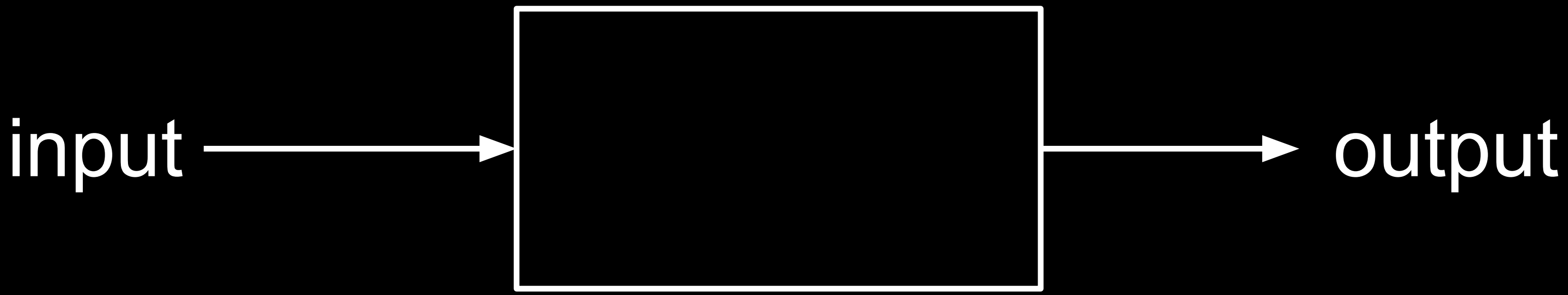
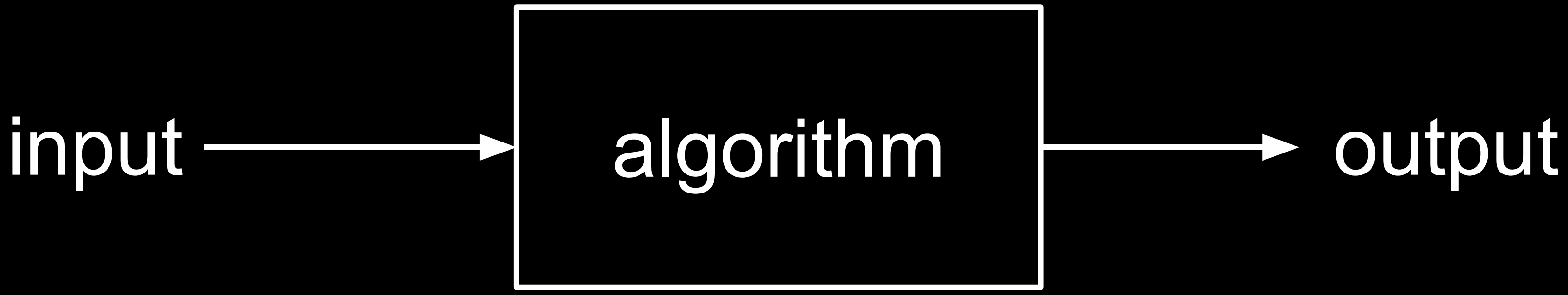
ALGORITHMS

who solves the problem?




input
algorithm
output

algorithm, program, process

"A finitely long rule consisting of individual instructions is called an algorithm."

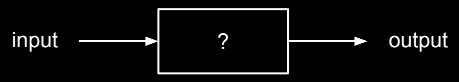
Source: Vornberger, O., Algorithmen und Datenstrukturen, Lecture notes: http://www-lehre.inf.uos.de/~ainf/2013/PDF/skript.pdf

"A program is an algorithm expressed in a programming language."

Source: Vornberger, O., Algorithmen und Datenstrukturen, Lecture notes: http://www-lehre.inf.uos.de/~ainf/2013/PDF/skript.pdf

"A process is a program that is currently executed by a computer."

Source: Vornberger, O., Algorithmen und Datenstrukturen, Lecture notes: http://www-lehre.inf.uos.de/~ainf/2013/PDF/skript.pdf



greatest common divisor

euclidean algorithm

Identify the larger number of a and b. If a < b, swap numbers so that a > b

Subtract b from a and replace a with the result

Repeat until one of the numbers becomes 0

Return the number that is not zero

Loop 1:
a = 18, b = 48 → swap

Loop 1:

a = 18, b = 48 → swap → a = 48, b = 18

a = 48 - 18 = 30

```
Loop 1:
a = 18, b = 48 ⟶ swap ⟶ a = 48, b = 18
a = 48 - 18 = 30

Loop 2:
a = 30, b = 18 ⟶ no swap
a = 30 - 18 = 12
```

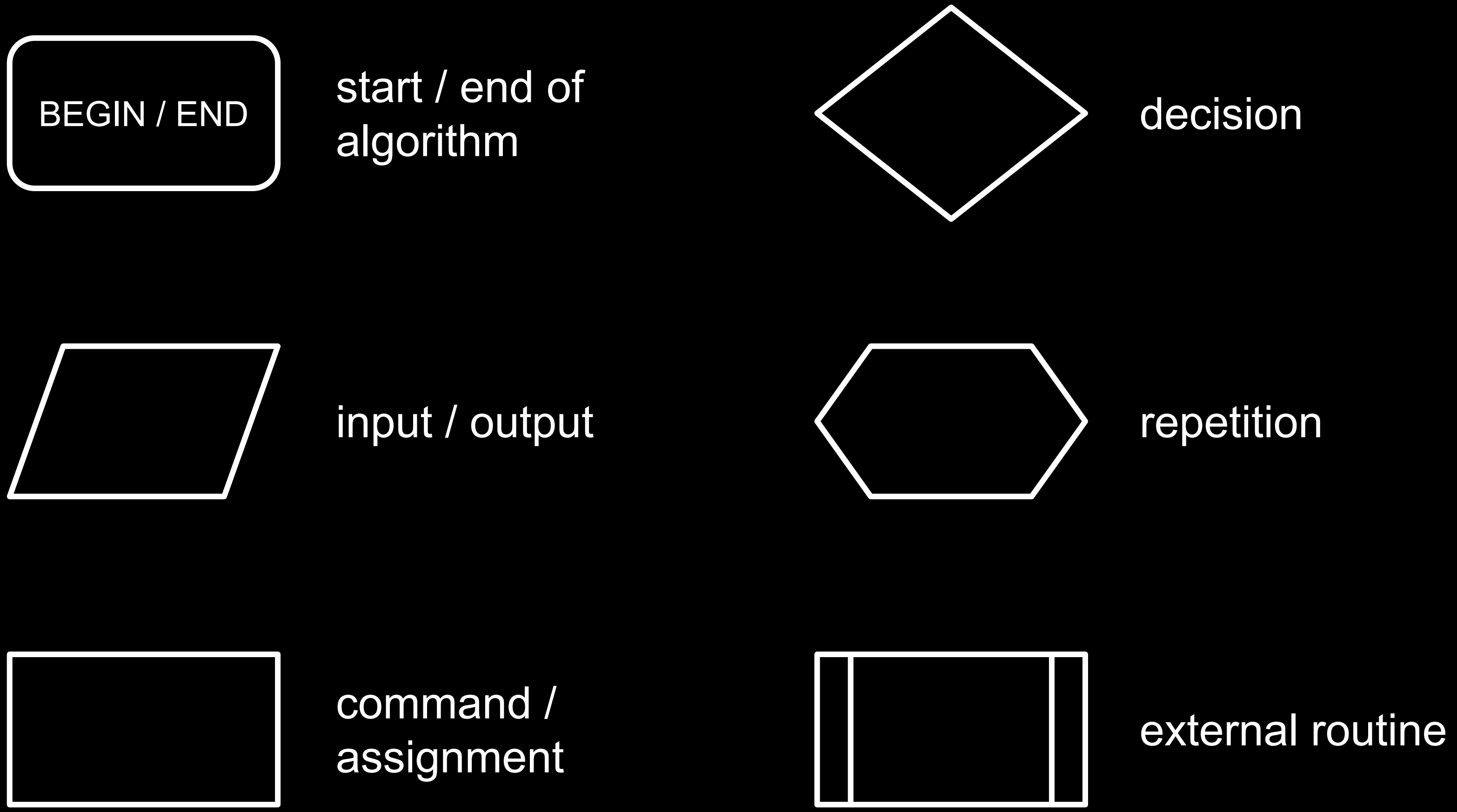
```
Loop 1:
a = 18, b = 48 → swap → a = 48, b = 18
a = 48 - 18 = 30

Loop 2:
a = 30, b = 18 → no swap
a = 30 - 18 = 12

Loop 3:
a = 12, b = 18 → swap → a = 18, b = 12
a = 18 - 12 = 6
```

```
Loop 1:
a = 18, b = 48 → swap → a = 48, b = 18
a = 48 - 18 = 30

Loop 2:
a = 30, b = 18 → no swap
a = 30 - 18 = 12

Loop 3:
a = 12, b = 18 → swap → a = 18, b = 12
a = 18 - 12 = 6

Loop 4:
a = 6, b = 12 → swap → a = 12, b = 6
a = 12 - 6 = 6
```

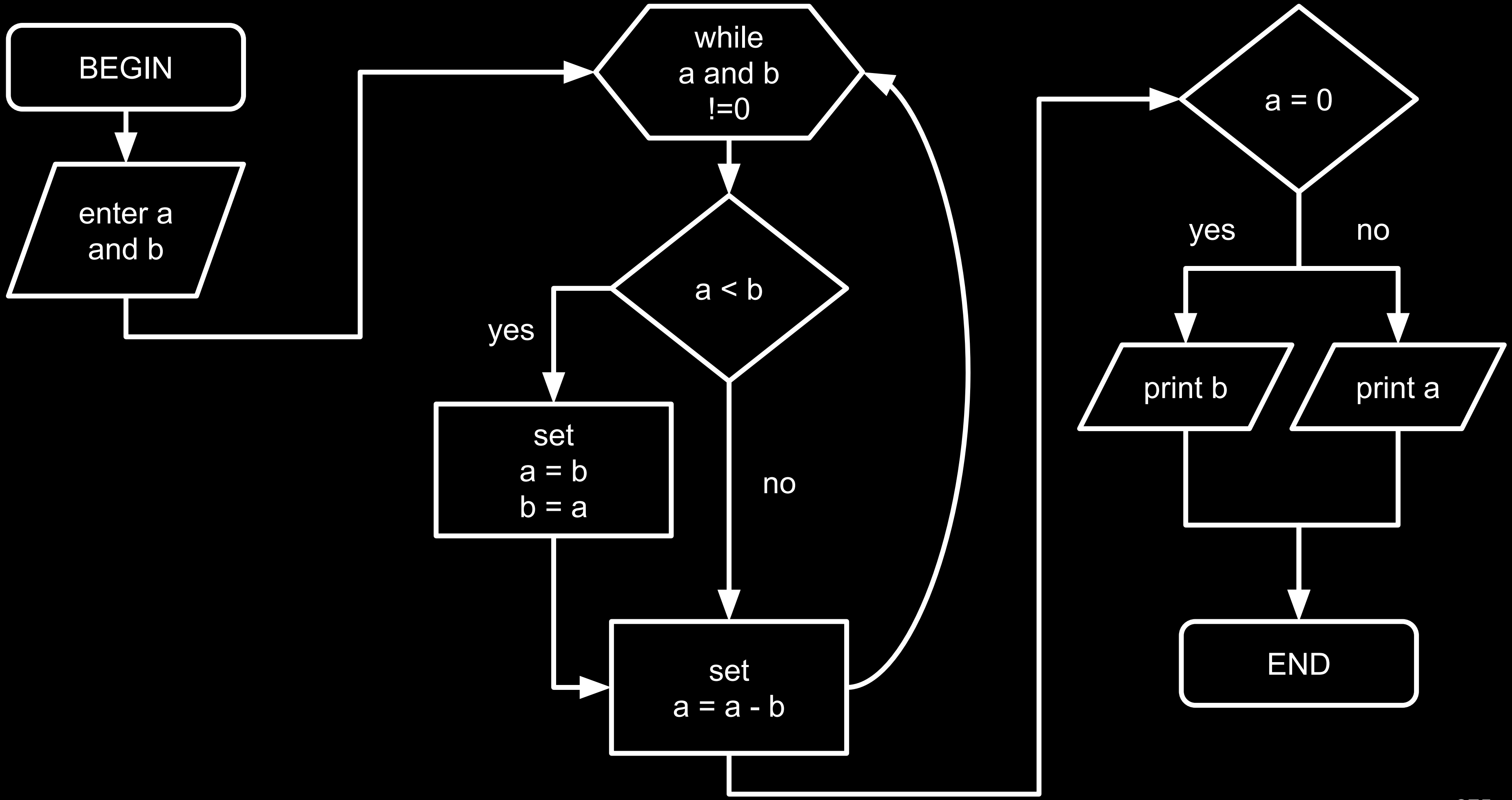
```
Loop 1:
a = 18, b = 48 → swap → a = 48, b = 18
a = 48 - 18 = 30

Loop 2:
a = 30, b = 18 → no swap
a = 30 - 18 = 12

Loop 3:
a = 12, b = 18 → swap → a = 18, b = 12
a = 18 - 12 = 6

Loop 4:
a = 6, b = 12 → swap → a = 12, b = 6
a = 12 - 6 = 6

Loop 5:
a = 6, b = 6 → no swap
a = 6 - 6 = 0
```

```
Loop 1:
a = 18, b = 48 → swap → a = 48, b = 18
a = 48 - 18 = 30

Loop 2:
a = 30, b = 18 → no swap
a = 30 - 18 = 12

Loop 3:
a = 12, b = 18 → swap → a = 18, b = 12
a = 18 - 12 = 6

Loop 4:
a = 6, b = 12 → swap → a = 12, b = 6
a = 12 - 6 = 6

Loop 5:
a = 6, b = 6 → no swap
a = 6 - 6 = 0

return b = 6
```

flow diagrams



BEGIN / END
start / end of algorithm
decision
input / output
repetition
command / assignment
external routine


BEGIN
enter a
and b
while
a and b
!=0
a < b
yes
set
a = b
b = a
no
set
a = a - b
a = 0
yes
no
print b
print a
END

square roots

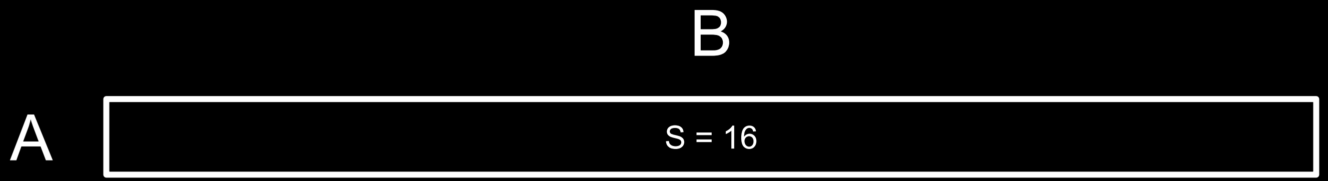
babylonian method

calculate square root of

$x = 16$

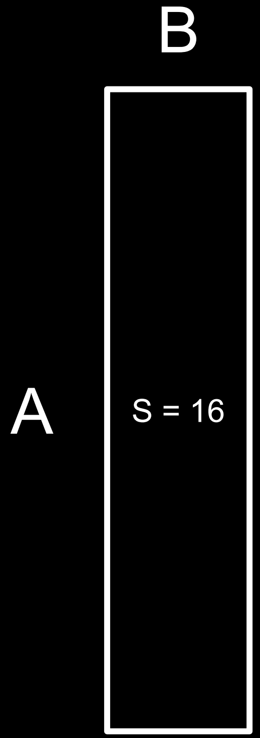
$A = 1$

$B = X / A = 16$



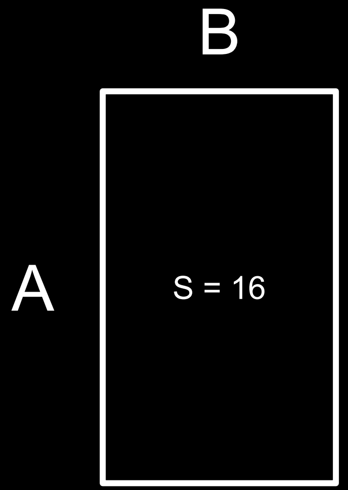
$$A = (A + B) / 2 = 17 / 2 = 8.5$$

$$B = X / A = 16 / 8.5 \approx 1.88$$



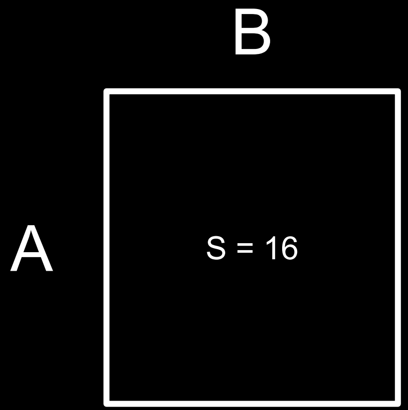
$$A = (A + B) / 2 \approx 10.38 / 2 \approx 5.19$$

$$B = X / A \approx 16 / 5.19 \approx 3.08$$



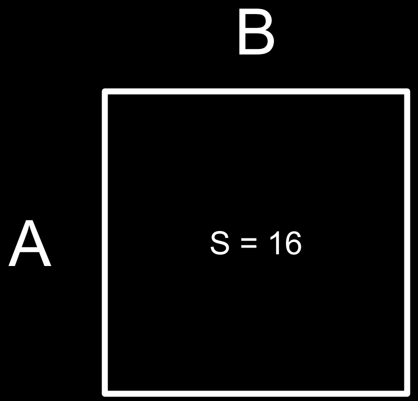
$$A = (A + B) / 2 \approx 8.27 / 2 \approx 4.14$$

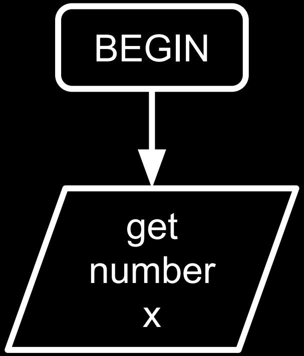
$$B = X / A \approx 16 / 4.14 \approx 3.86$$

