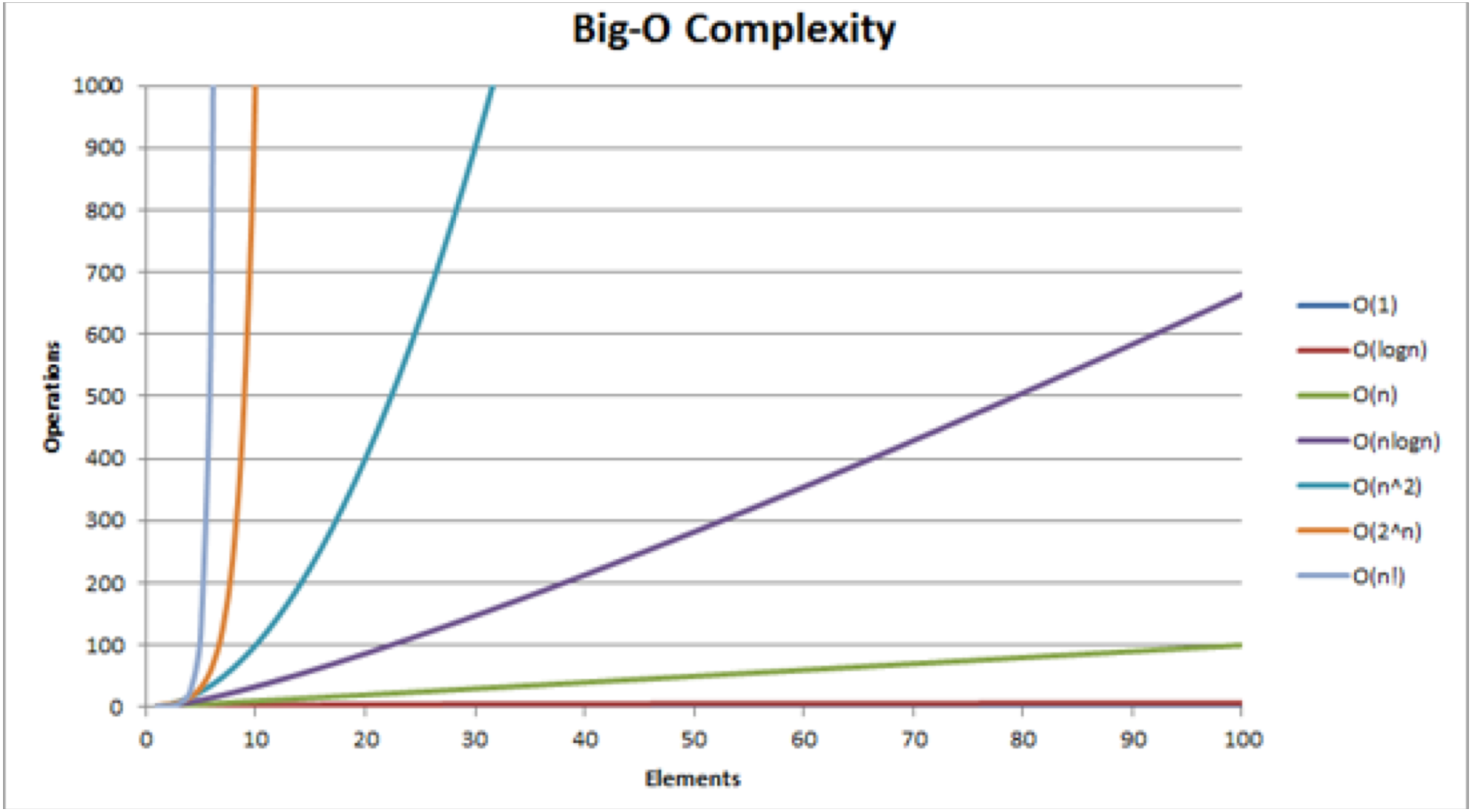
- We measure runtime in proportion to the input data size, N.
  - **growth rate**: Change in runtime as N changes.

- Say an algorithm runs **$0.4N^3 + 25N^2 + 8N + 17$** statements.
  - Consider the runtime when N is *extremely large* .
  - We ignore constants like 25 because they are tiny next to N.
  - The ***highest-order term (N^3) dominates*** the overall runtime.
  - We say that this algorithm runs "on the order of" N^3.
  - or **$O(N^3)$** for short ("**Big-Oh** of N cubed")

Big O?

```
count = 0
while (count < n)
   repeat = 0
   while (repeat < n)
        statement1
        statement2
        statement3
        statement4
   end inner while
end outer while
```

A) O(1)
B) O(n)
C) $O(\log_2 n)$
D) $O(n^2)$
E) $O(2^n)$



http://www.daveperrett.com/articles/2010/12/07/comp-sci-101-big-o-notation/

Efficiency – Big O?

```
search(list[], a)
    N = number of elements in list
    count = 0

    while count < N do the following
        if (list[count] == a) // if matches
            return count    // return the position

        count++

    return -1  // not found

End search
```

