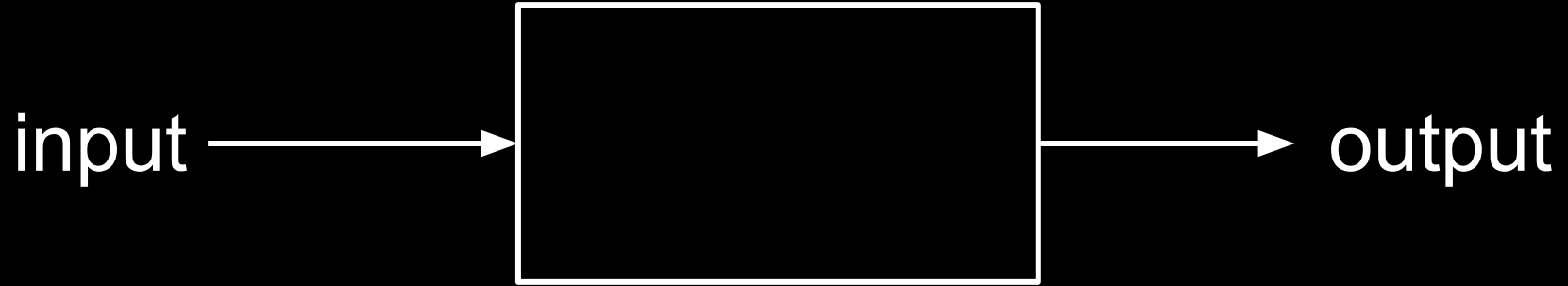


ALGORITHMS

[BACK](#)

what solves the problem?





algorithm, program, process

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

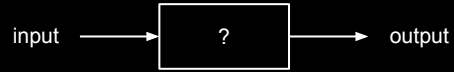
Source: Vornberger, O., Algorithmen und Datenstrukturen, Lecture notes: <http://www-lehre.inf.uos.de/~ainf/2013/PDF/skript.pdf>

"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. 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:

$a = 18, b = 48 \rightarrow \text{swap} \rightarrow a = 48, b = 18$

$a = 48 - 18 = 30$

Loop 1:

$a = 18, b = 48 \rightarrow \text{swap} \rightarrow a = 48, b = 18$

$a = 48 - 18 = 30$

Loop 2:

$a = 30, b = 18 \rightarrow \text{no swap}$

$a = 30 - 18 = 12$

Loop 1:

$a = 18, b = 48 \rightarrow \text{swap} \rightarrow a = 48, b = 18$

$a = 48 - 18 = 30$

Loop 2:

$a = 30, b = 18 \rightarrow \text{no swap}$

$a = 30 - 18 = 12$

Loop 3:

$a = 12, b = 18 \rightarrow \text{swap} \rightarrow a = 18, b = 12$

$a = 18 - 12 = 6$

Loop 1:

$a = 18, b = 48 \rightarrow \text{swap} \rightarrow a = 48, b = 18$

$a = 48 - 18 = 30$

Loop 2:

$a = 30, b = 18 \rightarrow \text{no swap}$

$a = 30 - 18 = 12$

Loop 3:

$a = 12, b = 18 \rightarrow \text{swap} \rightarrow a = 18, b = 12$

$a = 18 - 12 = 6$

Loop 4:

$a = 6, b = 12 \rightarrow \text{swap} \rightarrow a = 12, b = 6$

$a = 12 - 6 = 6$

Loop 1:

$a = 18, b = 48 \rightarrow \text{swap} \rightarrow a = 48, b = 18$

$a = 48 - 18 = 30$

Loop 2:

$a = 30, b = 18 \rightarrow \text{no swap}$

$a = 30 - 18 = 12$

Loop 3:

$a = 12, b = 18 \rightarrow \text{swap} \rightarrow a = 18, b = 12$

$a = 18 - 12 = 6$

Loop 4:

$a = 6, b = 12 \rightarrow \text{swap} \rightarrow a = 12, b = 6$

$a = 12 - 6 = 6$

Loop 5:

$a = 6, b = 6 \rightarrow \text{no swap}$

$a = 6 - 6 = 0$

Loop 1:

$a = 18, b = 48 \rightarrow \text{swap} \rightarrow a = 48, b = 18$

$a = 48 - 18 = 30$

Loop 2:

$a = 30, b = 18 \rightarrow \text{no swap}$

$a = 30 - 18 = 12$

Loop 3:

$a = 12, b = 18 \rightarrow \text{swap} \rightarrow a = 18, b = 12$

$a = 18 - 12 = 6$

Loop 4:

$a = 6, b = 12 \rightarrow \text{swap} \rightarrow a = 12, b = 6$

$a = 12 - 6 = 6$

Loop 5:

$a = 6, b = 6 \rightarrow \text{no swap}$

$a = 6 - 6 = 0$

return $b = 6$

algorithm representation

100 Good Cookies

1 cup white sugar

1 cup brown sugar

1 cup margarine

1 cup cooking oil

1 egg

1 teaspoon vanilla

1 teaspoon cream of tartar

1 teaspoon baking soda

1 cup quick oatmeal

1 cup coconut

1 cup Rice Krispies

1 cup chopped walnuts

3½ cups flour

1-12oz. package mini

chocolate chips

Mix in order given. Drop by teaspoonful onto
cookie sheet. Bake at 350° for 8-10 minutes.
(over)