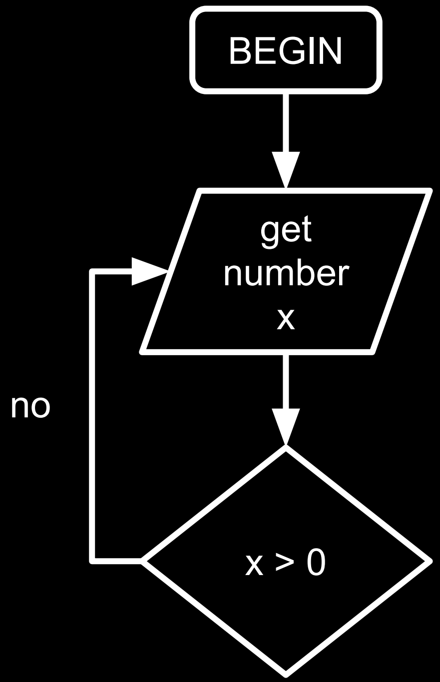


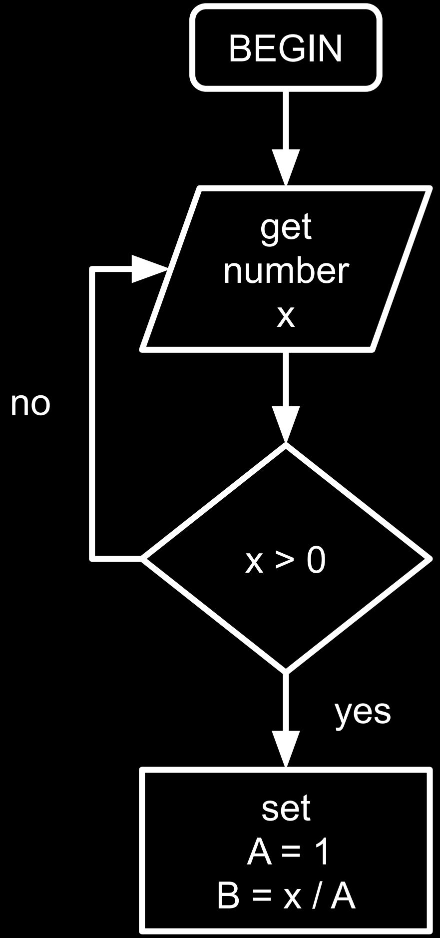
$A = (A + B) / 2 = 8 / 2 = \mathbf{4}$

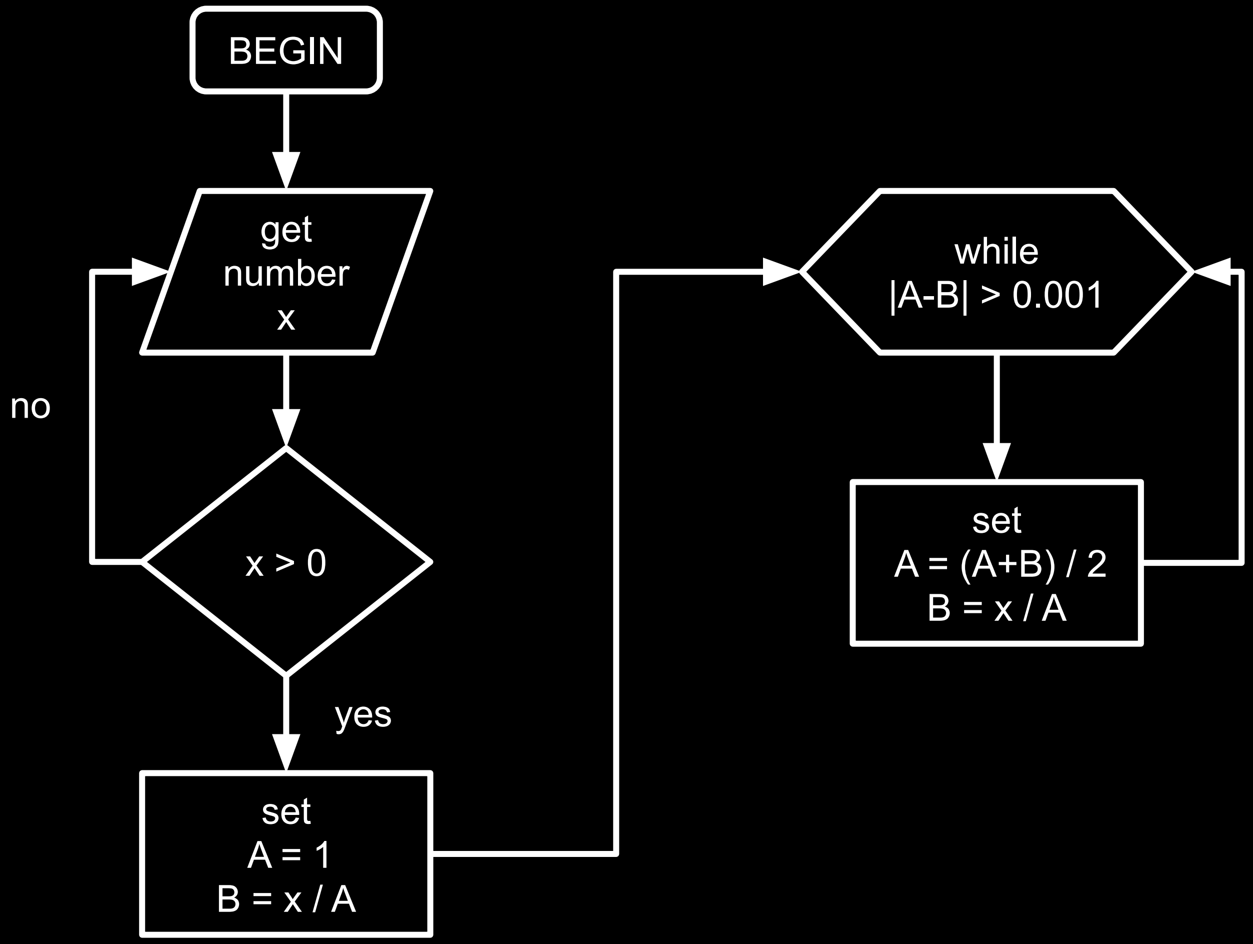
$B = X / A = 16 / 4 = \mathbf{4}$

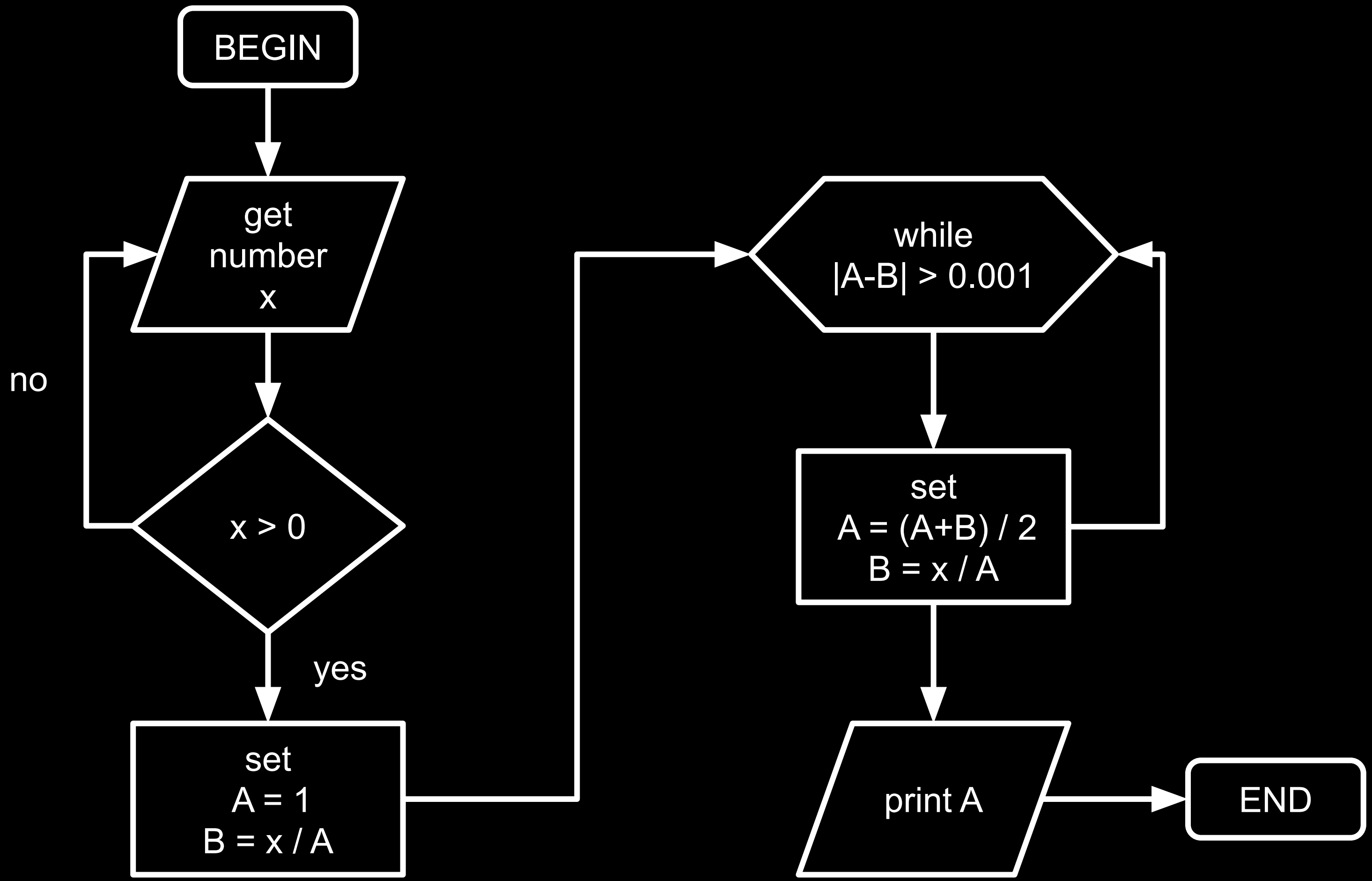


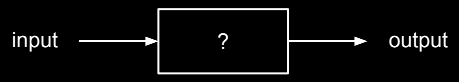

BEGIN
get
number
x


BEGIN
get
number
x
no
x > 0

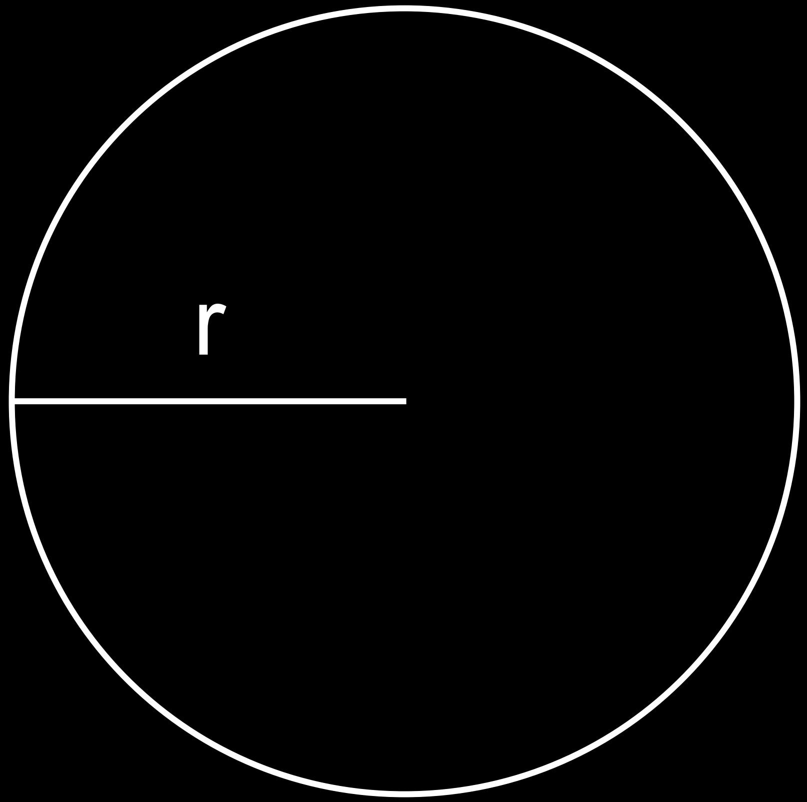

BEGIN
get
number
x
no
x > 0
yes
set
A = 1
B = x / A

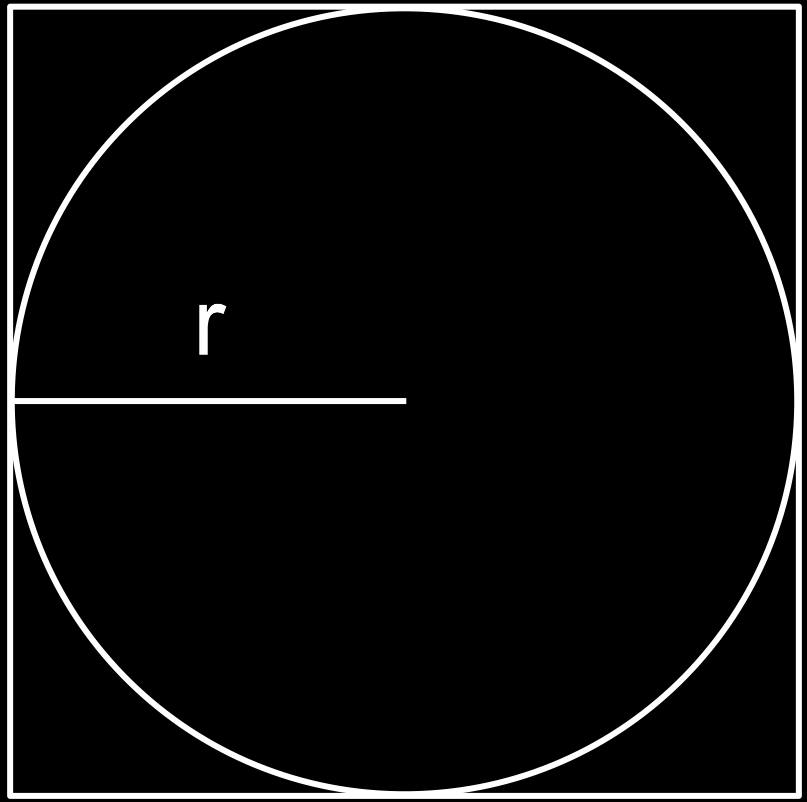

BEGIN
get
number
x
no
x > 0
yes
set
A = 1
B = x / A
while
|A-B| > 0.001
set
A = (A+B) / 2
B = x / A

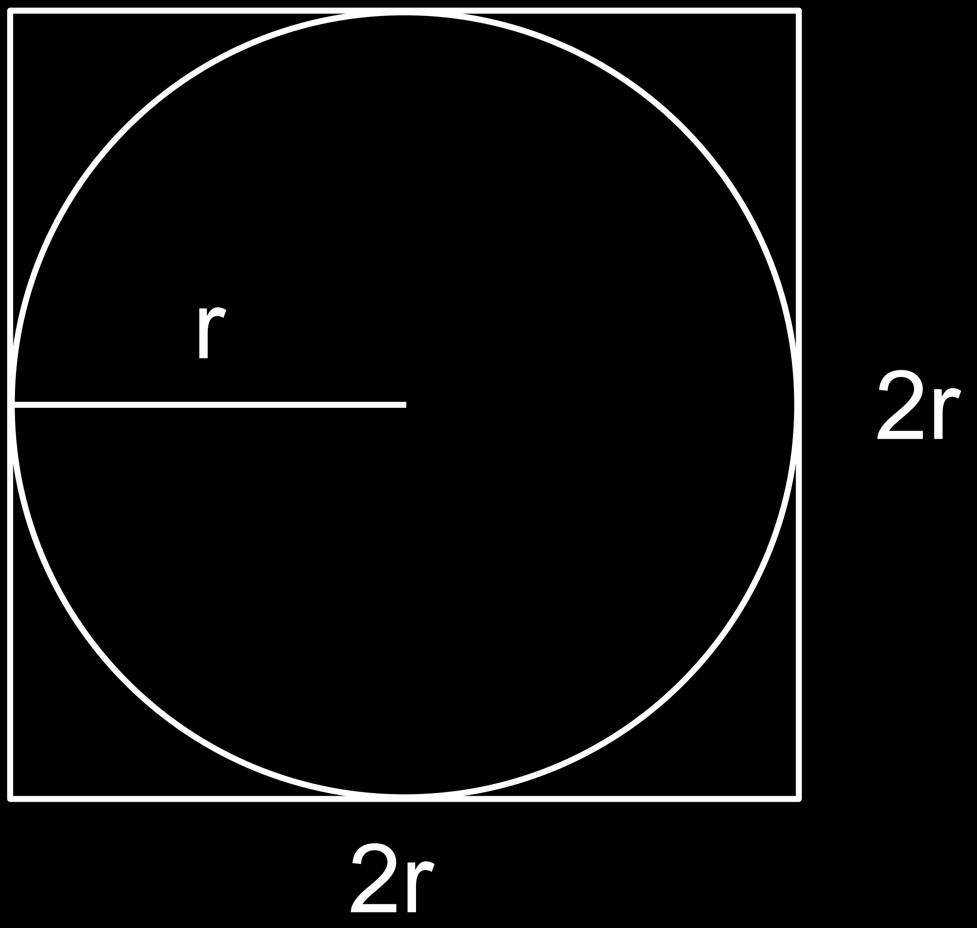

BEGIN
get number x
x > 0
no
yes
set
A = 1
B = x / A
while
|A-B| > 0.001
set
A = (A+B) / 2
B = x / A
print A
END