A) O(1)
B) O(n)
C) $O(\log_2 n)$
D) $O(n^2)$
E) $O(2^n)$

Recall our game from yesterday…

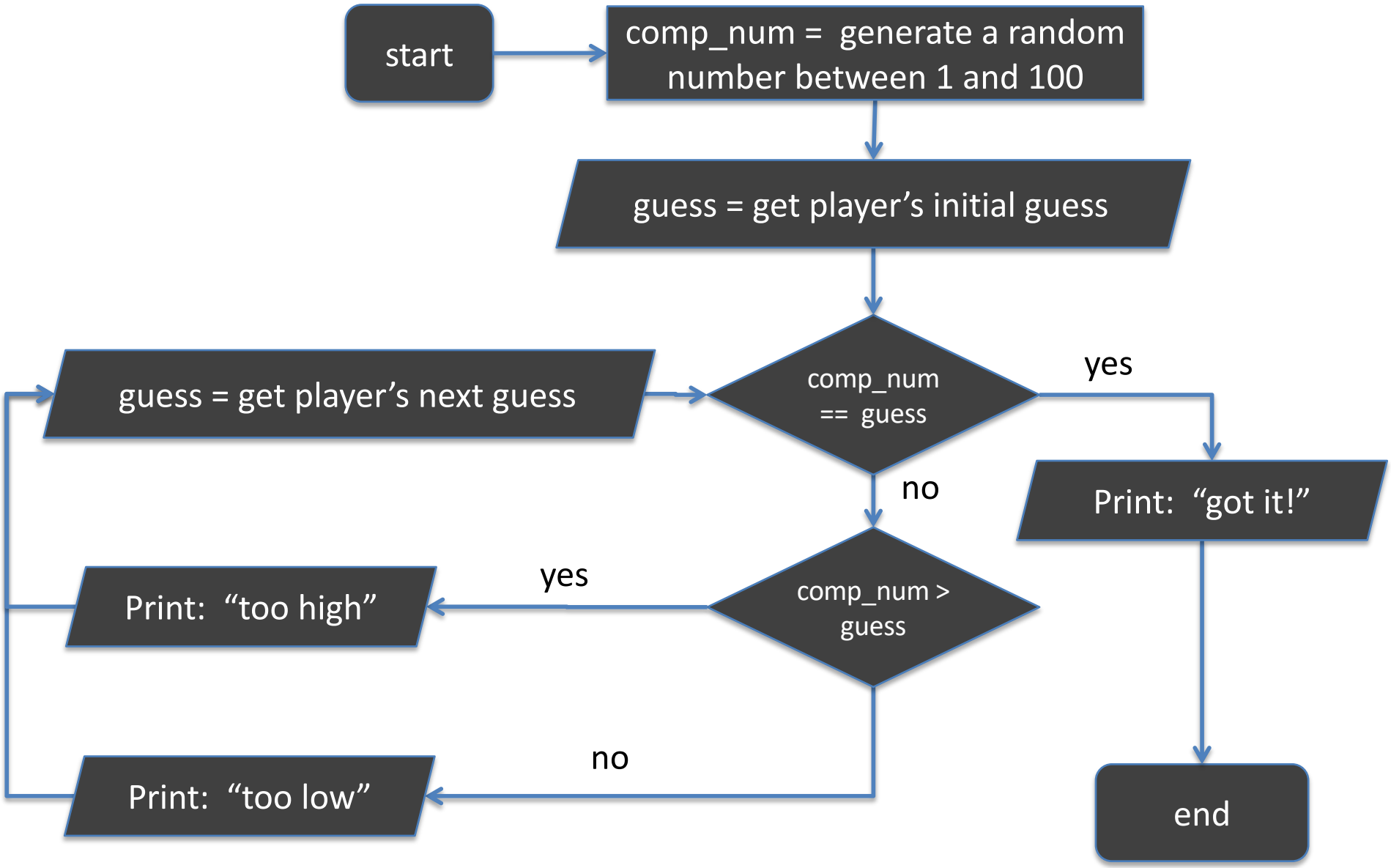
- A number between 1 and 100 is generated
- The player tries to guess the number.
- Each time the player guesses, the player is told if their guess is correct, too high, or too low.

- What strategy did our guessers use to guess?

Design a guessing algorithm

- Design an algorithm for playing against the game
- Create a flowchart and pseudocode that represents your algorithm

flowchart


start
comp_num = generate a random number between 1 and 100
guess = get player's initial guess
comp_num == guess
yes
Print: "got it!"
no
guess = get player's next guess
comp_num > guess
yes
Print: "too high"
no
Print: "too low"
end

Algorithm exercise: guessing game

Assumptions: **comp_num** will be an integer taken at random between 1 and 100
players_guess will be an integer between 1 and 100.

comp_num = an int selected at random between 1 and 100
players_guess = an int entered by player between 1 and 100

WHILE players_guess != comp_num

Check **IF** players_guess > comp_num
PRINT "Your number is too high. Guess again."
ELSE
PRINT "Your number is too low. Guess again."

players_guess = an int entered by player between 1 and 100
// player enters a new integer in players_guess and it is
// checked for equality with comp_num
END WHILE
Print "You got it – finally!"

Evaluating your Algorithm

- Does it halt?
- Does it generate the correct output for any input?
- Is it efficient?