pseudocode

READ a, b

REPEAT

IF $a < b$ THEN

$a = b$

$b = a$

$a = a - b$

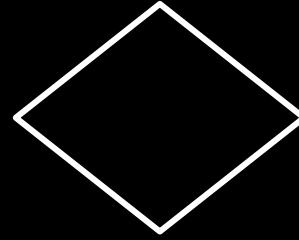
UNTIL $a = 0$ OR $b = 0$

RETURN the variable that is not 0

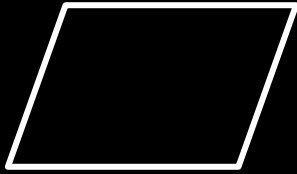
flow diagrams



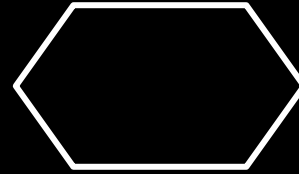
start / end of
algorithm



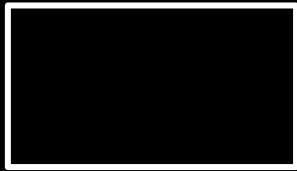
decision



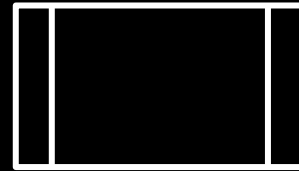
input / output



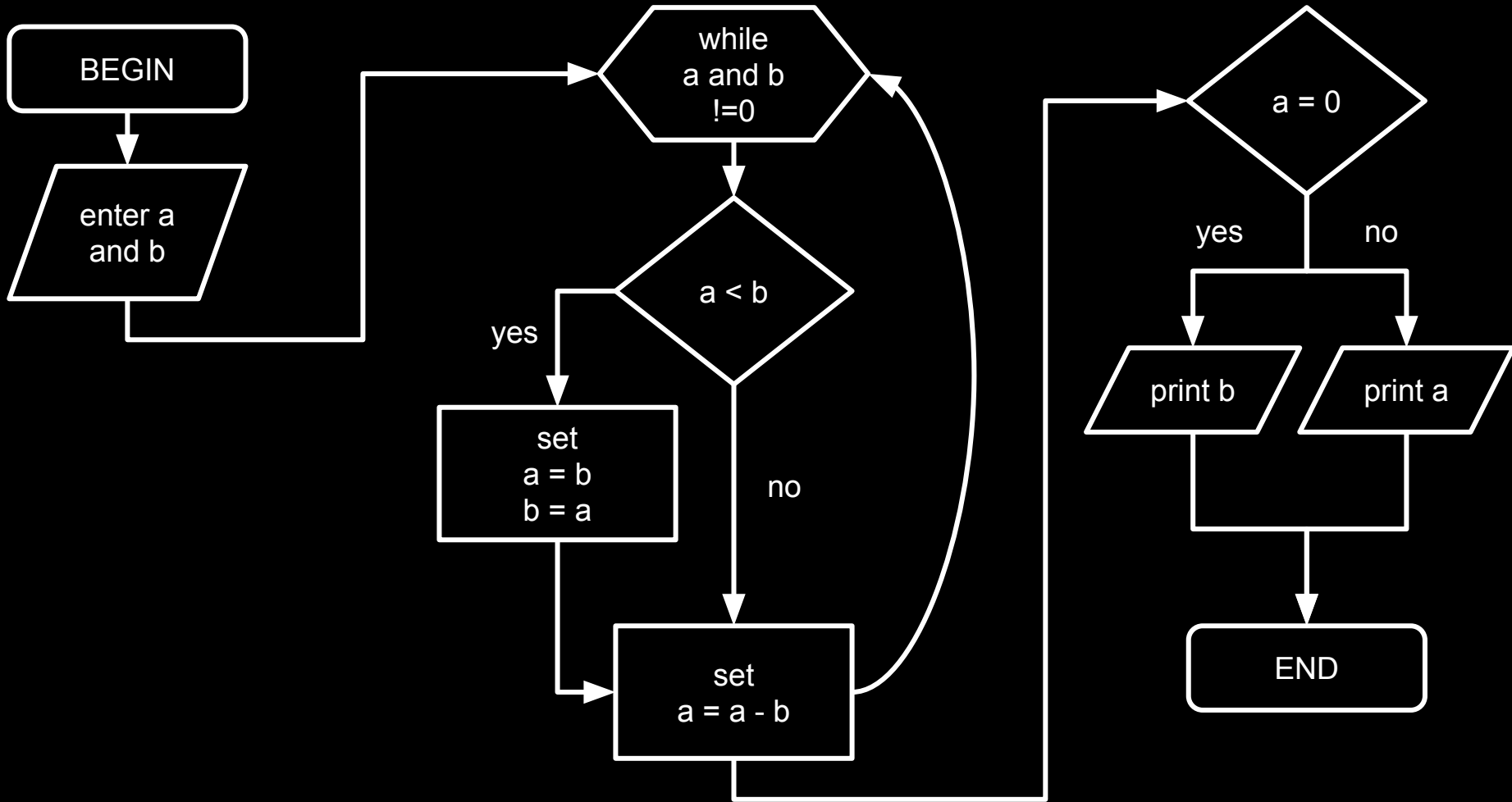
repetition



command /
assignment



external routine





square roots

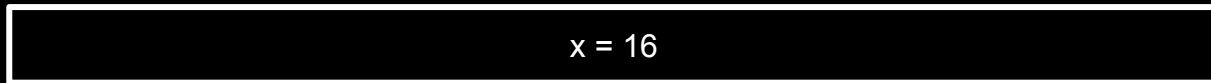
babylonian method