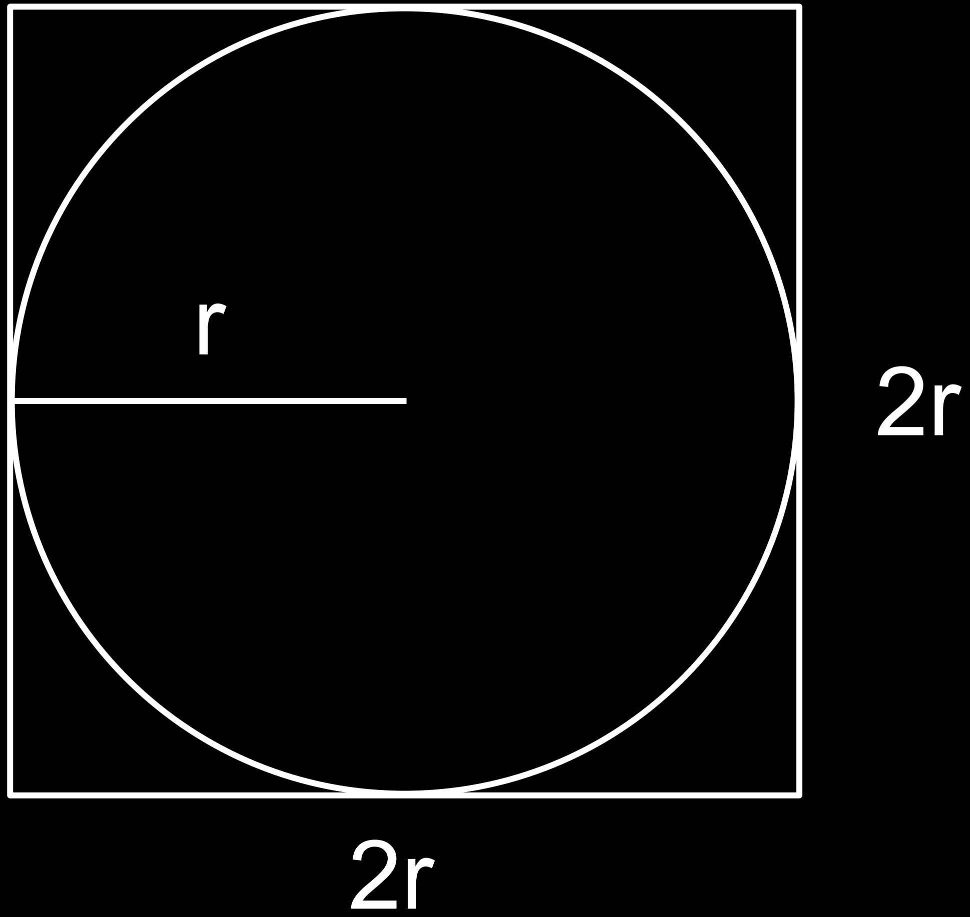


estimating π

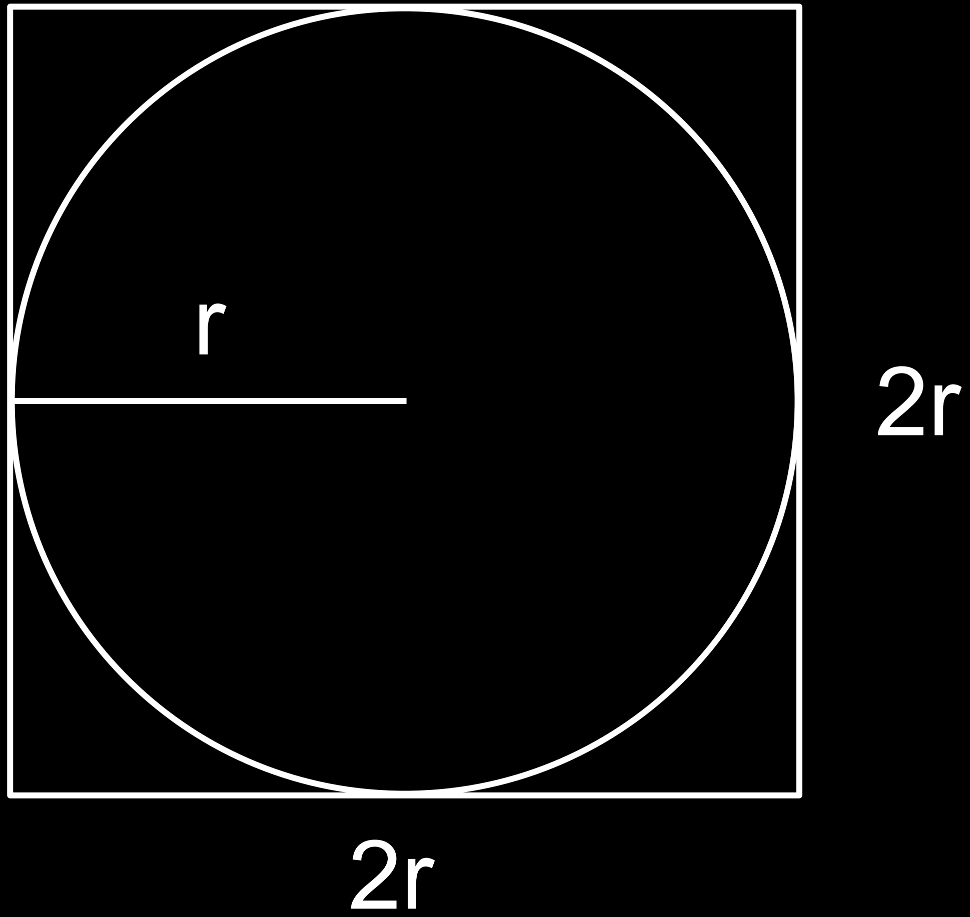

r


r

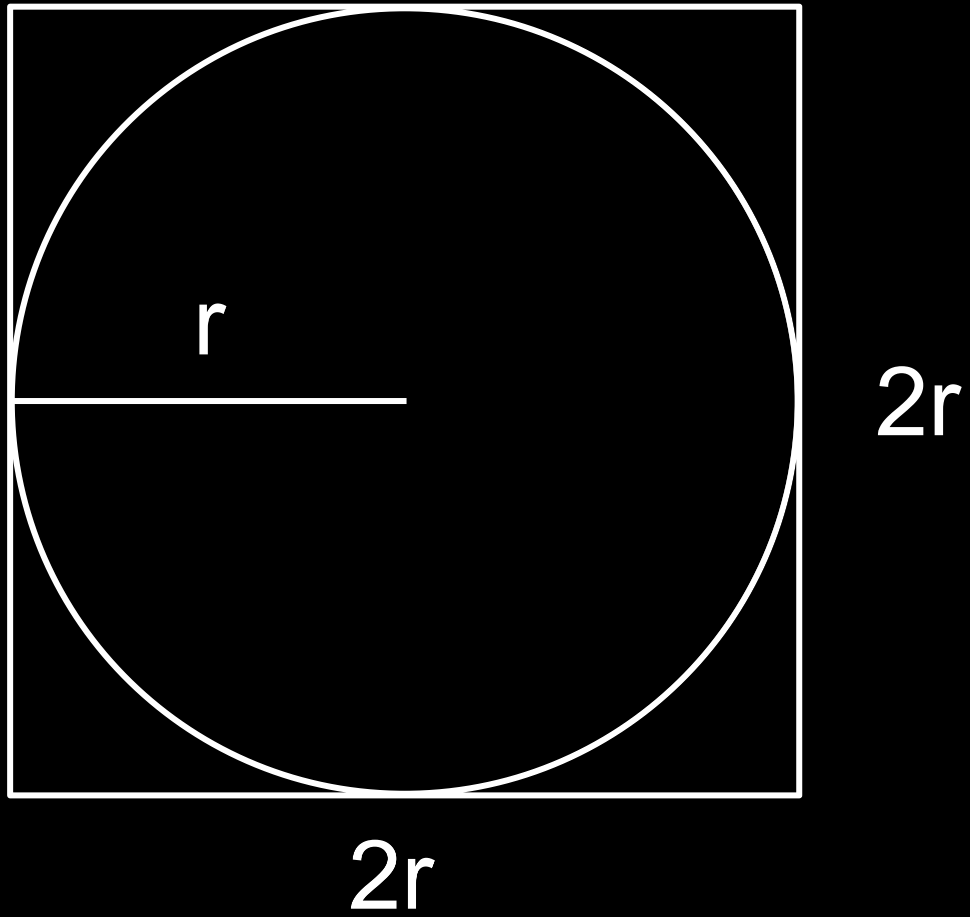

r
2r
2r



$$\frac{\bigcirc}{\square} = \frac{\pi r^2}{4r^2}$$

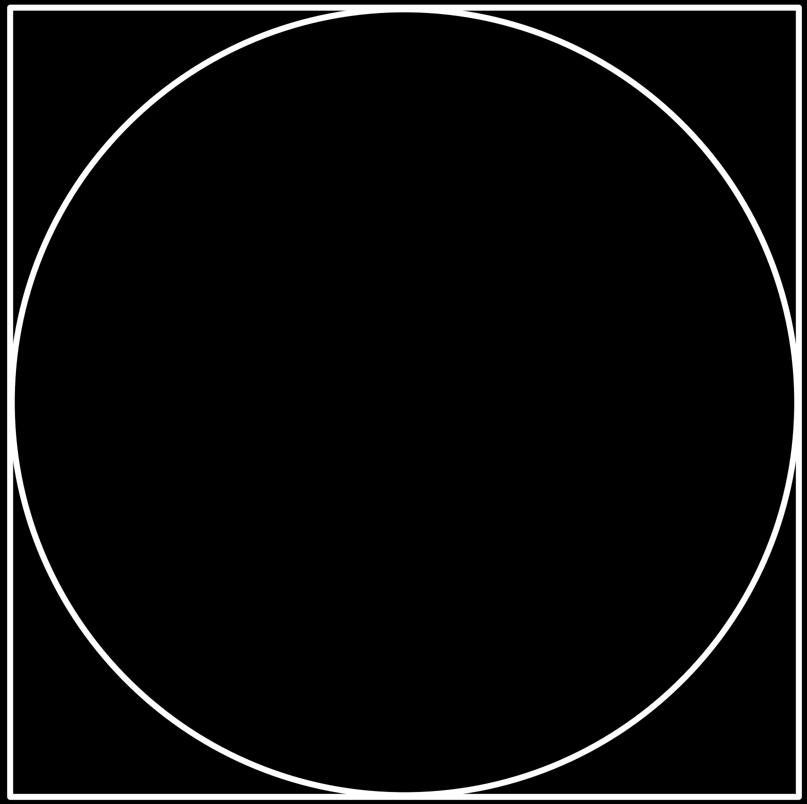


$$\frac{\bigcirc}{\square} = \frac{\pi r^2}{4 r^2}$$

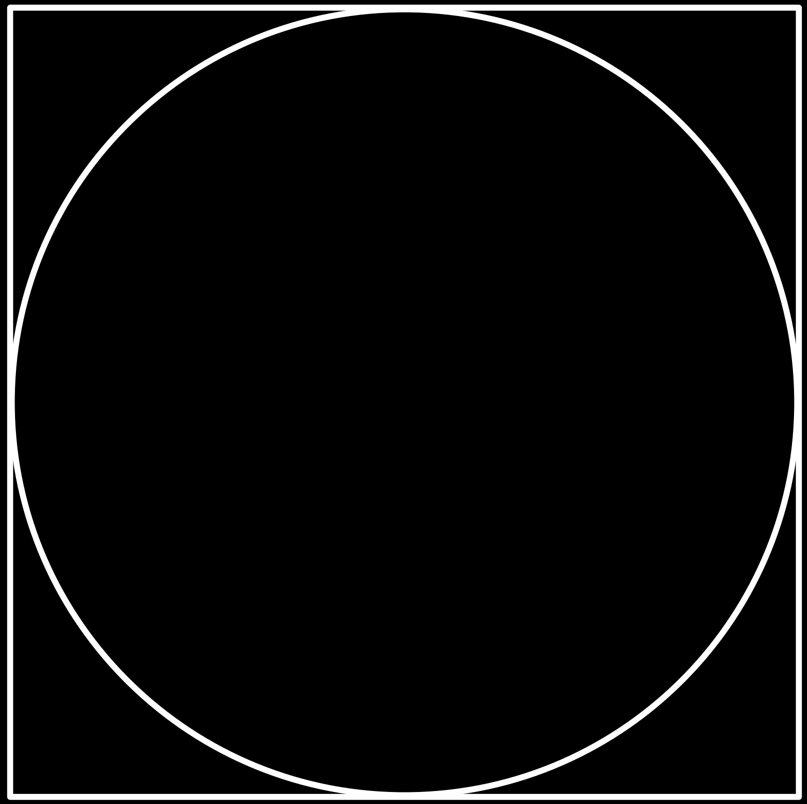


$$\frac{\bigcirc}{\square} = \frac{\pi}{4}$$

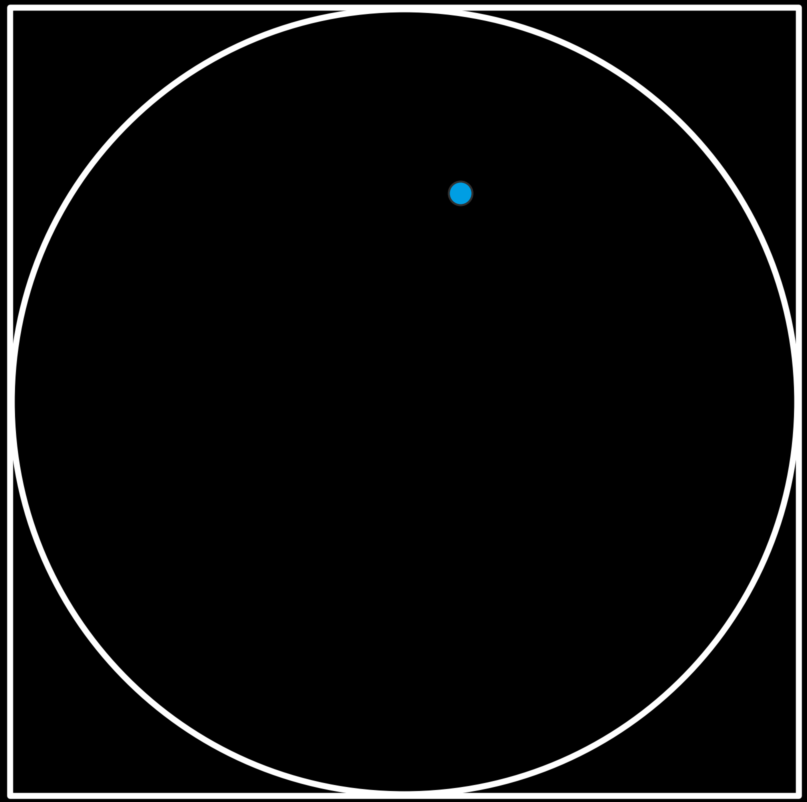
monte carlo simulation



$$\frac{\bigcirc}{\square} = \frac{\pi}{4}$$

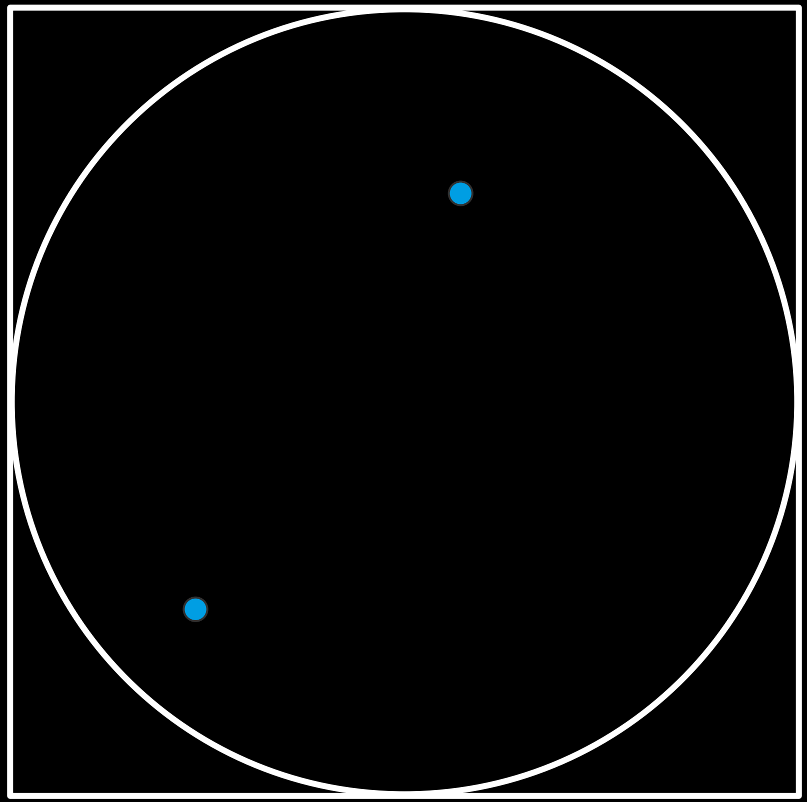


$$4 \frac{\bigcirc}{\square} = \pi$$



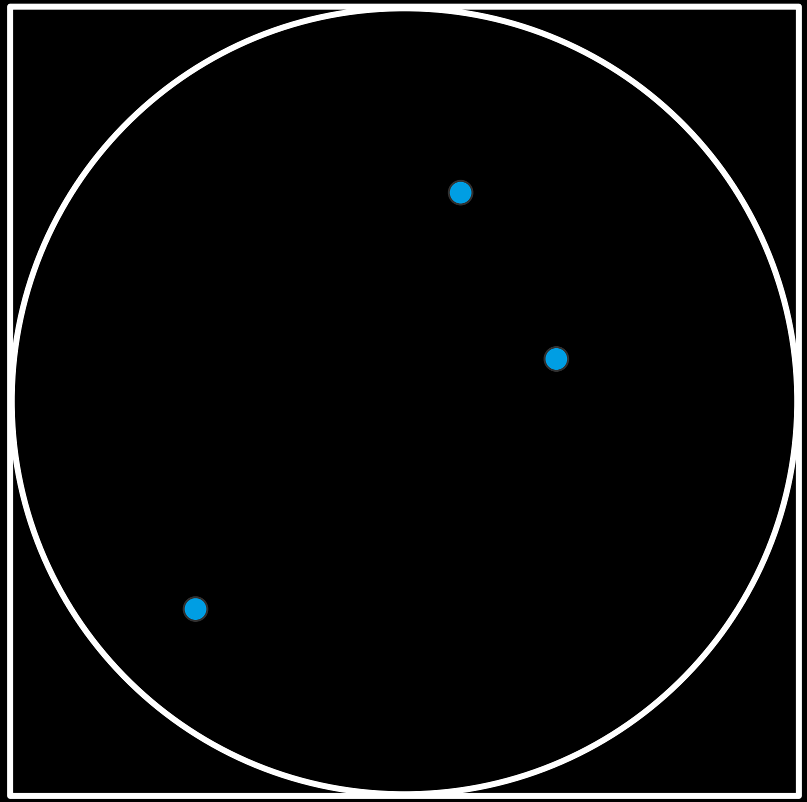
$$4 \frac{\bigcirc}{\square} = \pi$$

$$= 4$$



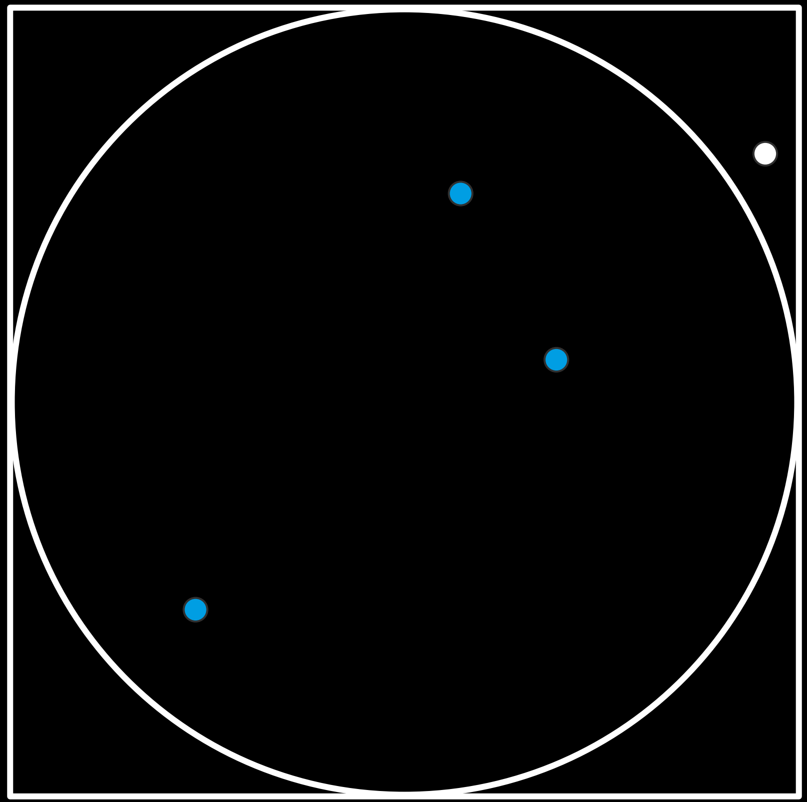
$$4\,\frac{\bigcirc}{\square} = \pi$$

$$= 4$$



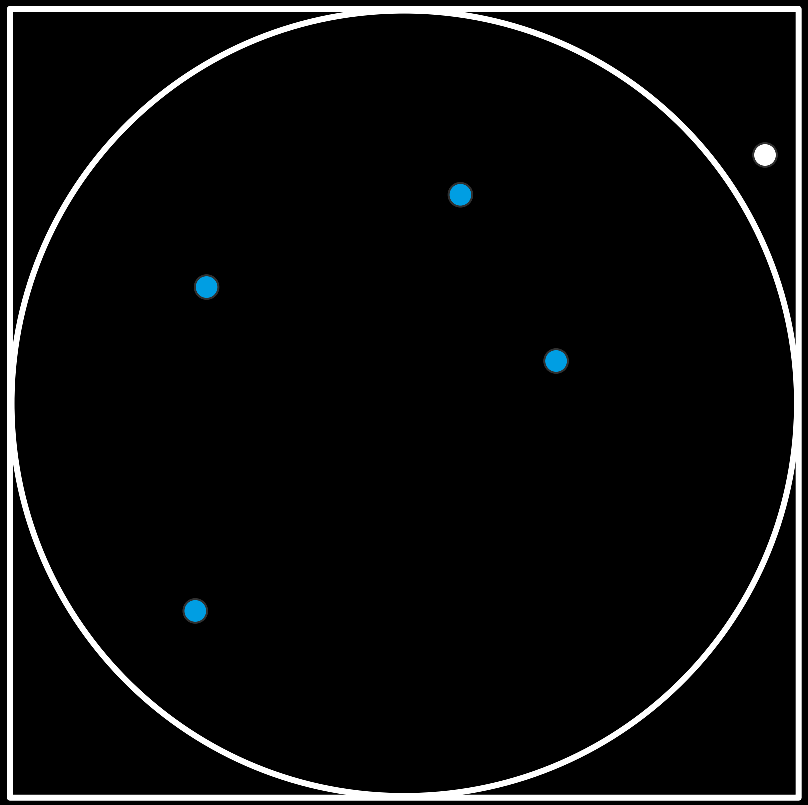
$$4\,\frac{\bigcirc}{\square} = \pi$$

$$= 4$$



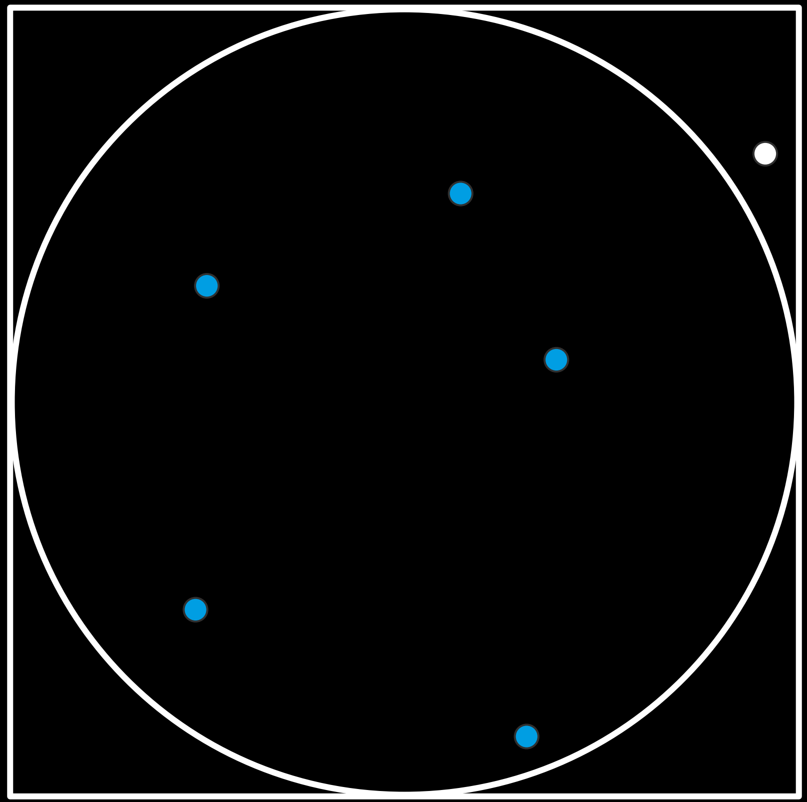
$$4\,\frac{\bigcirc}{\square} = \pi$$

$$= 3$$



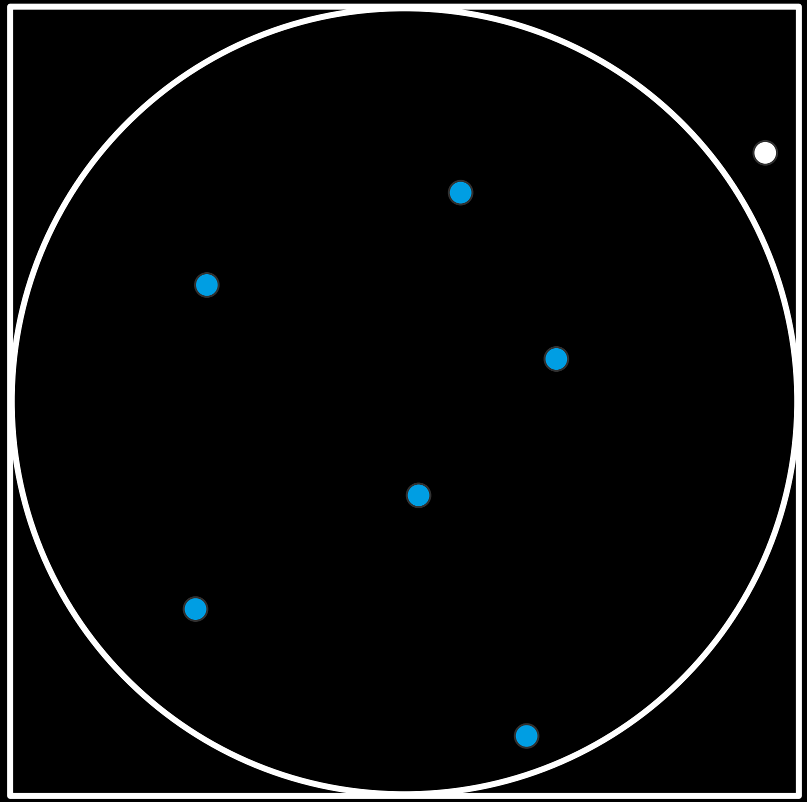
$$4\,\frac{\bigcirc}{\square} = \pi$$

$$= 3{,}2$$



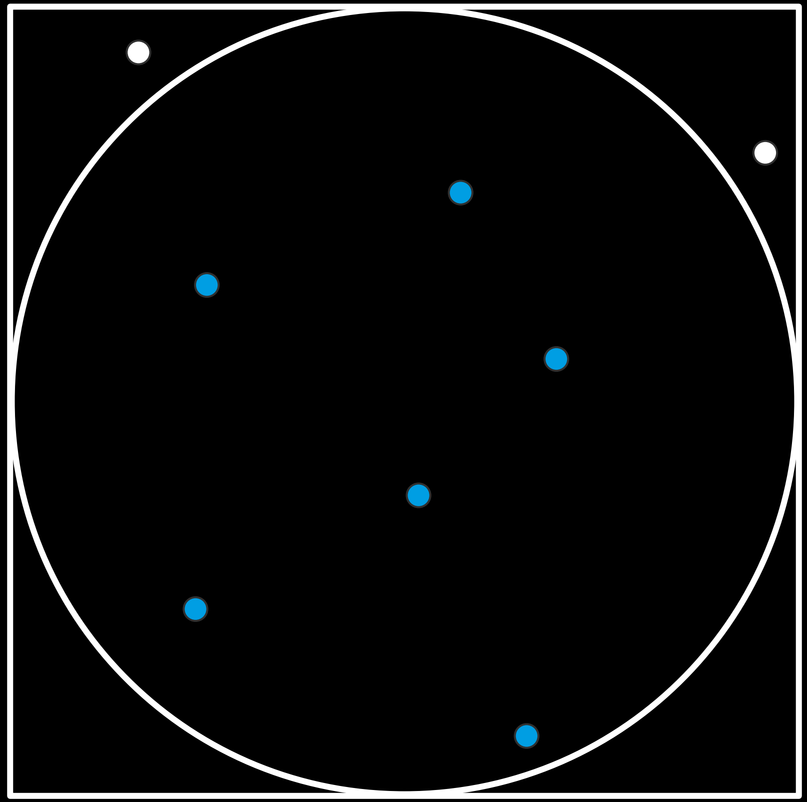
$$4 \, \frac{\bigcirc}{\square} = \pi$$

$$= 3{,}33$$



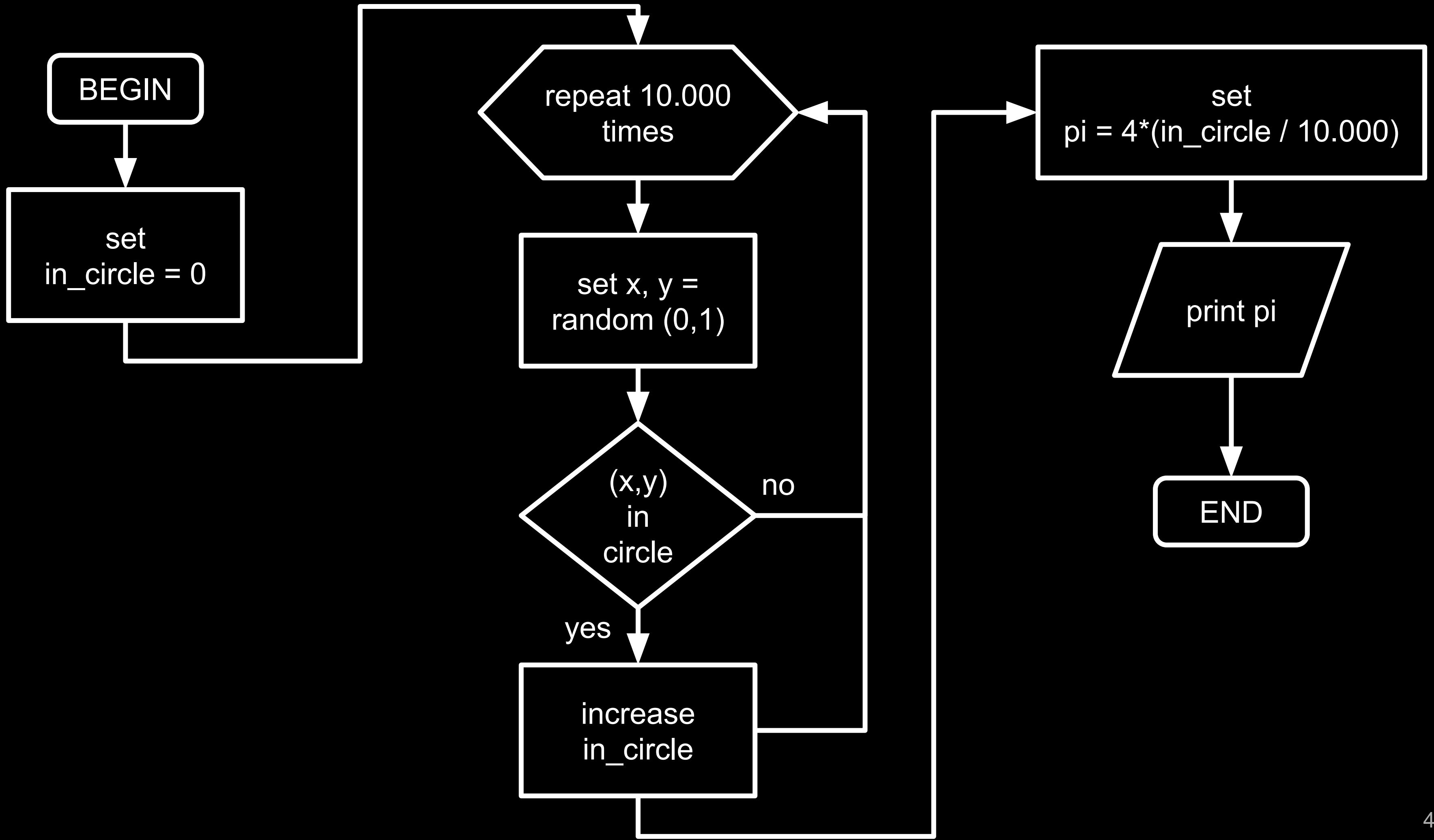
$$4\,\frac{\bigcirc}{\square} = \pi$$

$$= 3{,}43$$



$$4 \frac{\bigcirc}{\square} = \pi$$

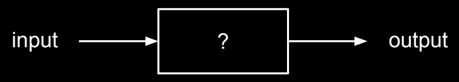
$$= 3$$


BEGIN
set
in_circle = 0
repeat 10.000
times
set x, y =
random (0,1)
(x,y)
in
circle
no
yes
increase
in_circle
set
pi = 4*(in_circle / 10.000)
print pi
END

gregory-leibniz series

$$\pi = 4\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

$$\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \ldots\right)$$



sorting

[9, 5, 2, 1, 4, 7]

bubble sort

repeatedly compare and swap elements
until done.

[9, 5, 2, 1, 4, 7]

[9, 5, 2, 1, 4, 7] ⟶ 9 > 5 ?

[9, 5, 2, 1, 4, 7] → 9 > 5 ? —yes→ [5, 9, 2, 1, 4, 7]

[9, 5, 2, 1, 4, 7] → 9 > 5 ? —yes→ [5, 9, 2, 1, 4, 7]

[5, 9, 2, 1, 4, 7] → 9 > 2 ? —yes→ [5, 2, 9, 1, 4, 7]

[9, 5, 2, 1, 4, 7] → 9 > 5 ? —yes→ [5, 9, 2, 1, 4, 7]

[5, 9, 2, 1, 4, 7] → 9 > 2 ? —yes→ [5, 2, 9, 1, 4, 7]

[5, 2, 9, 1, 4, 7] → 9 > 1 ? —yes→ [5, 2, 1, 9, 4, 7]

[9, 5, 2, 1, 4, 7]	→	9 > 5 ?	yes →	[5, 9, 2, 1, 4, 7]
[5, 9, 2, 1, 4, 7]	→	9 > 2 ?	yes →	[5, 2, 9, 1, 4, 7]
[5, 2, 9, 1, 4, 7]	→	9 > 1 ?	yes →	[5, 2, 1, 9, 4, 7]
[5, 2, 1, 9, 4, 7]	→	9 > 4 ?	yes →	[5, 2, 1, 4, 9, 7]

[9, 5, 2, 1, 4, 7] → 9 > 5 ? —yes→ [5, 9, 2, 1, 4, 7]

[5, 9, 2, 1, 4, 7] → 9 > 2 ? —yes→ [5, 2, 9, 1, 4, 7]

[5, 2, 9, 1, 4, 7] → 9 > 1 ? —yes→ [5, 2, 1, 9, 4, 7]

[5, 2, 1, 9, 4, 7] → 9 > 4 ? —yes→ [5, 2, 1, 4, 9, 7]

[5, 2, 1, 4, 9, 7] → 9 > 7 ? —yes→ [5, 2, 1, 4, 7, 9]

[5, 2, 1, 4, 7, 9] → 5 > 2 ? —yes→ [2, 5, 1, 4, 7, 9]

[5, 2, 1, 4, 7, 9] → 5 > 2 ? —yes→ [2, 5, 1, 4, 7, 9]

[2, 5, 1, 4, 7, 9] → 5 > 1 ? —yes→ [2, 1, 5, 4, 7, 9]

[5, 2, 1, 4, 7, 9] → 5 > 2 ? —yes→ [2, 5, 1, 4, 7, 9]

[2, 5, 1, 4, 7, 9] → 5 > 1 ? —yes→ [2, 1, 5, 4, 7, 9]

[2, 1, 5, 4, 7, 9] → 5 > 4 ? —yes→ [2, 1, 4, 5, 7, 9]

[5, 2, 1, 4, 7, 9] → 5 > 2 ? —yes→ [2, 5, 1, 4, 7, 9]

[2, 5, 1, 4, 7, 9] → 5 > 1 ? —yes→ [2, 1, 5, 4, 7, 9]

[2, 1, 5, 4, 7, 9] → 5 > 4 ? —yes→ [2, 1, 4, 5, 7, 9]

[2, 1, 4, 5, 7, 9] → 5 > 7 ? —no→ [2, 1, 4, 5, 7, 9]

[2, 1, 4, 5, 7, 9] → 2 > 1 ? —yes→ [1, 2, 4, 5, 7, 9]

[2, 1, 4, 5, 7, 9] → 2 > 1 ? → (yes) [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] → 2 > 4 ? → (no) [1, 2, 4, 5, 7, 9]

[2, 1, 4, 5, 7, 9] → 2 > 1 ? → yes → [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] → 2 > 4 ? → no → [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] → 4 > 5 ? → no → [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] ⟶ 1 > 2 ? —no⟶ [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] → 1 > 2 ? —no→ [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] → 2 > 4 ? —no→ [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] → 1 > 2 ? —no→ [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] → 1 > 2 ? —no→ [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] DONE!

selection sort

find the smallest element and move it to front. repeat for the rest of the elements.

move to front

[9, 5, 2, 1, 4, 7] → [1, 5, 9, 2, 4, 7]


move to front
[9, 5, 2, 1, 4, 7]
[1, 5, 9, 2, 4, 7]
[1, 5, 9, 2, 4, 7]
[1, 2, 5, 9, 4, 7]

move to front

[9, 5, 2, 1, 4, 7] → [1, 5, 9, 2, 4, 7]

[1, 5, 9, 2, 4, 7] → [1, 2, 5, 9, 4, 7]

[1, 2, 5, 9, 4, 7] → [1, 2, 4, 5, 9, 7]

move to front

[9, 5, 2, 1, 4, 7] → [1, 5, 9, 2, 4, 7]

[1, 5, 9, 2, 4, 7] → [1, 2, 5, 9, 4, 7]

[1, 2, 5, 9, 4, 7] → [1, 2, 4, 5, 9, 7]

[1, 2, 4, 5, 9, 7] → [1, 2, 4, 5, 9, 7]

move to front

[9, 5, 2, 1, 4, 7] → [1, 5, 9, 2, 4, 7]

[1, 5, 9, 2, 4, 7] → [1, 2, 5, 9, 4, 7]

[1, 2, 5, 9, 4, 7] → [1, 2, 4, 5, 9, 7]

[1, 2, 4, 5, 9, 7] → [1, 2, 4, 5, 9, 7]

[1, 2, 4, 5, 9, 7] → [1, 2, 4, 5, 7, 9]

move to front

[9, 5, 2, 1, 4, 7] → [1, 5, 9, 2, 4, 7]

[1, 5, 9, 2, 4, 7] → [1, 2, 5, 9, 4, 7]

[1, 2, 5, 9, 4, 7] → [1, 2, 4, 5, 9, 7]

[1, 2, 4, 5, 9, 7] → [1, 2, 4, 5, 9, 7]

[1, 2, 4, 5, 9, 7] → [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] → [1, 2, 4, 5, 7, 9]

merge sort

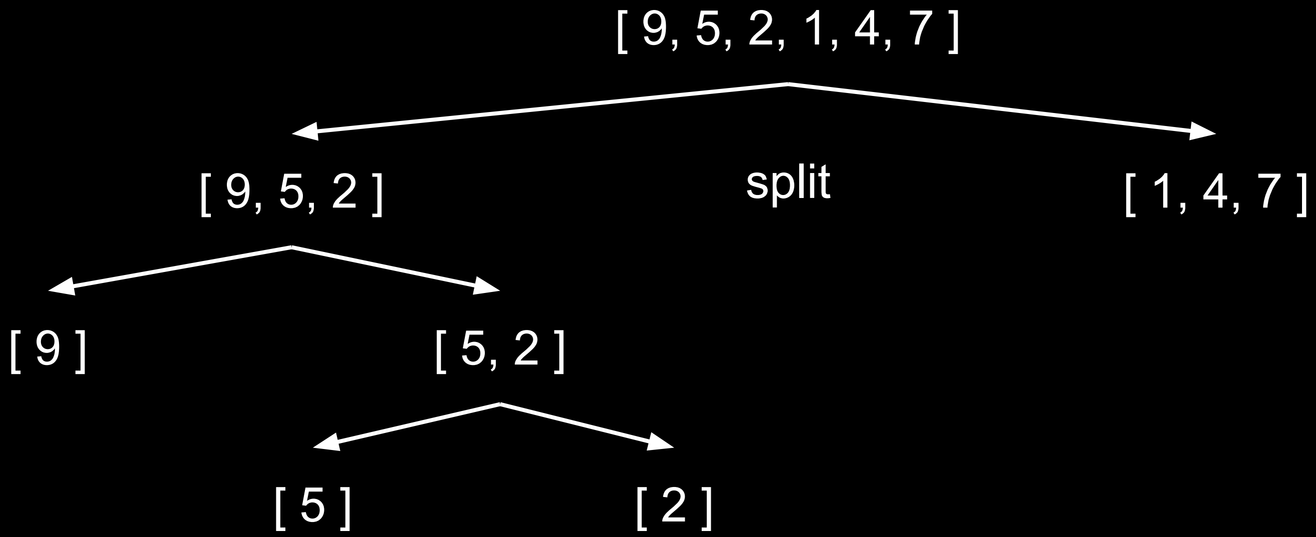
```
divide the elements recursively in two
halves until only one element is left.
then merge the sorted halves back together.
```

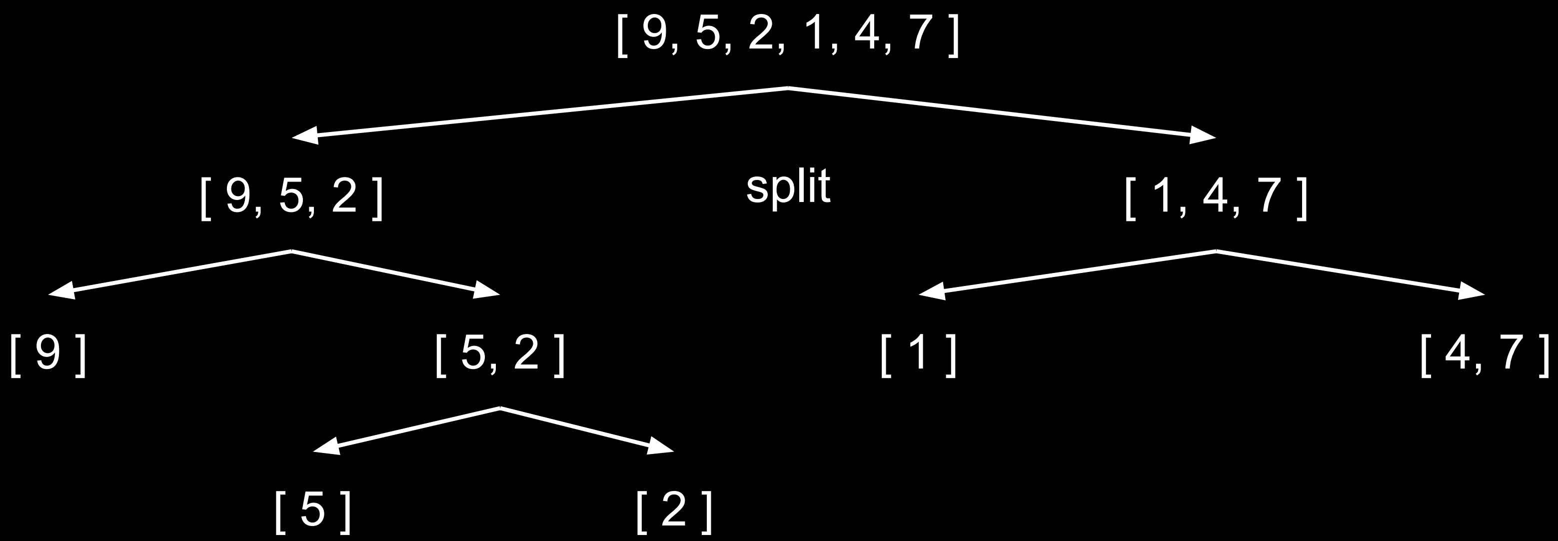
divide the elements recursively in two halves until only one element is left. then merge the sorted halves back together.

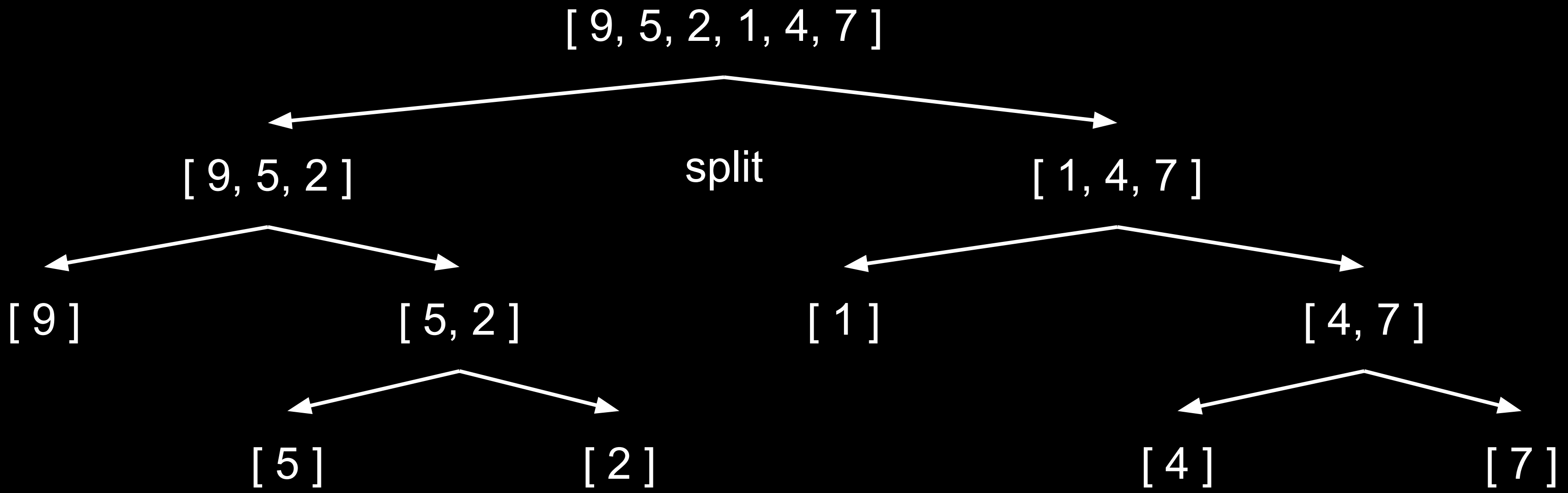
[9, 5, 2, 1, 4, 7]

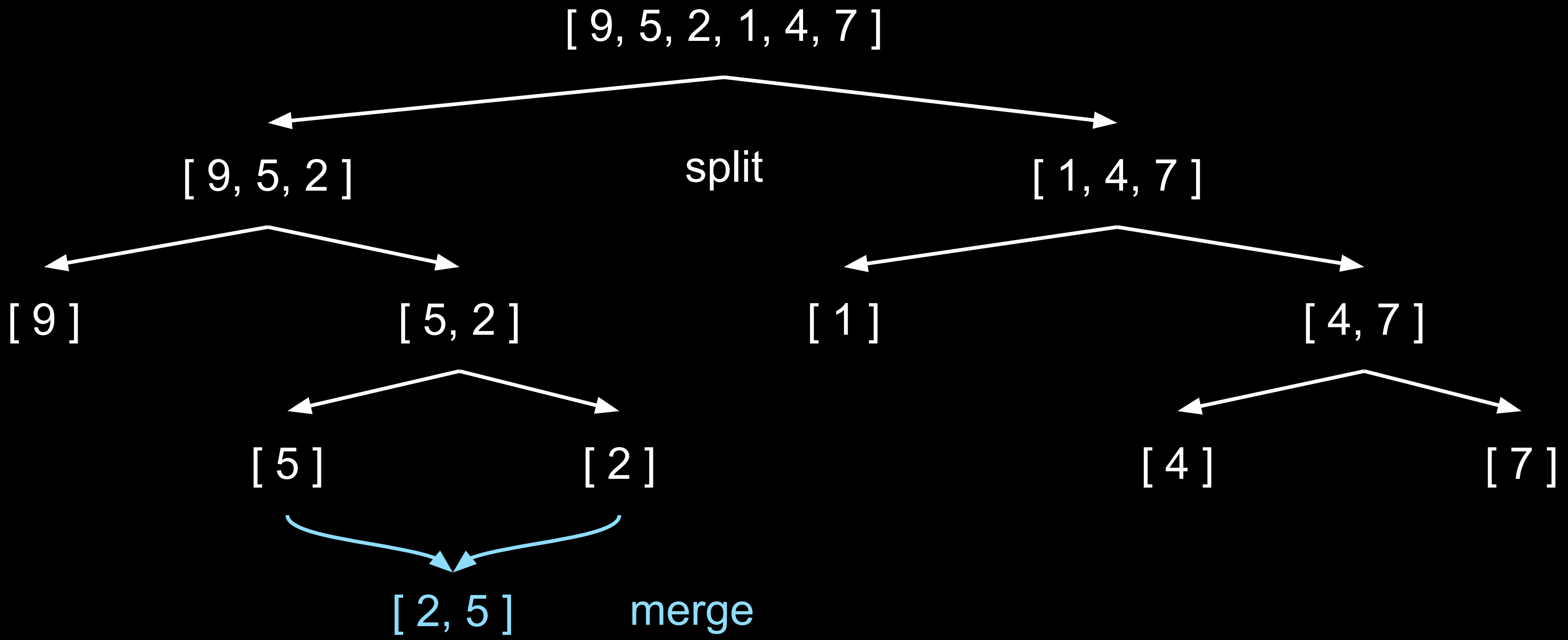
[9, 5, 2]   split   [1, 4, 7]