calculate square root of
 $x = 16$

$$A = 1$$

$$B = x / A = 16$$

A

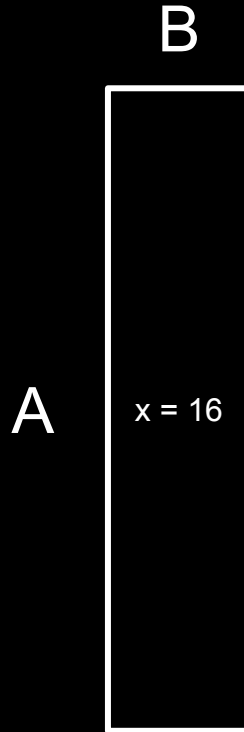


B

$x = 16$

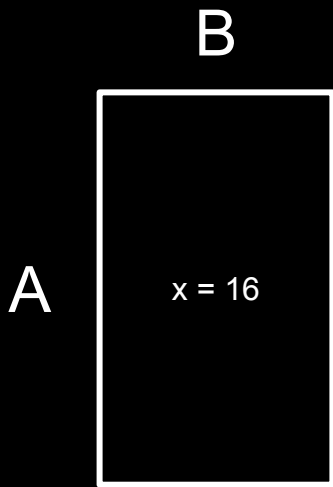
$$A = (A + B) / 2 = 17 / 2 = 8.5$$

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



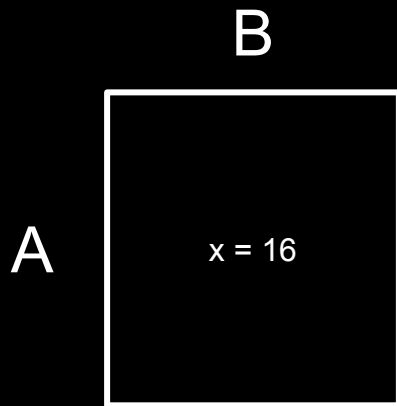
$$A = (A + B) / 2 \approx 10.38 / 2 \approx 5.19$$

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



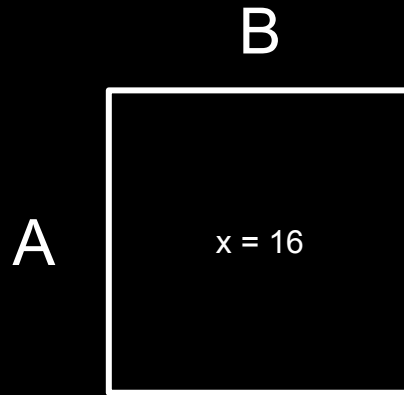
$$A = (A + B) / 2 \approx 8.27 / 2 \approx 4.14$$

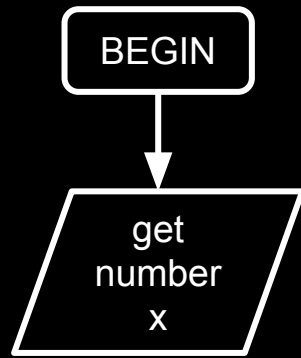
$$B = x / A \approx 16 / 4.14 \approx 3.86$$