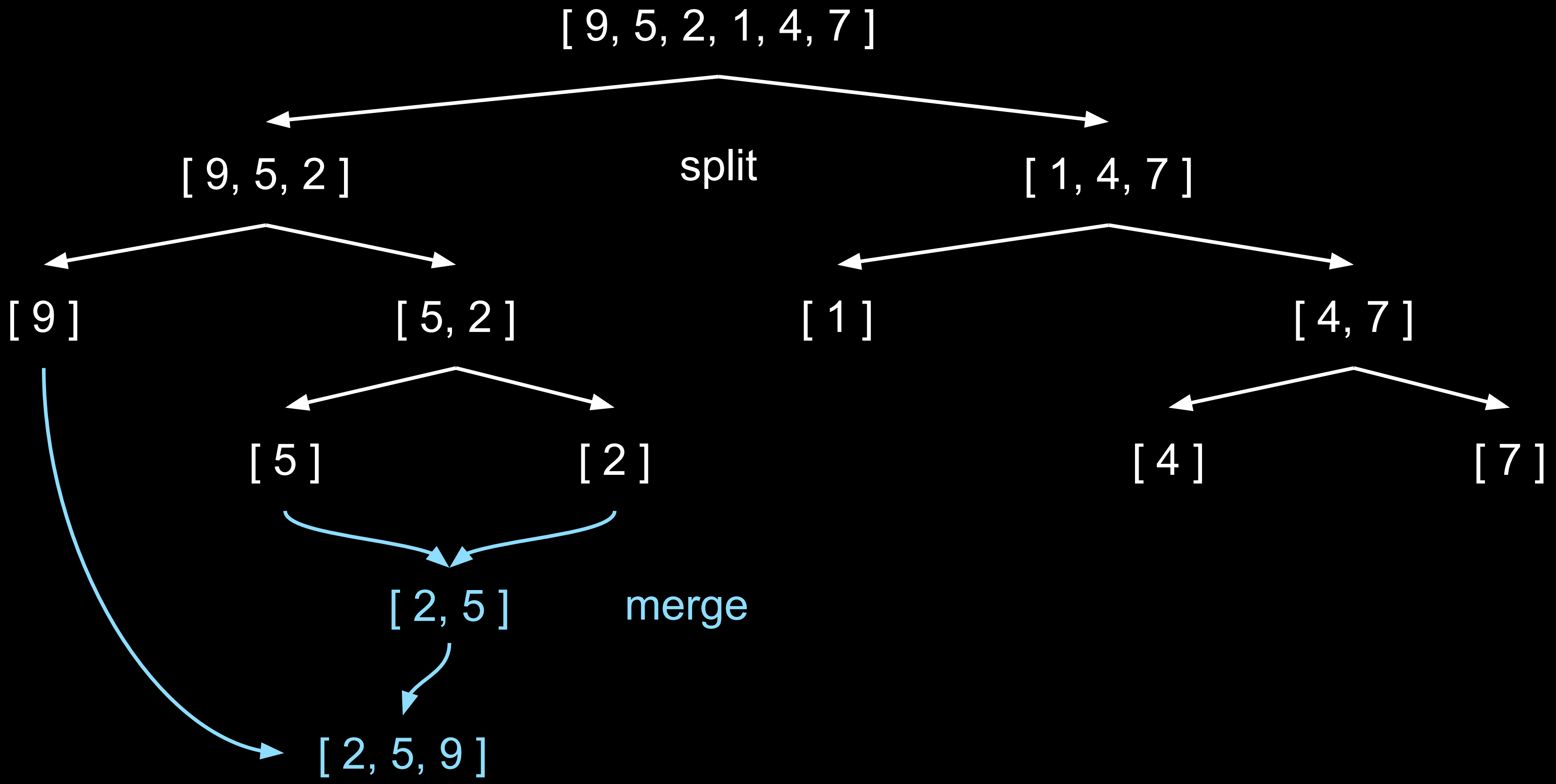

[9, 5, 2, 1, 4, 7]
split
[9, 5, 2]
[1, 4, 7]
[9]
[5, 2]

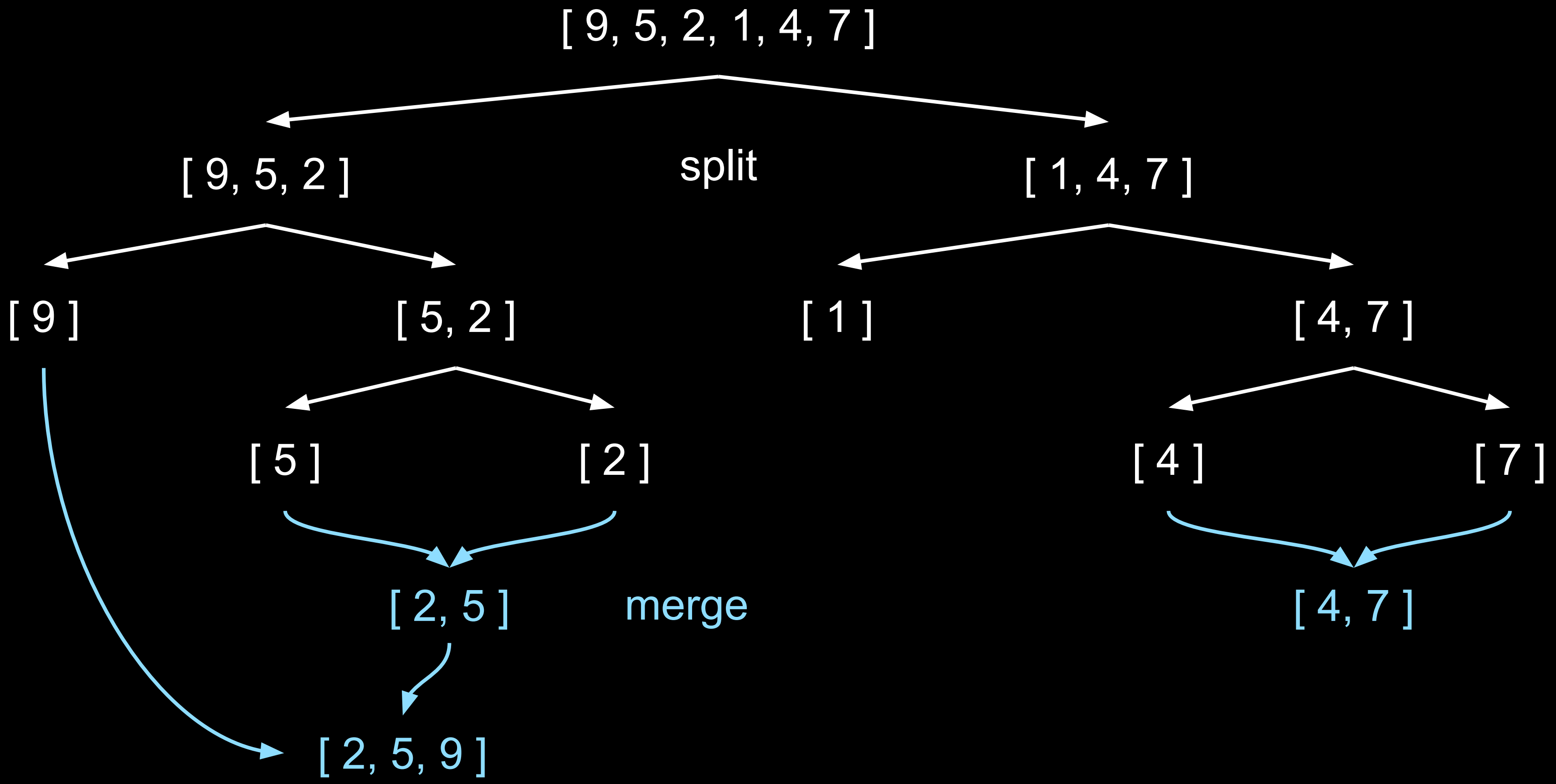

[9, 5, 2, 1, 4, 7]
split
[9, 5, 2]
[1, 4, 7]
[9]
[5, 2]
[5]
[2]

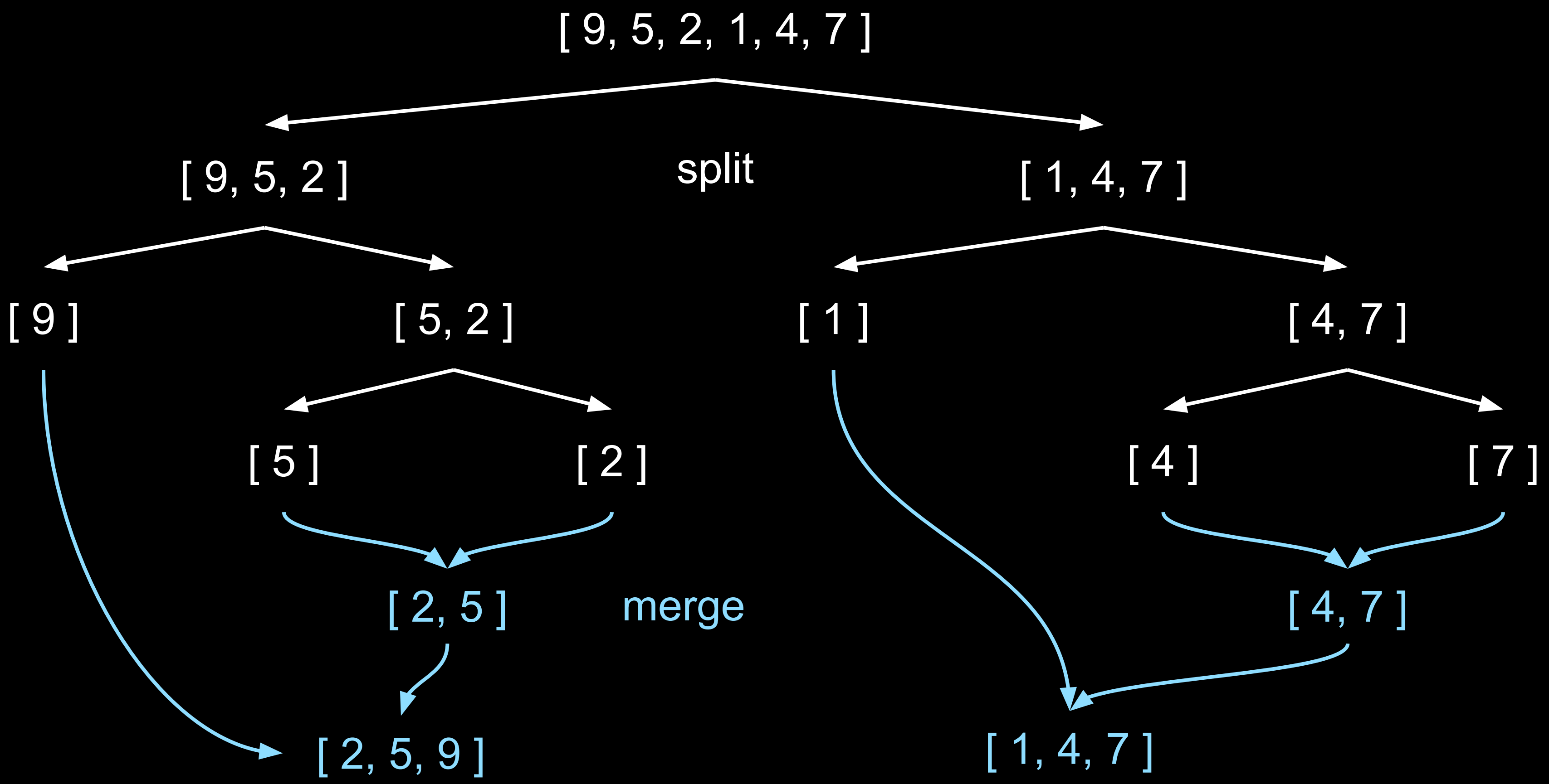

[9, 5, 2, 1, 4, 7]
split
[9, 5, 2]
[1, 4, 7]
[9]
[5, 2]
[1]
[4, 7]
[5]
[2]

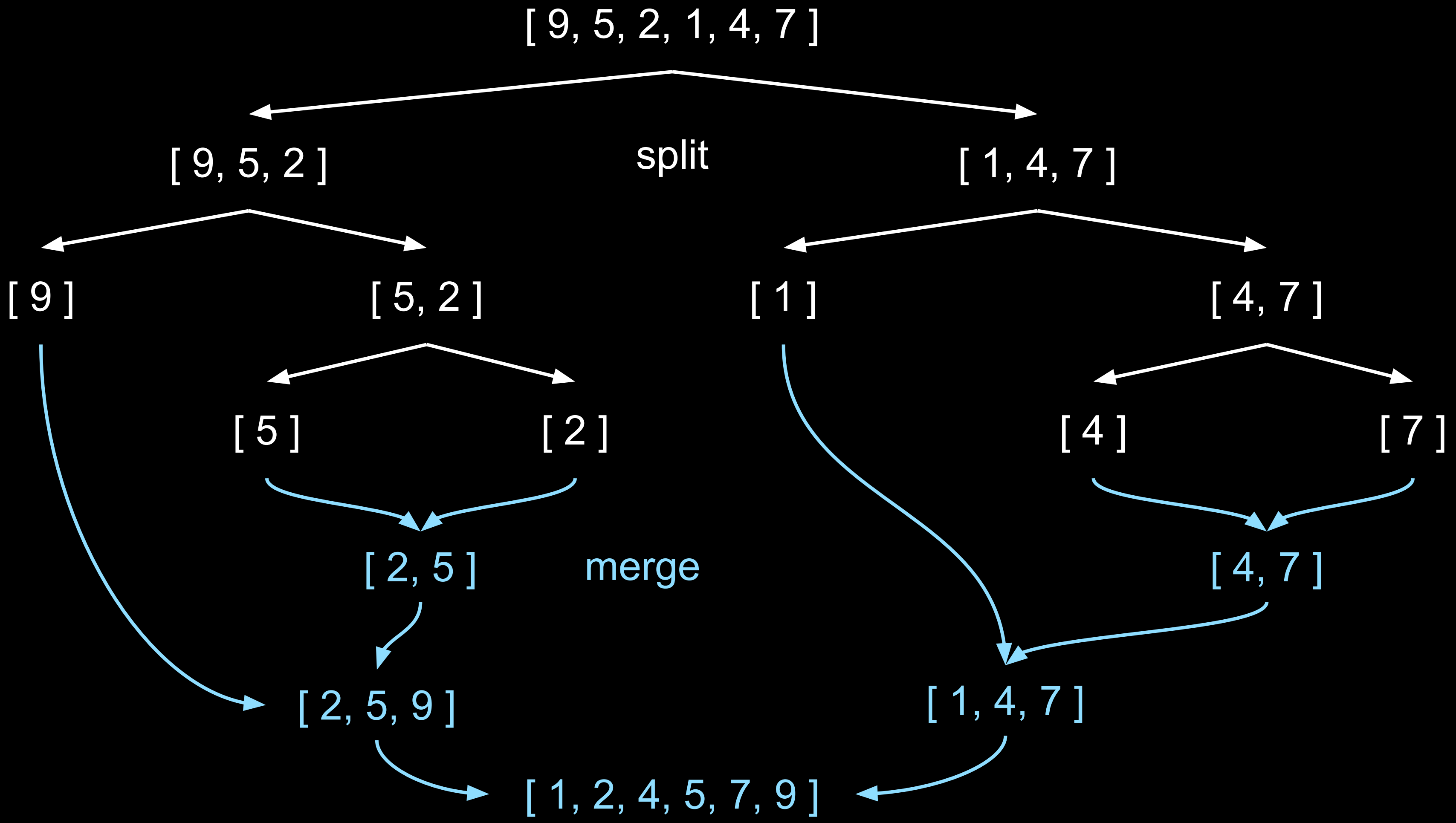

[9, 5, 2, 1, 4, 7]
split
[9, 5, 2]
[1, 4, 7]
[9]
[5, 2]
[1]
[4, 7]
[5]
[2]
[4]
[7]


[9, 5, 2, 1, 4, 7]
split
[9, 5, 2]
[1, 4, 7]
[9]
[5, 2]
[1]
[4, 7]
[5]
[2]
[4]
[7]
[2, 5]
merge


[9, 5, 2, 1, 4, 7]
split
[9, 5, 2]
[1, 4, 7]
[9]
[5, 2]
[1]
[4, 7]
[5]
[2]
[4]
[7]
[2, 5]
merge
[2, 5, 9]

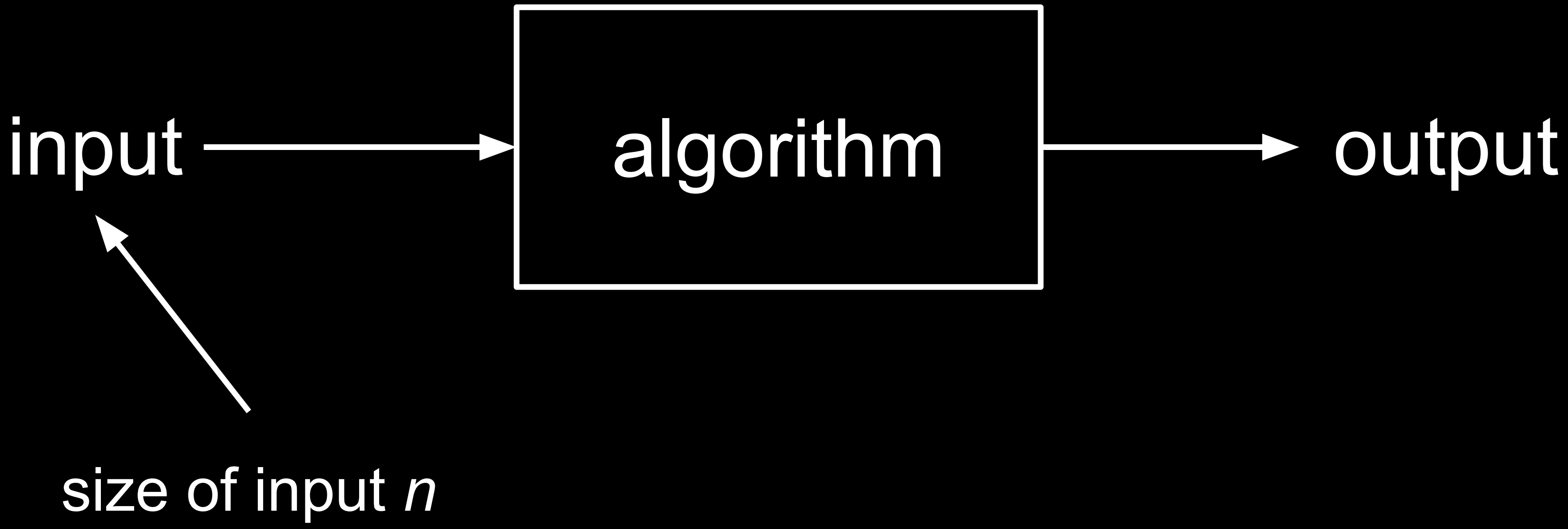

[9, 5, 2, 1, 4, 7]
split
[9, 5, 2]
[1, 4, 7]
[9]
[5, 2]
[1]
[4, 7]
[5]
[2]
[4]
[7]
[2, 5]
merge
[4, 7]
[2, 5, 9]

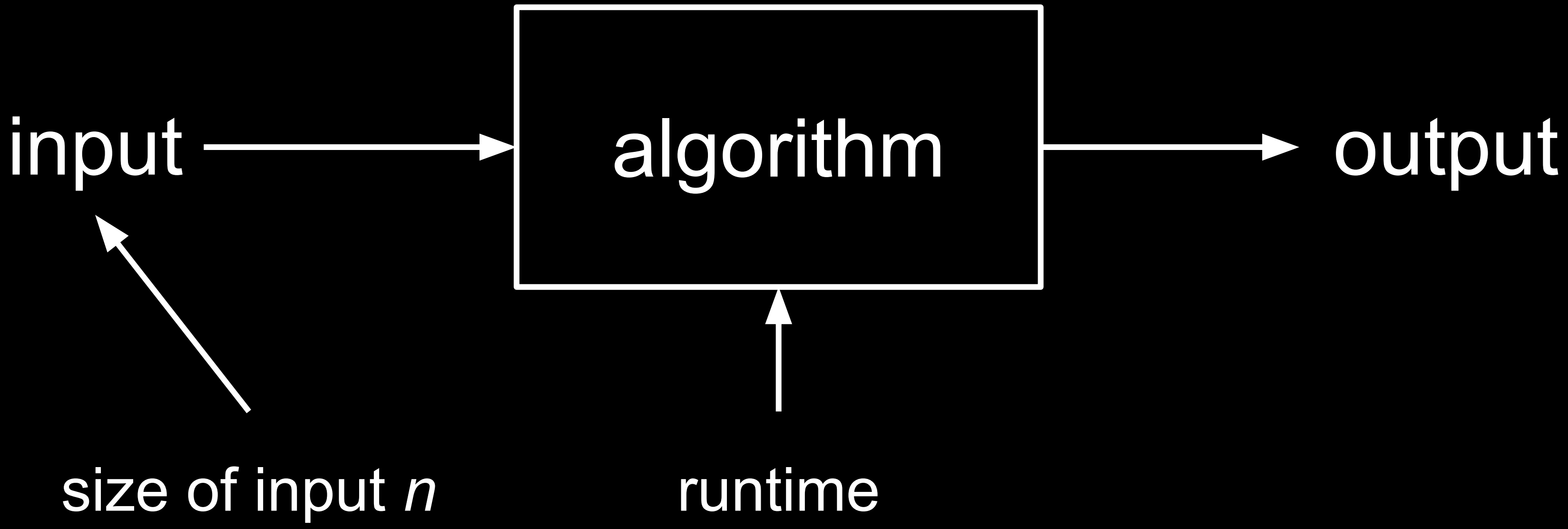

[9, 5, 2, 1, 4, 7]
[9, 5, 2]
split
[1, 4, 7]
[9]
[5, 2]
[1]
[4, 7]
[5]
[2]
[4]
[7]
[2, 5]
merge
[4, 7]
[2, 5, 9]
[1, 4, 7]