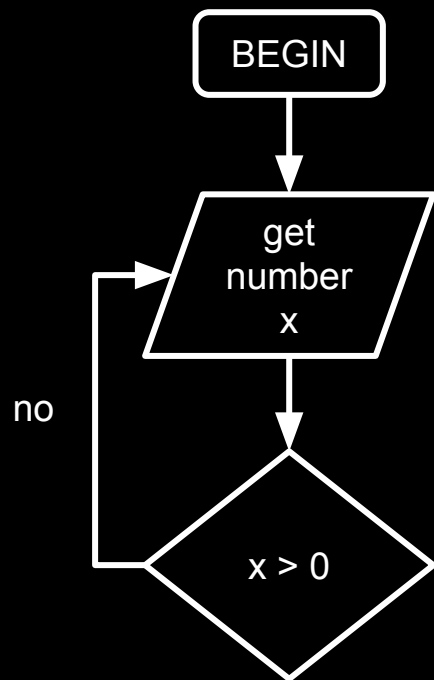


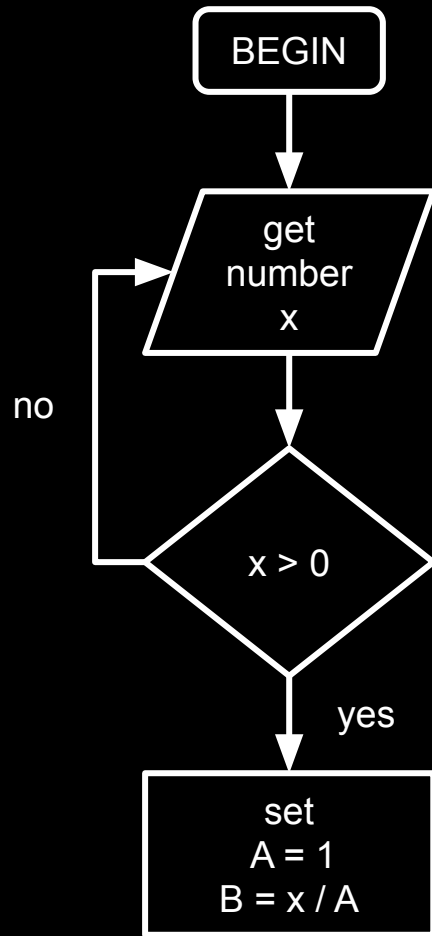
$$A = (A + B) / 2 = 8 / 2 = 4$$

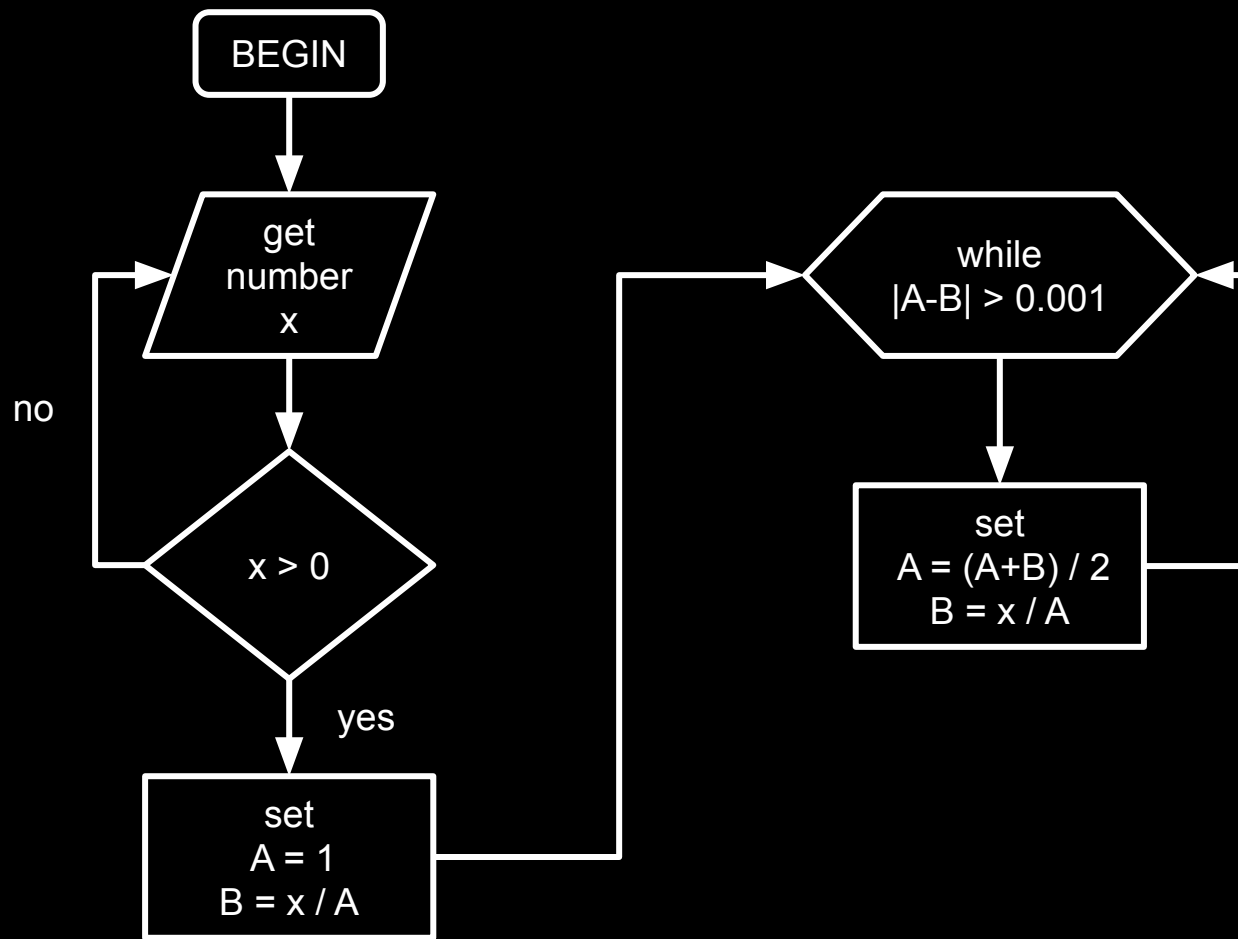
$$B = x / A = 16 / 4 = 4$$

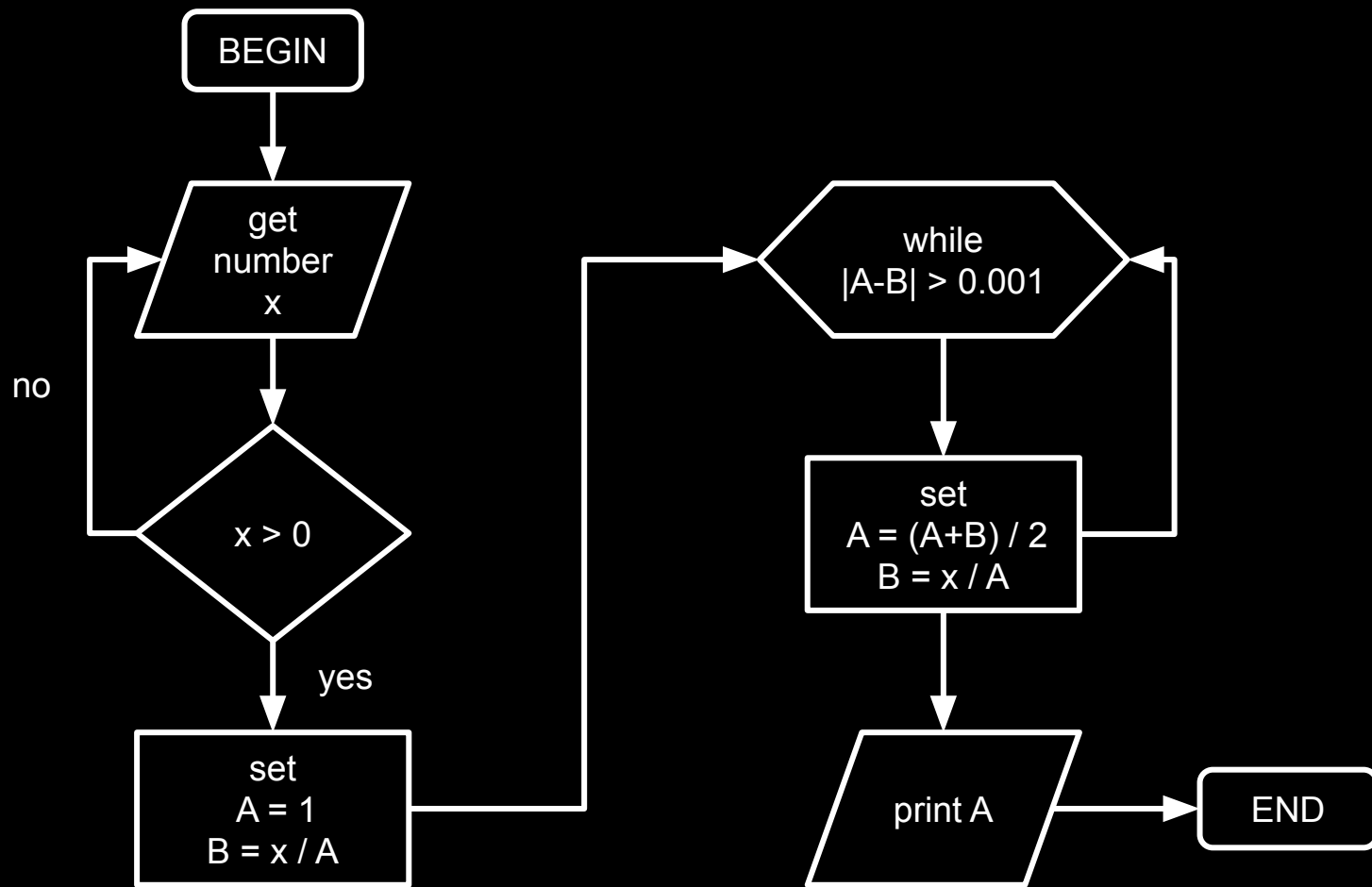


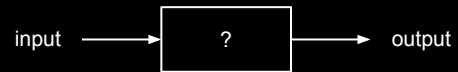




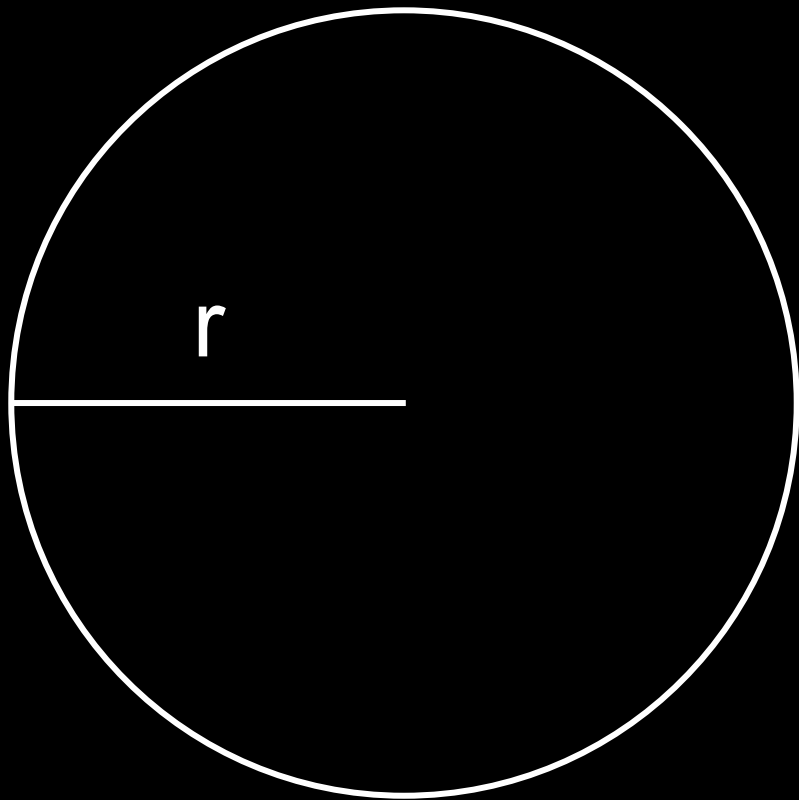


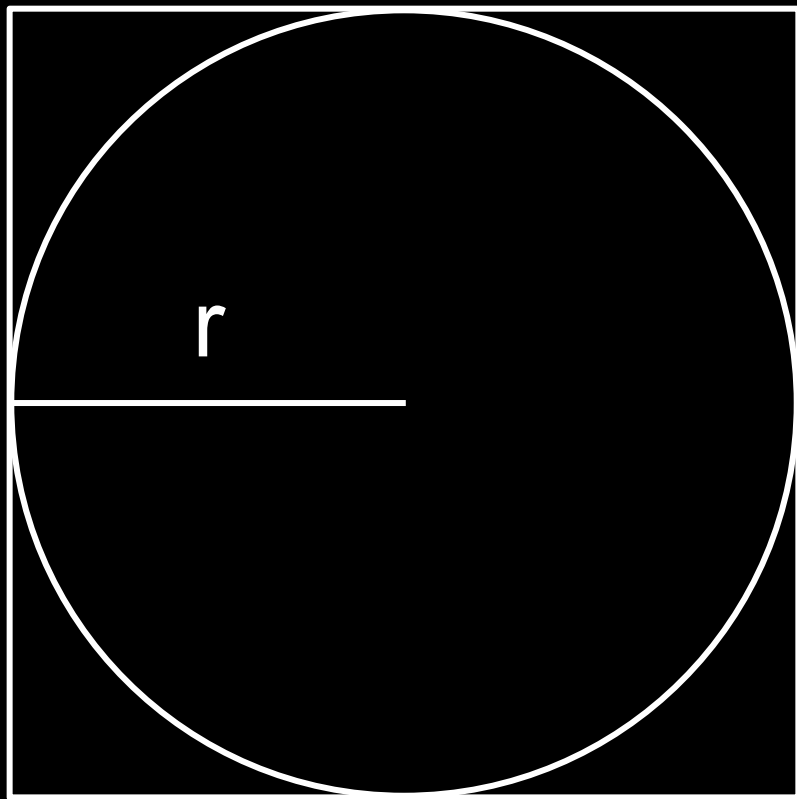