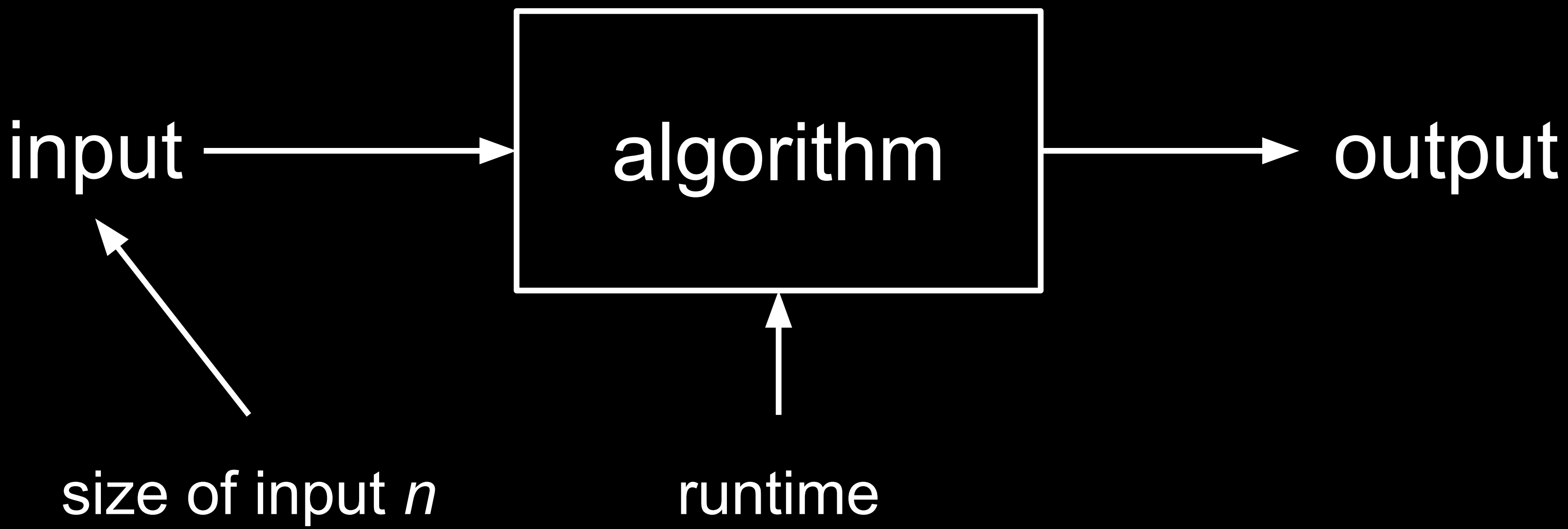
[9, 5, 2, 1, 4, 7]
split
[9, 5, 2]
[1, 4, 7]
[9]
[5, 2]
[1]
[4, 7]
[5]
[2]
[4]
[7]
[2, 5]
merge
[4, 7]
[2, 5, 9]
[1, 4, 7]
[1, 2, 4, 5, 7, 9]

complexity

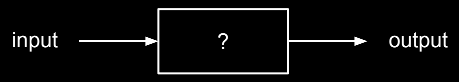

input
algorithm
output
size of input n


input
algorithm
output
size of input *n*
runtime

O(n)



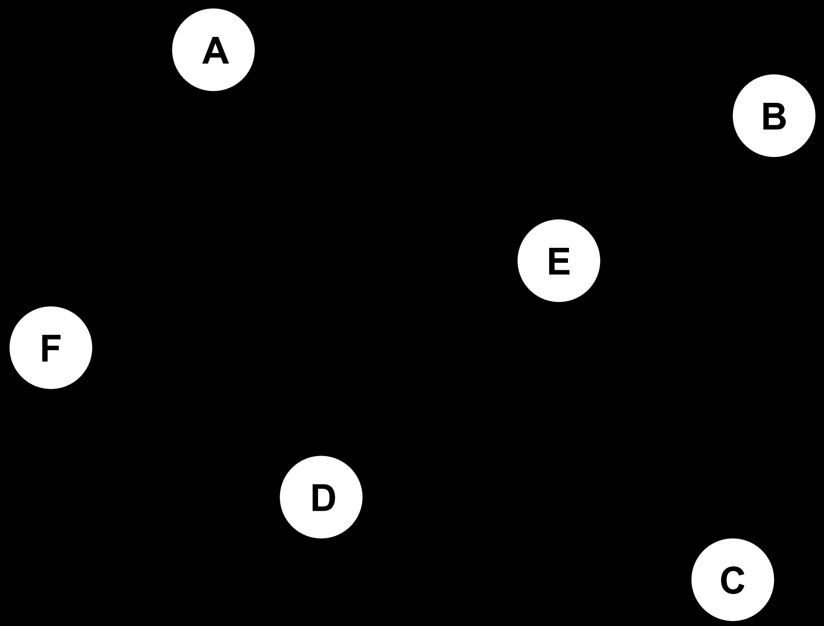
O(1)	runtime is constant and independent of problem size
O($\log_2$ n)	runtime is determined by the logarithm of problem size
O(n)	runtime is linear to problem size
O(n^2)	runtime grows quadratically with the size of the problem
O(n^3)	runtime grows cubically with the size of the problem
O(2^n)	runtime grows exponentially with the size of the problem
O(n!)	runtime grows factorially with the size of the problem



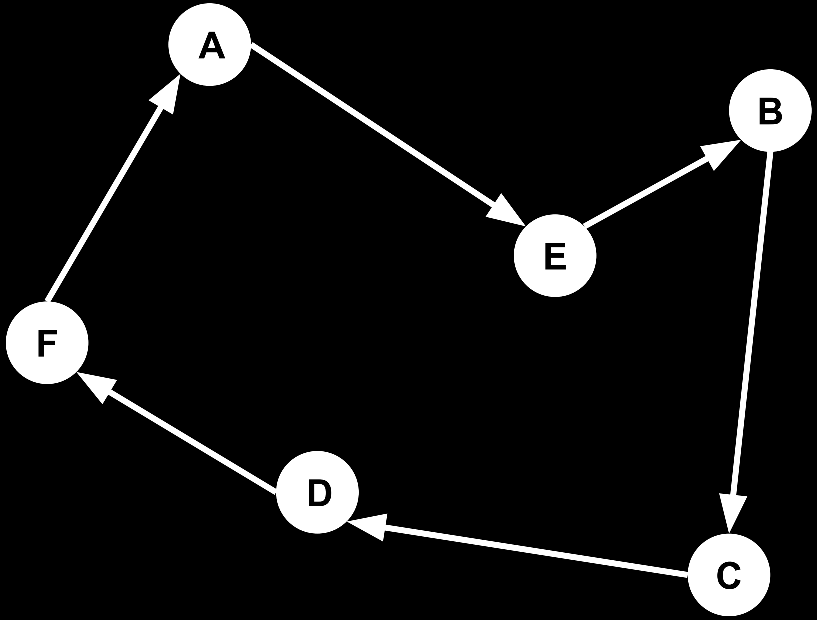
optimization

traveling salesmen

shortest tour?



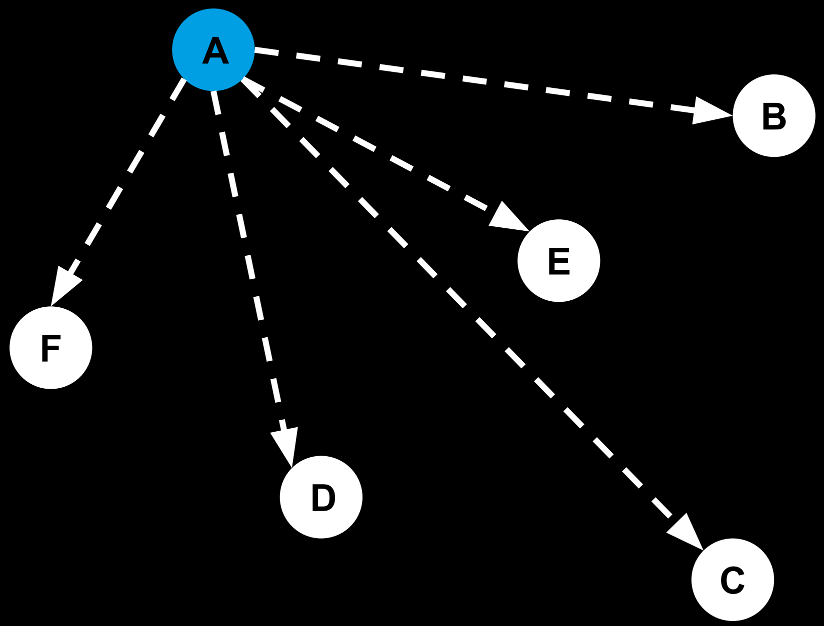
shortest tour?



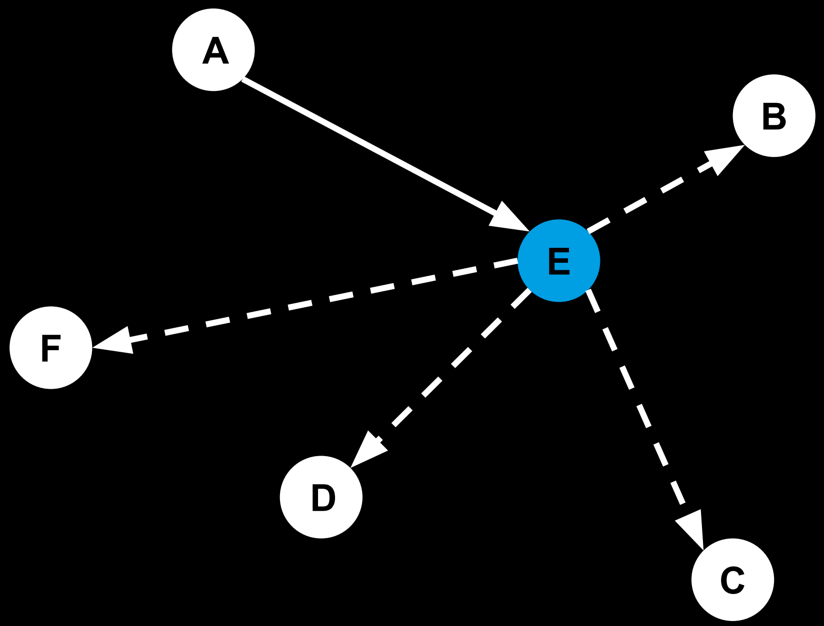
brute force

$$O(n) = n!$$

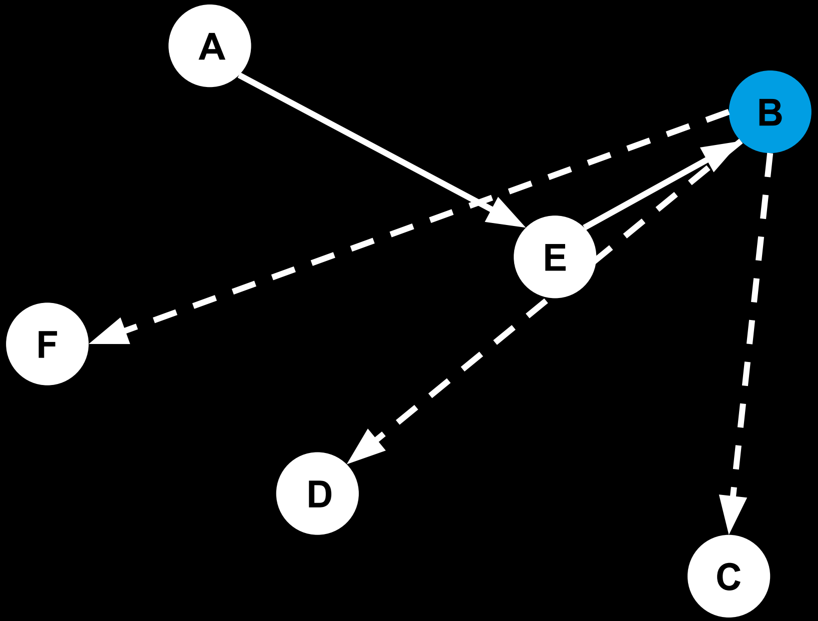
5 possible cities



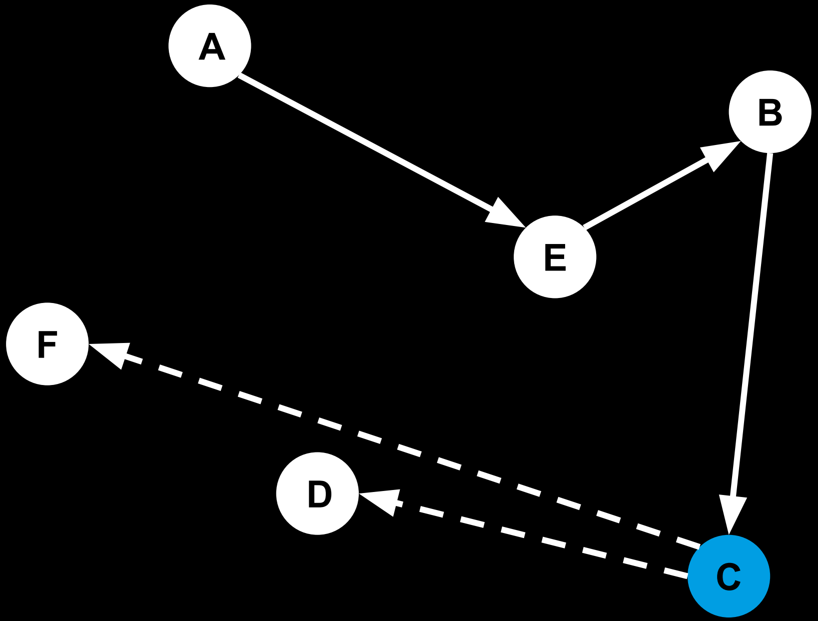
~~5~~ 4 possible cities



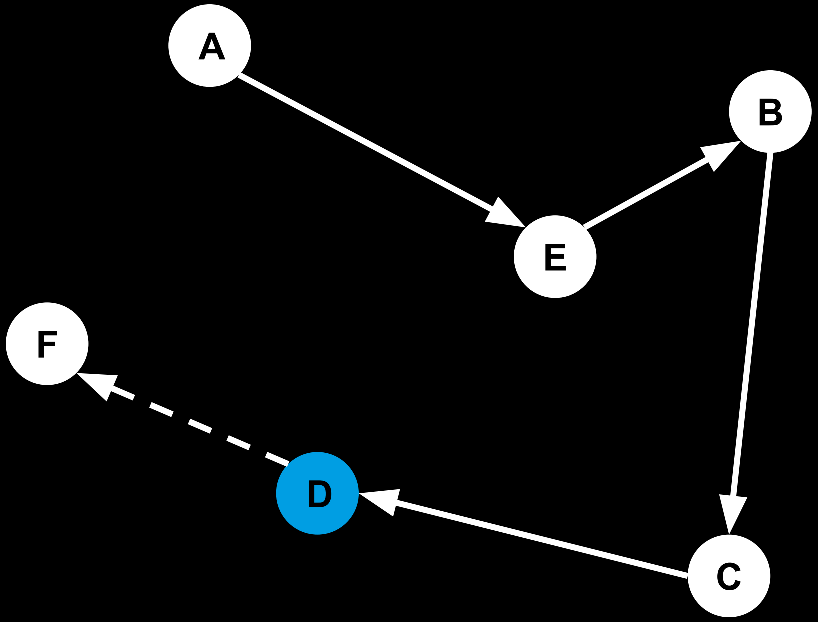
~~5~~ ~~4~~ 3 possible cities



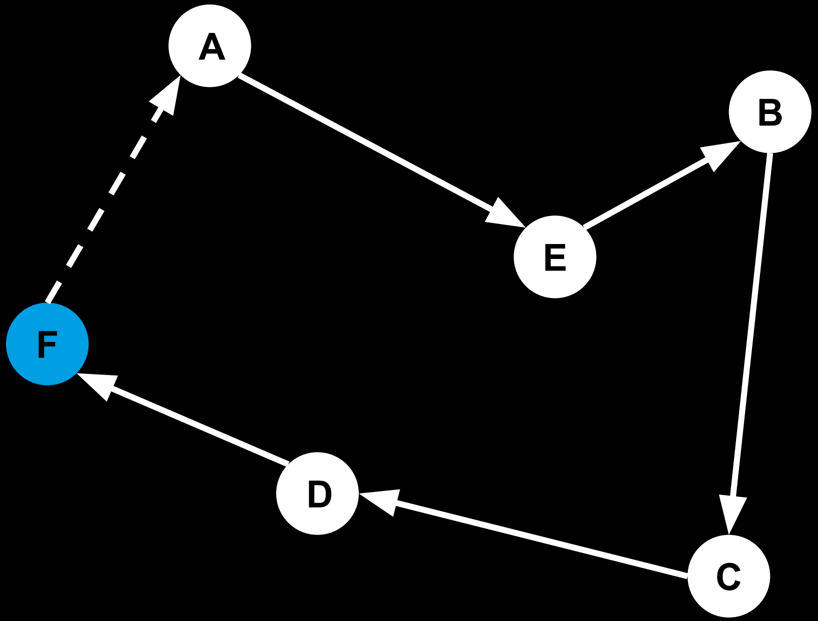
~~5~~ ~~4~~ ~~3~~ 2 possible cities



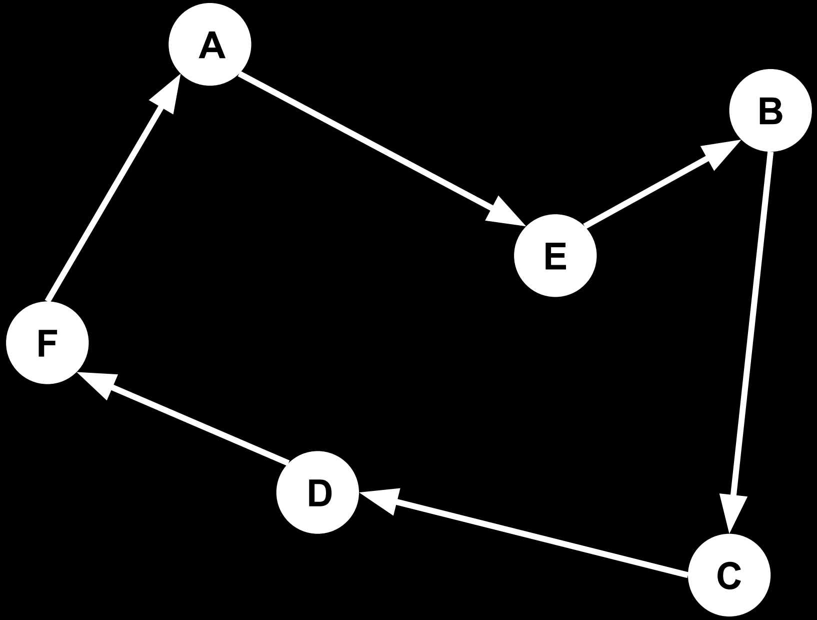
~~5~~ ~~4~~ ~~3~~ ~~2~~ 1 possible city



~~5~~ ~~4~~ ~~3~~ ~~2~~ ~~1~~ return nome



~~5~~ ~~4~~ ~~3~~ ~~2~~ ~~1~~ return nome



$$O(n) = n!$$

$$n = 5$$

$$O(n) = n!$$

$$n = 5$$

$$N = 5 * 4 * 3 * 2 * 1$$

$$O(n) = n!$$

$$n = 5$$

$$N = 5 * 4 * 3 * 2 * 1$$

$$= 120$$

$n = 10$

$n = 20$

$n = 30$

$n = 25$

brute force takes longer than the universe is old

$n = 60$

more possible routes than atoms in the universe

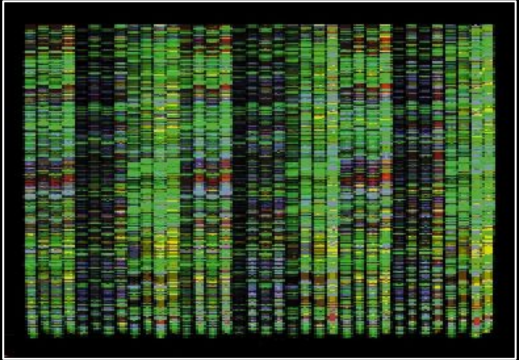


Image source: IEEE

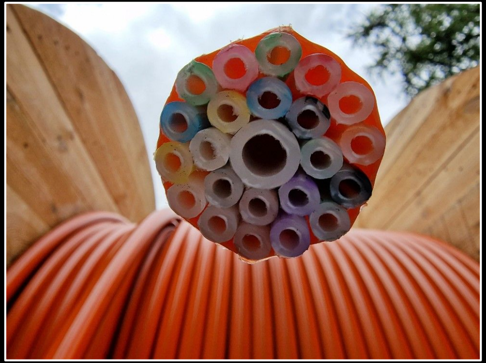
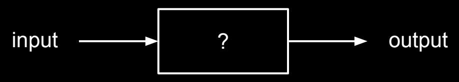