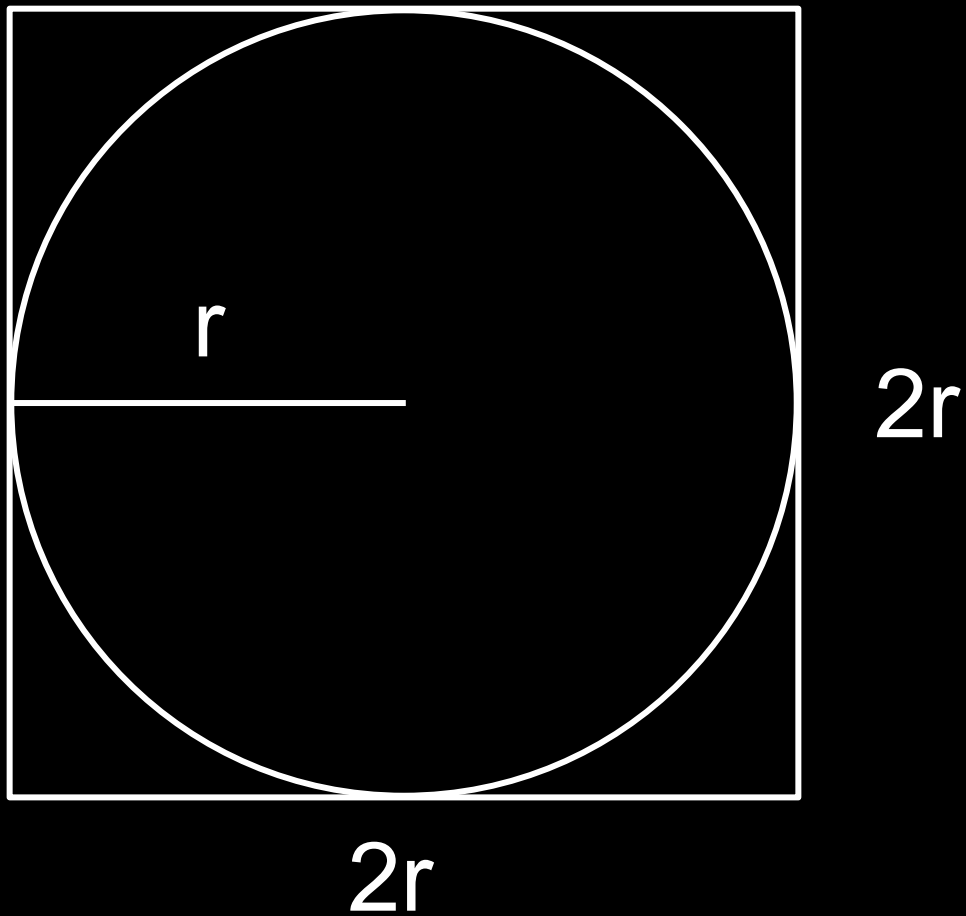


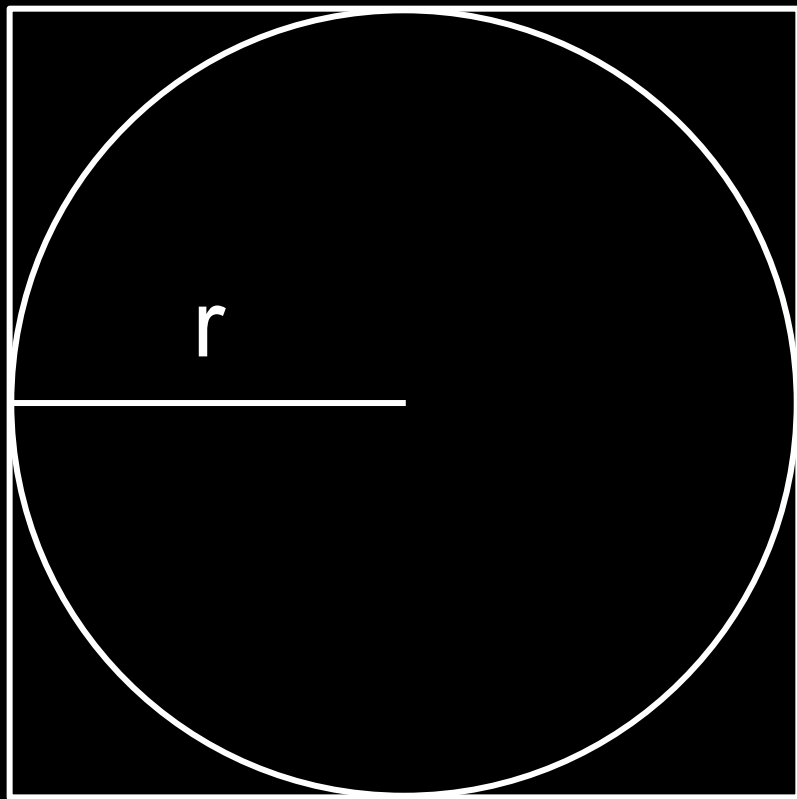


estimating π



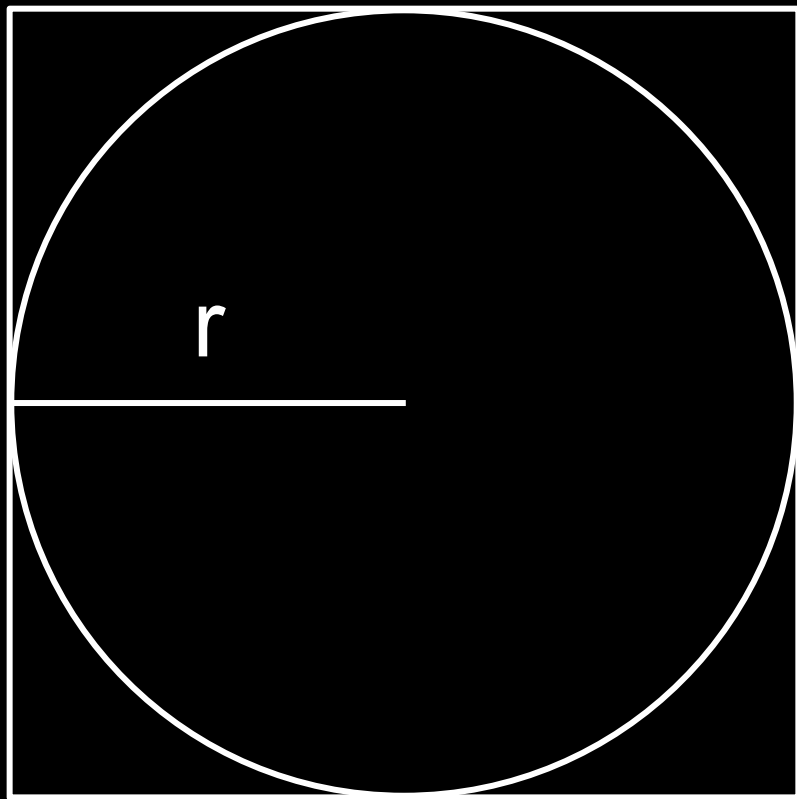