Image source: VDI Nachrichten

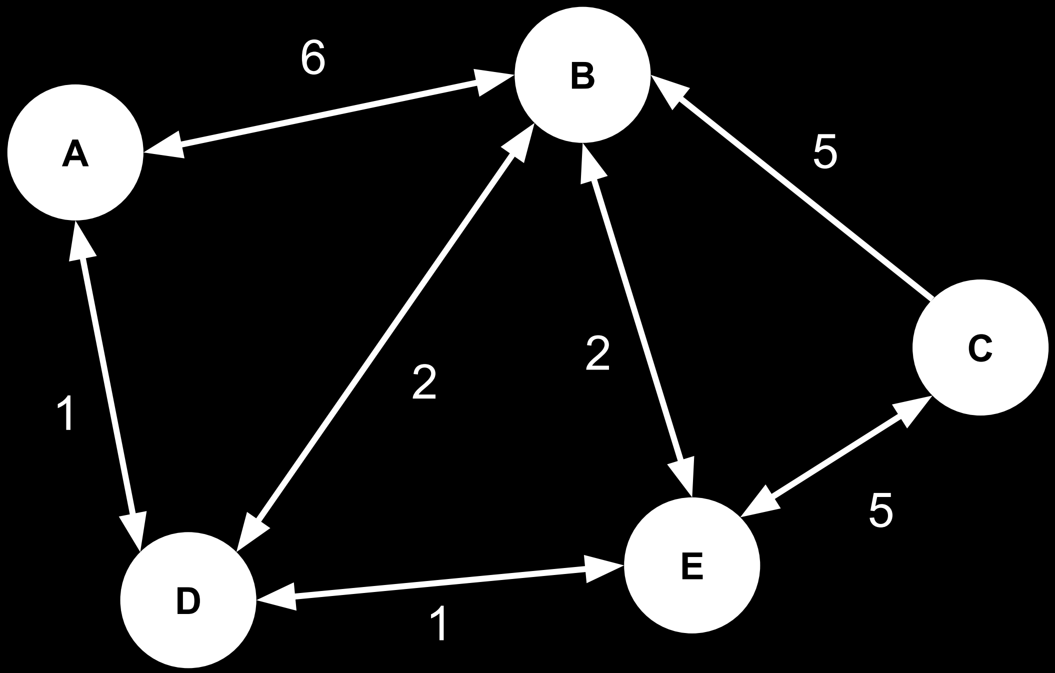
Image source: Wikimedia

Image source: https://github.com/meiyi1986/tutorials/blob/master/notebooks/img/pcb-drilling.jpeg

Image source: IAS Observatory



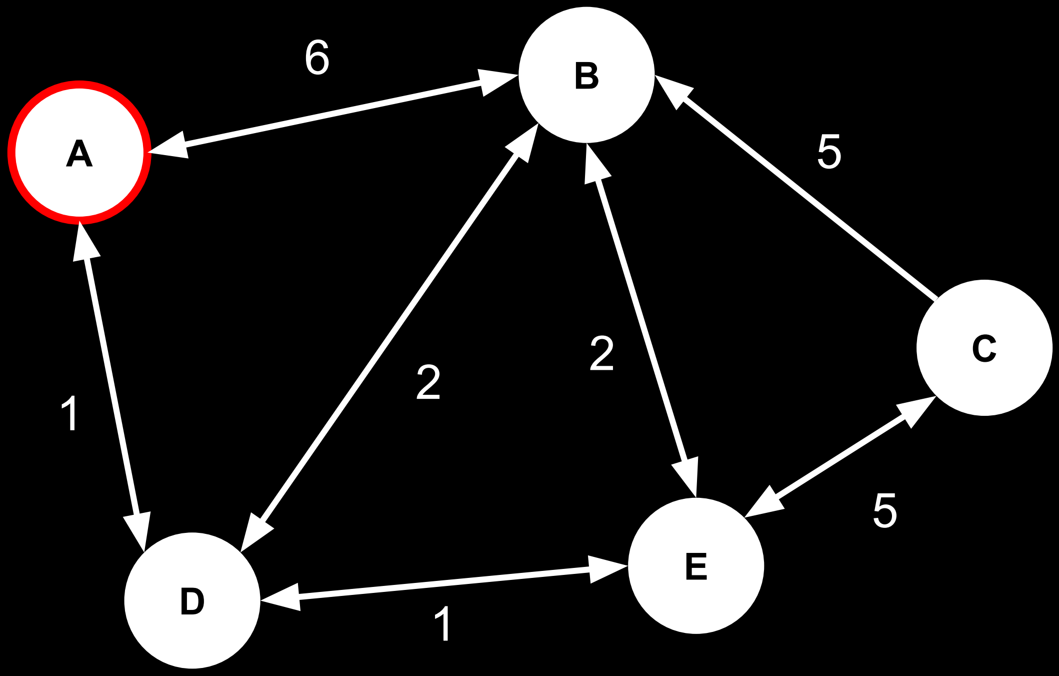
shortest paths


A
B
C
D
E
6
5
2
2
1
5
1

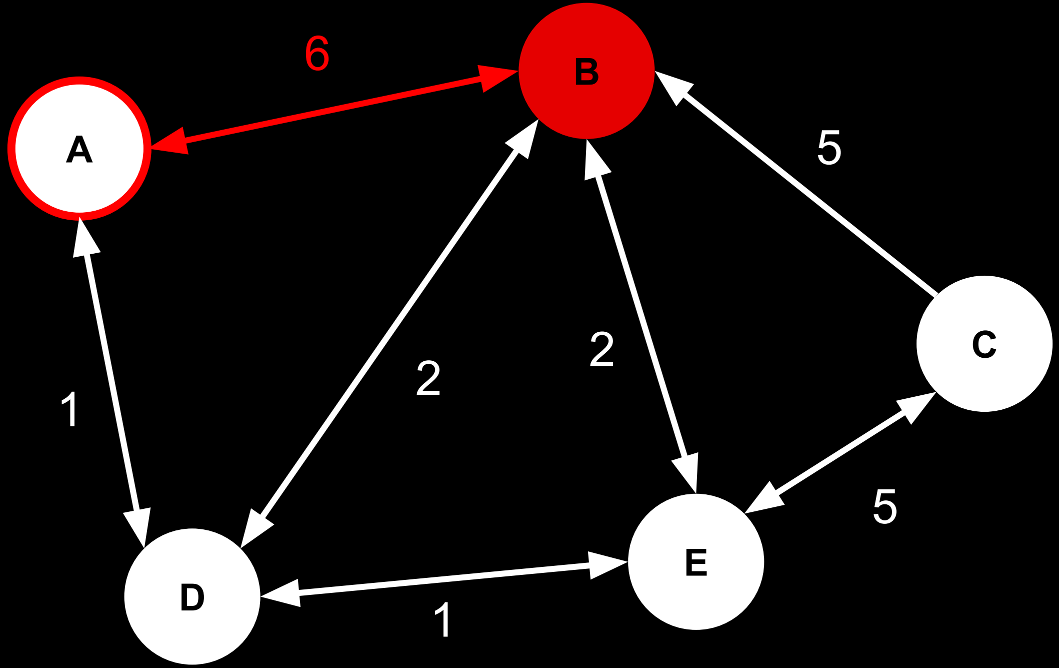
dijkstra's algorithm

Distances: A = 0, B = ∞, C = ∞, D = ∞, E = ∞

Set A as first location



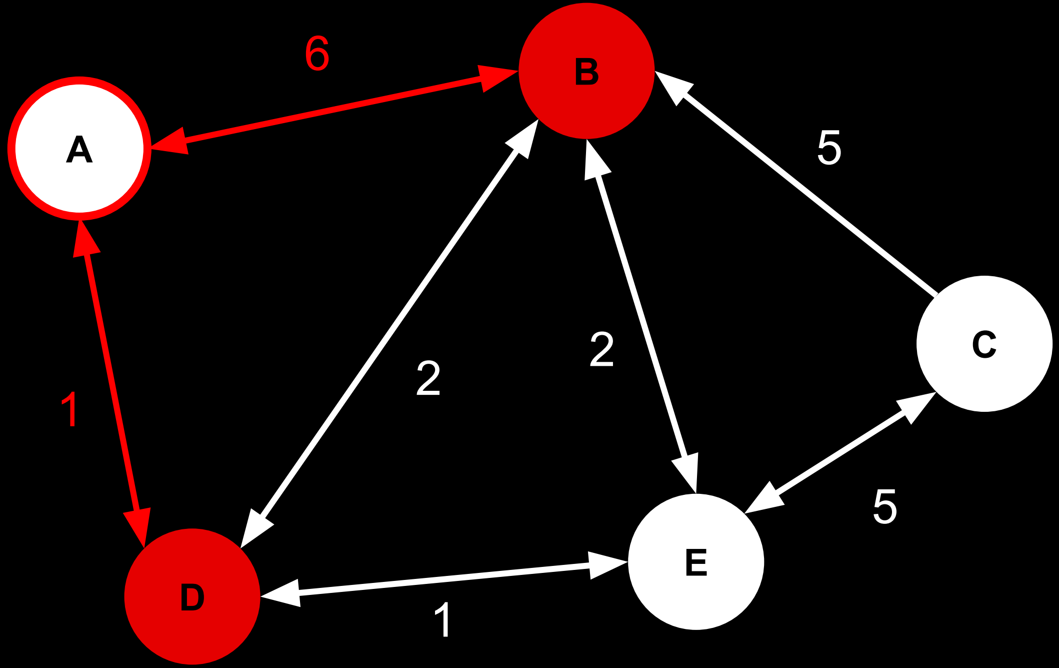
Distances: A = 0, B = 6, C = ∞, D = ∞, E = ∞



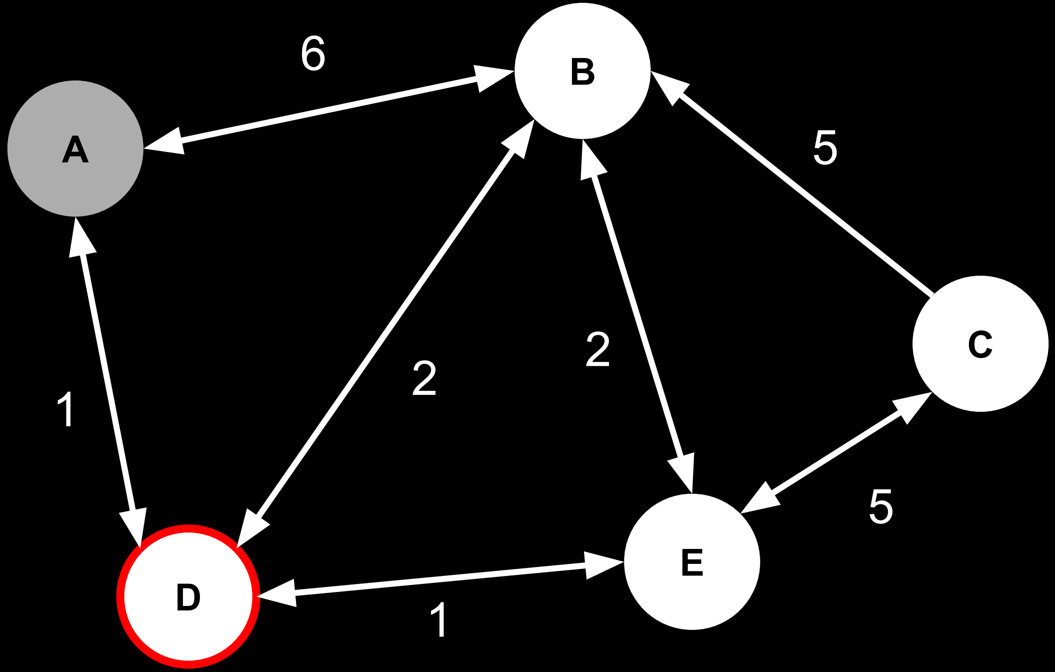
Update distance
to B to 6

Distances: A = 0, B = 6, C = ∞, D = 1, E = ∞

Update distance to D to 1



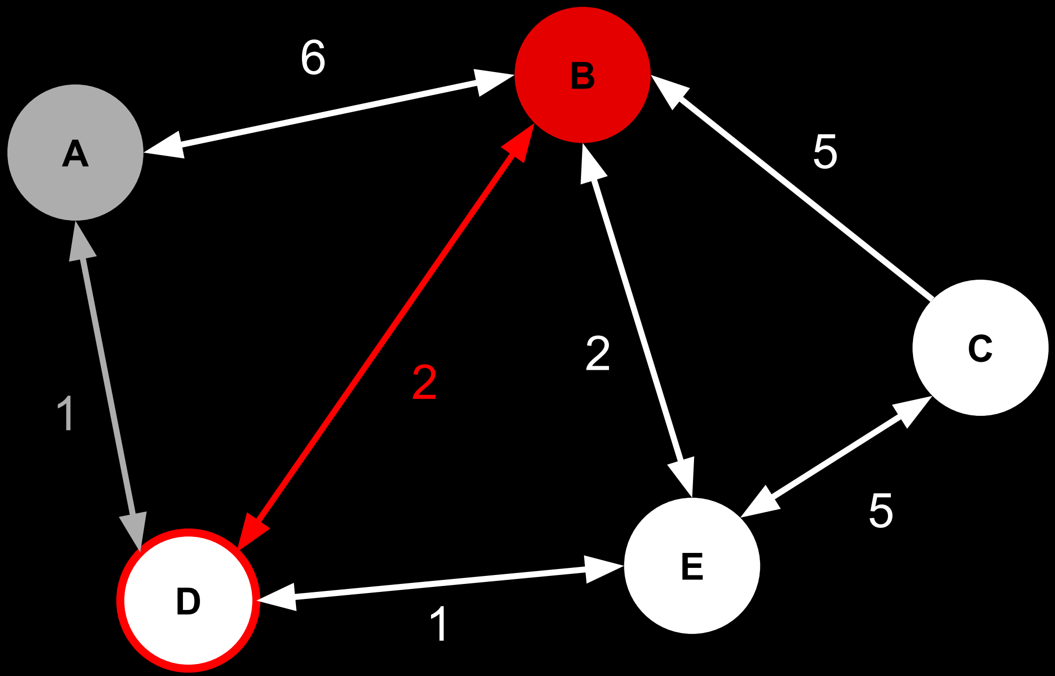
Distances: A = 0, B = 6, C = ∞, D = 1, E = ∞



Move to D as next location

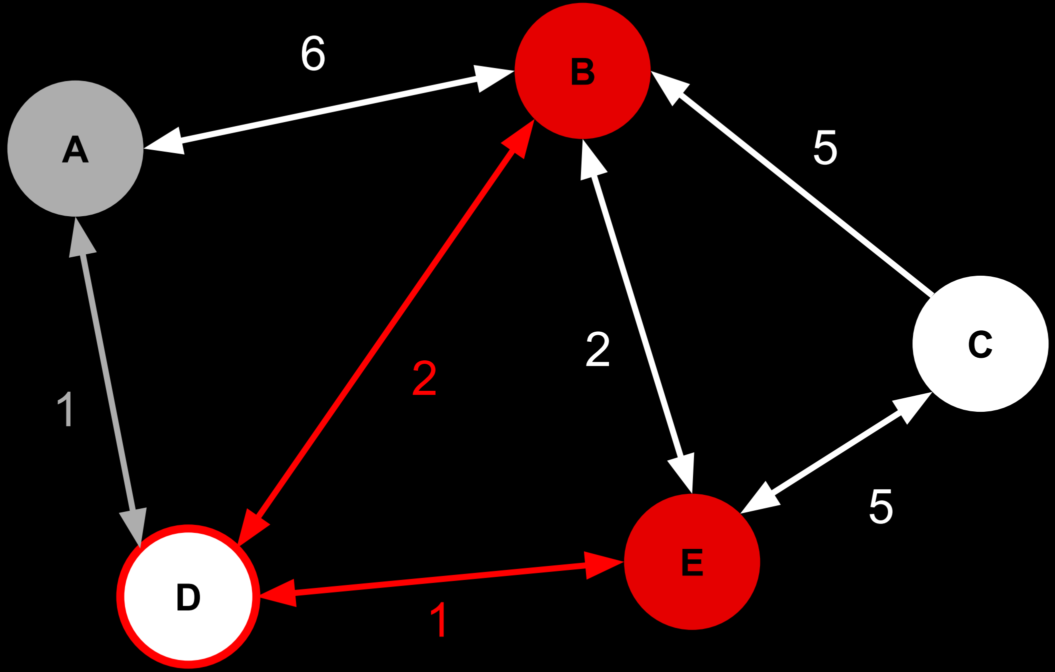
Mark A as visited

Distances: A = 0, B = 3, C = ∞, D = 1, E = ∞



Update distance to B to 3

Distances: A = 0, B = 3, C = ∞, D = 1, E = 2

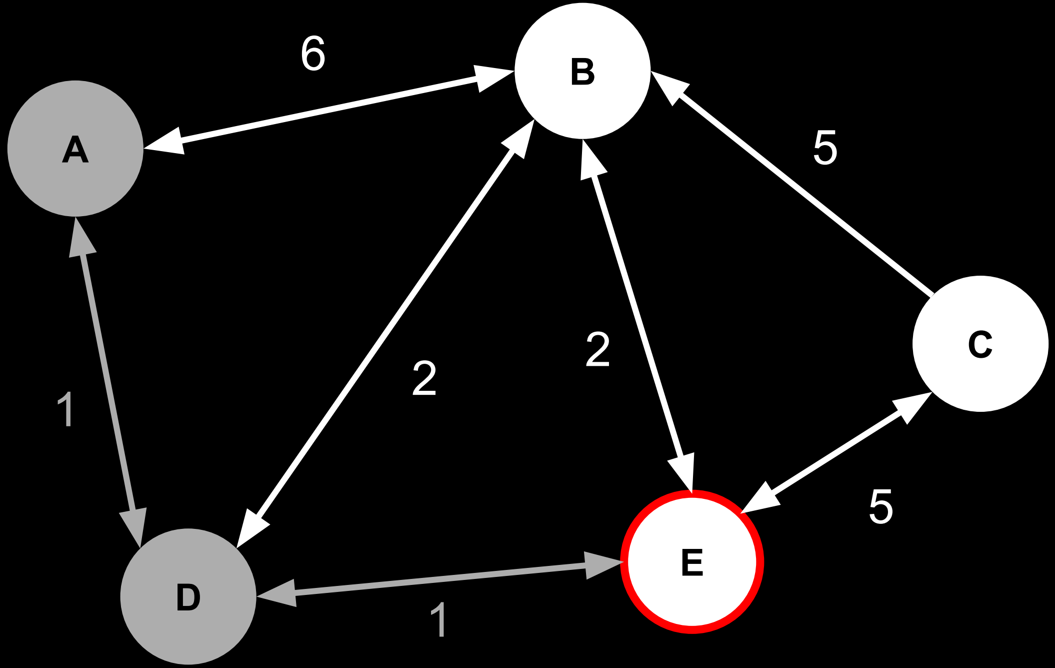


Update distance to E to 2

Distances: A = 0, B = 3, C = ∞, D = 1, E = 2

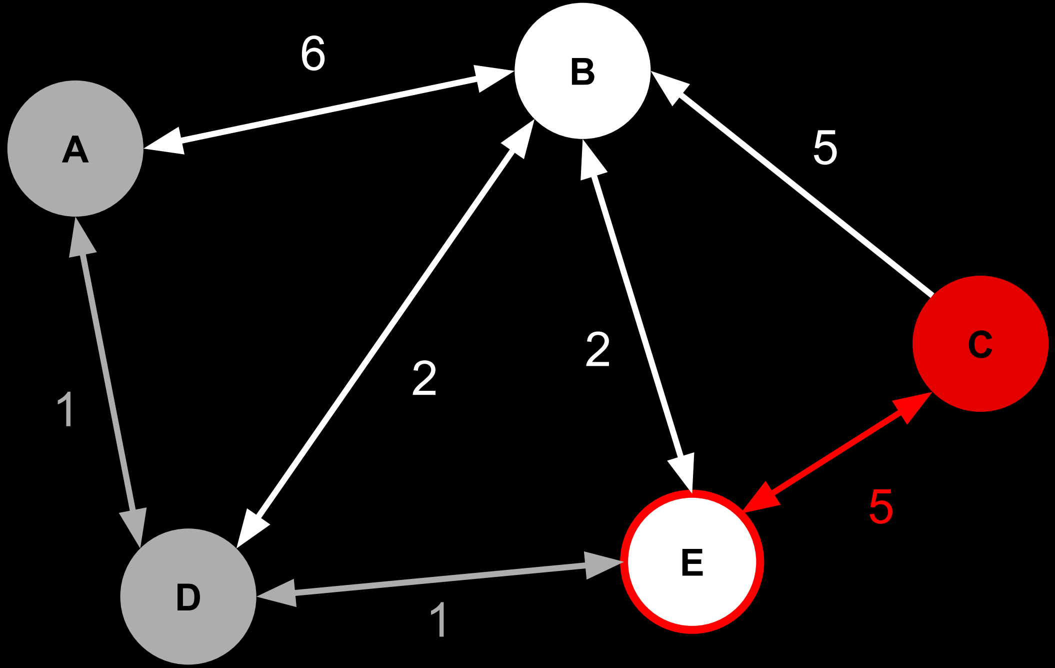
Move to E as next location

Mark D as visited



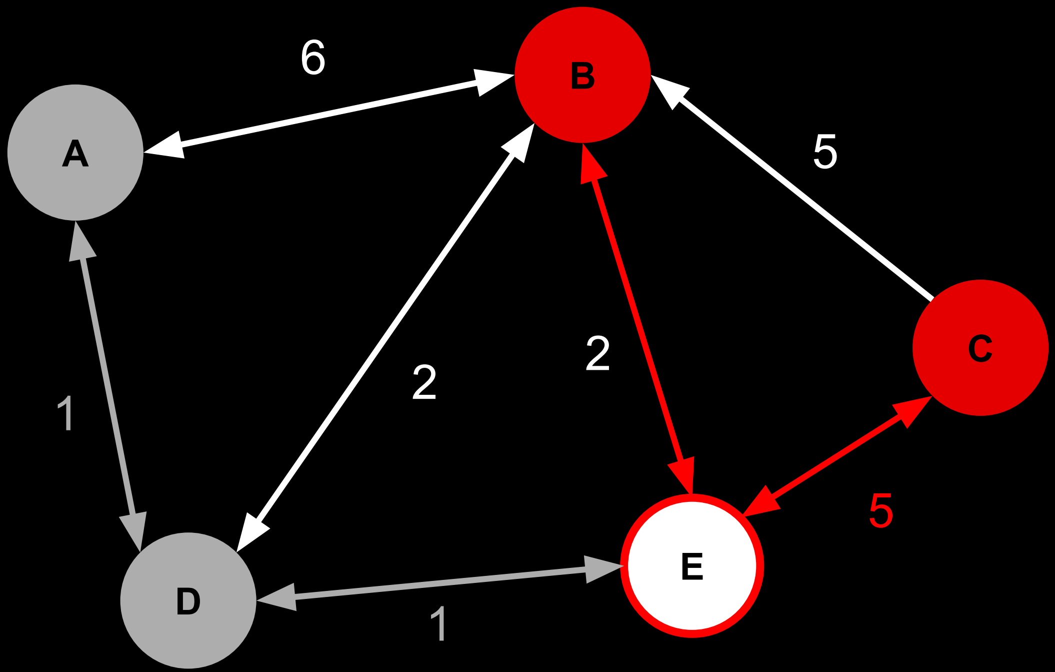
Distances: A = 0, B = 3, C = 7, D = 1, E = 2