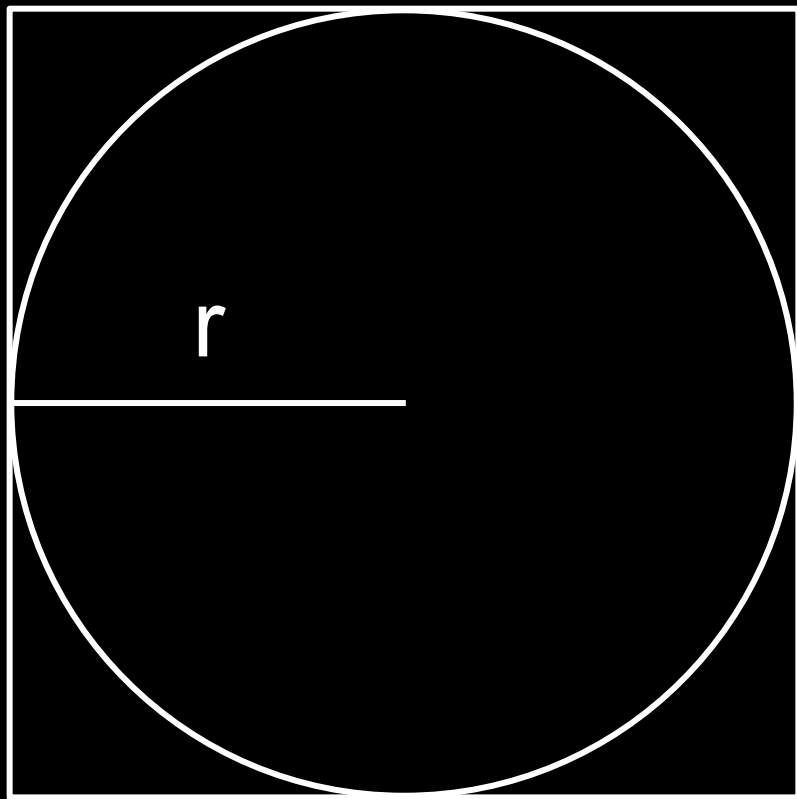
$2r$

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



$2r$

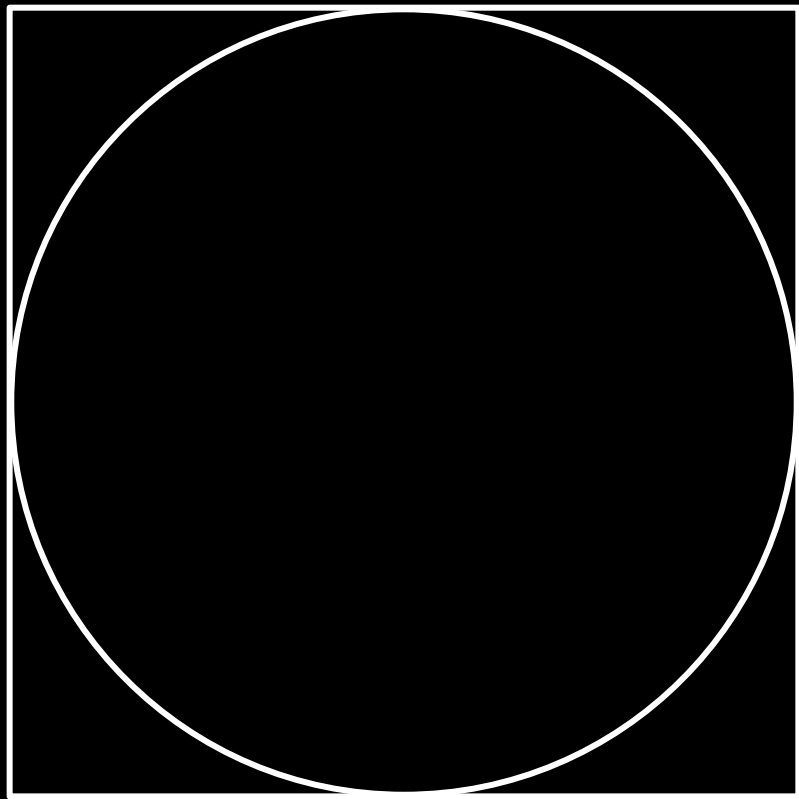
$$\frac{\text{○}}{\text{□}} = \frac{\pi r^2}{4r^2}$$