Update distance to C to 7

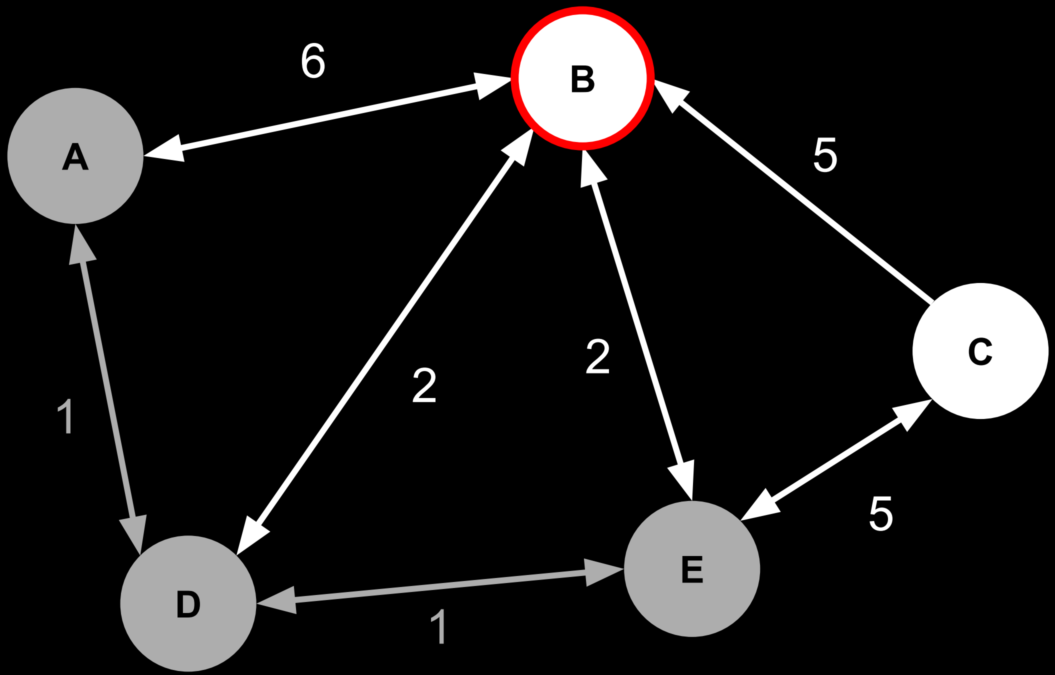


Distances: A = 0, B = 3, C = 7, D = 1, E = 2

No update for B, shorter path exists



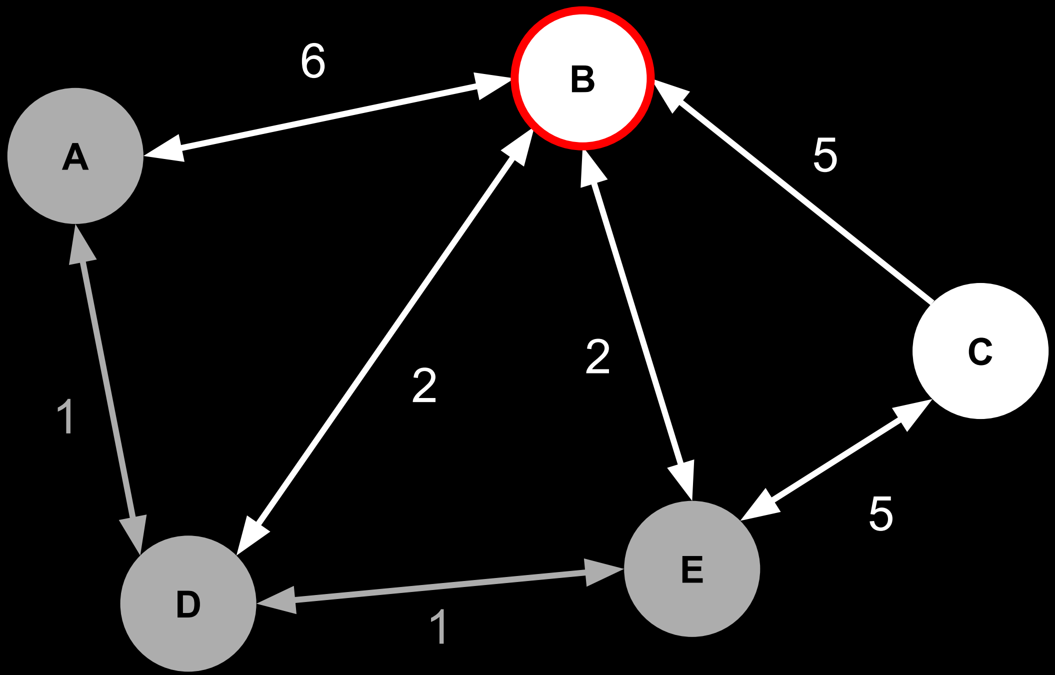
Distances: A = 0, B = 3, C = 7, D = 1, E = 2



Move to B as new location

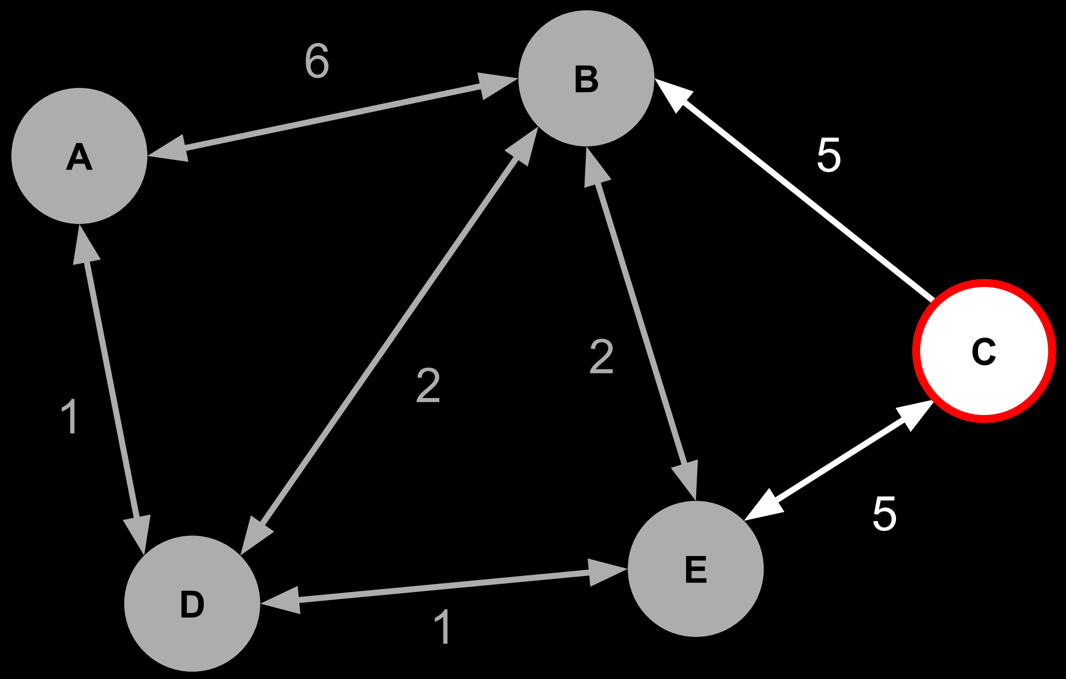
Mark E as visited

Distances: A = 0, B = 3, C = 7, D = 1, E = 2



No further locations reachable from B

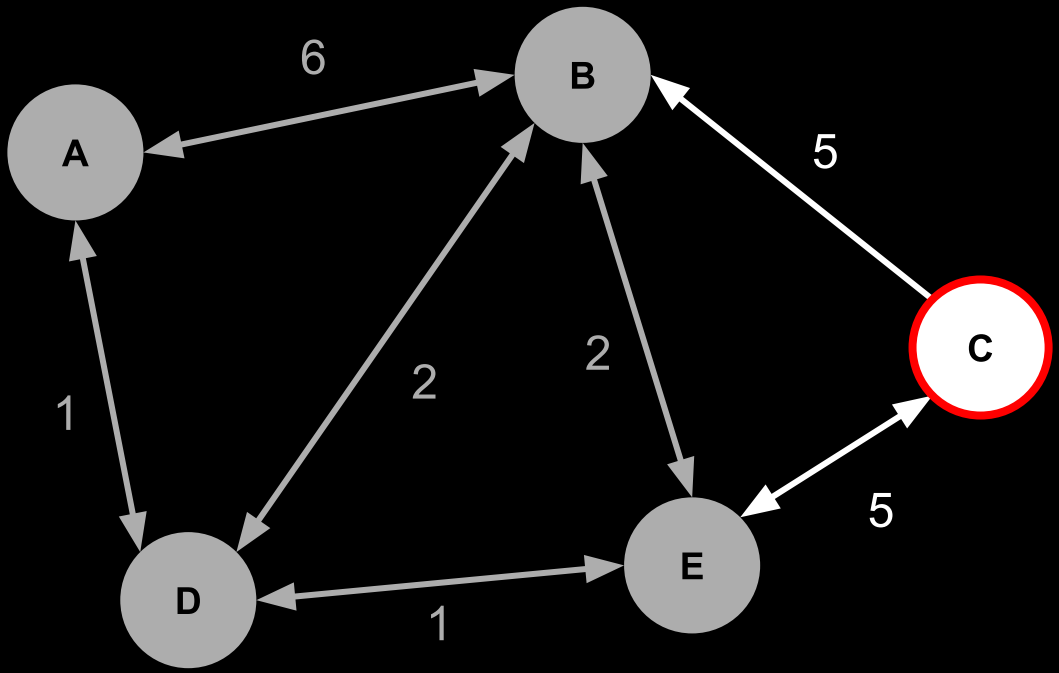
Distances: A = 0, B = 3, C = 7, D = 1, E = 2



Move to C as new location

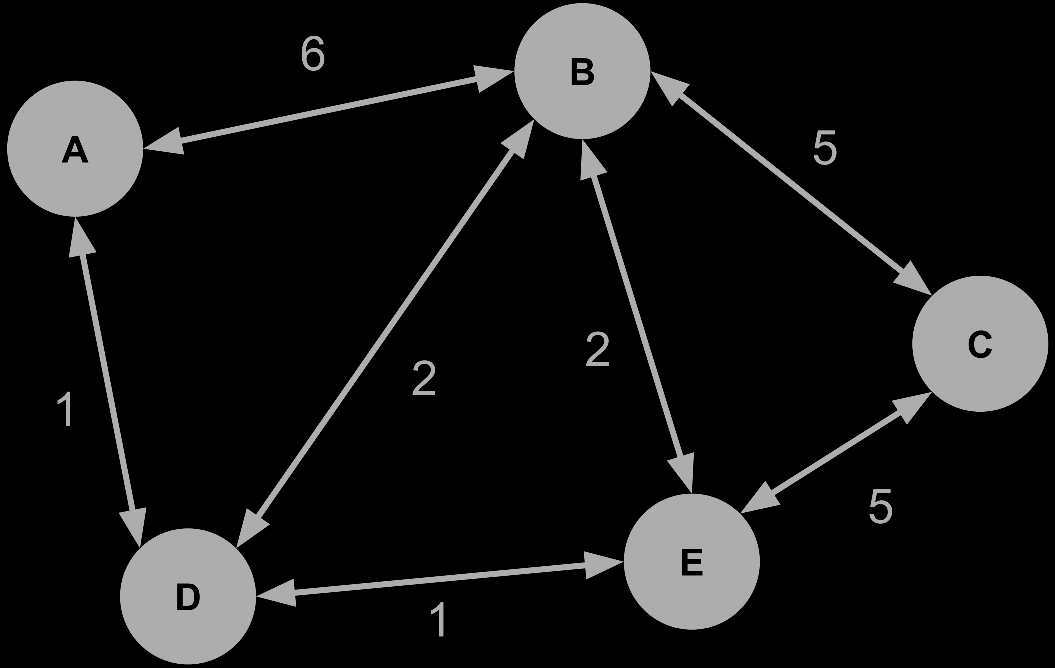
Distances: A = 0, B = 3, C = 7, D = 1, E = 2

No further locations reachable from C



Distances: A = 0, B = 3, C = 7, D = 1, E = 2

All nodes visited, we're done!



spam emails

finding oranges in images

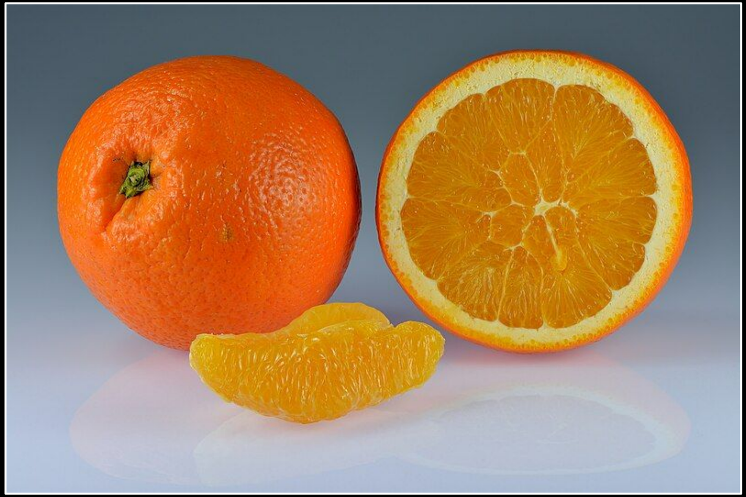


Image source: Wikimedia

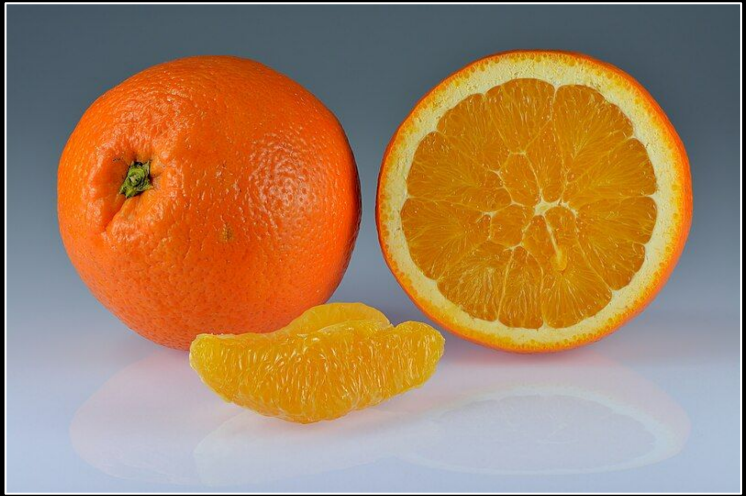


Image source: Wikimedia

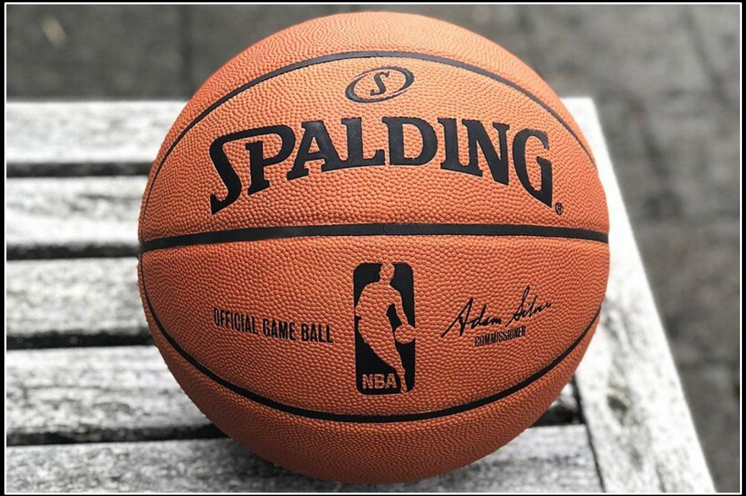
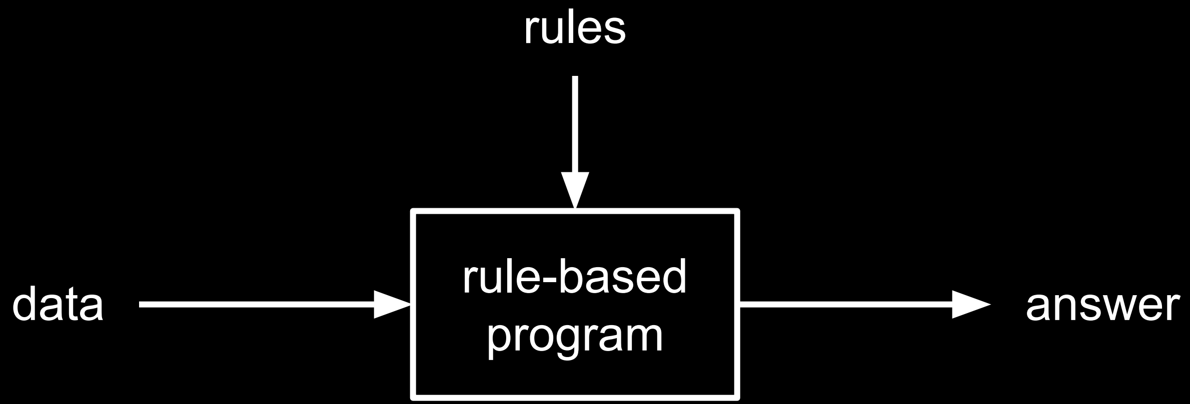
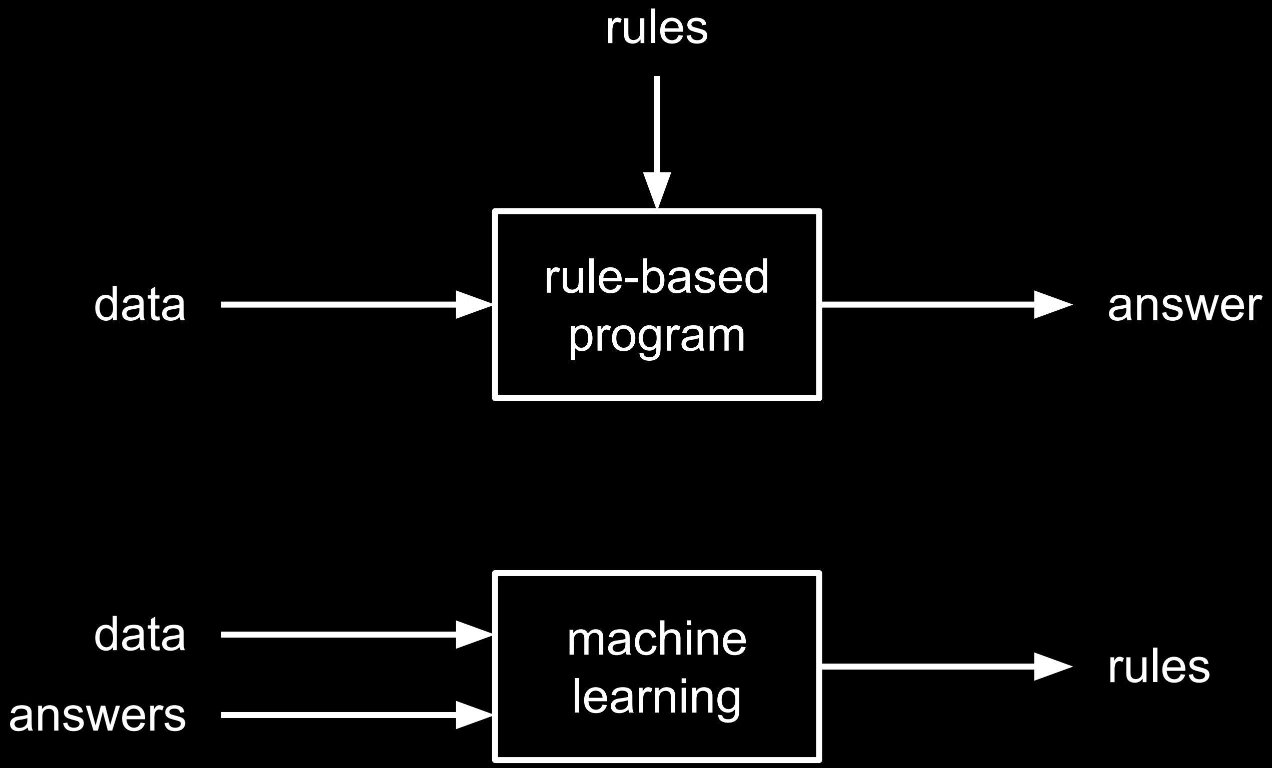


Image source: Wikimedia

what set of rules can solve this?

machine learning algorithms


rules
data
rule-based
program
answer


rules
data
rule-based
program
answer
data
answers
machine
learning
rules