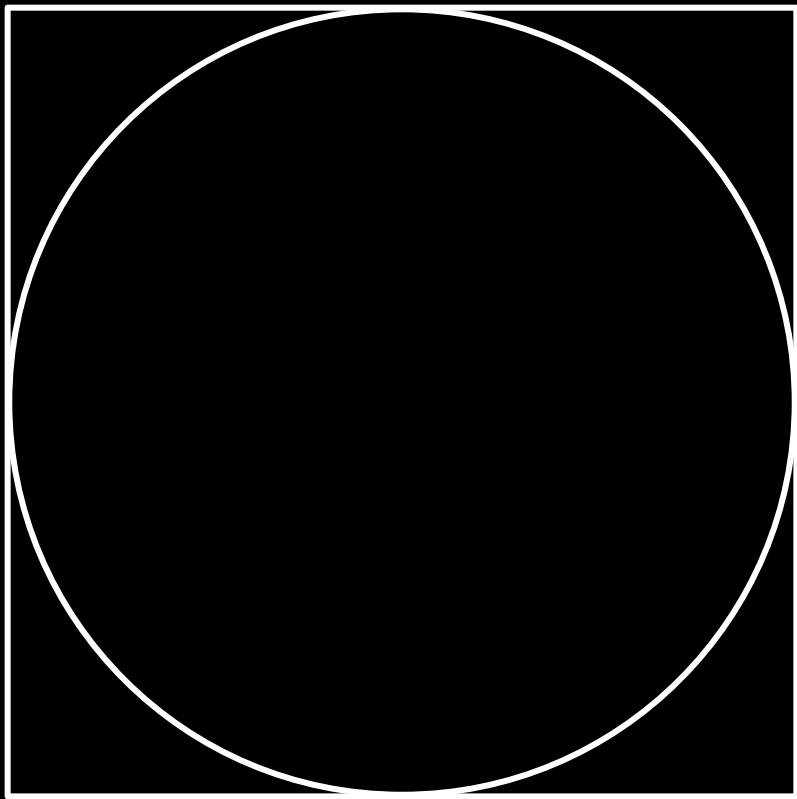
$2r$

$$\frac{\text{○}}{\text{□}} = \frac{\pi}{4}$$

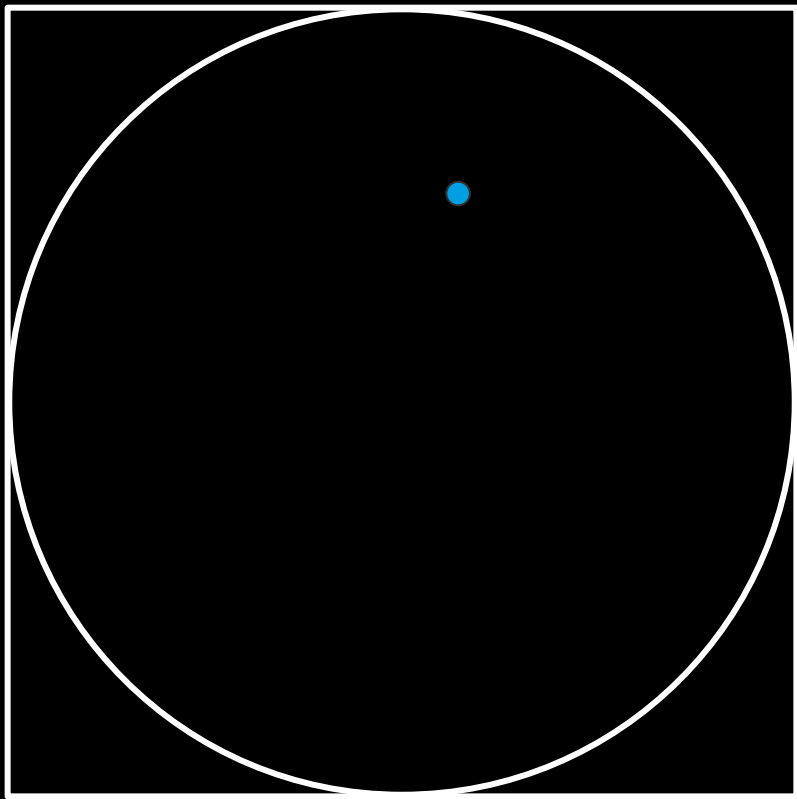
monte carlo simulation



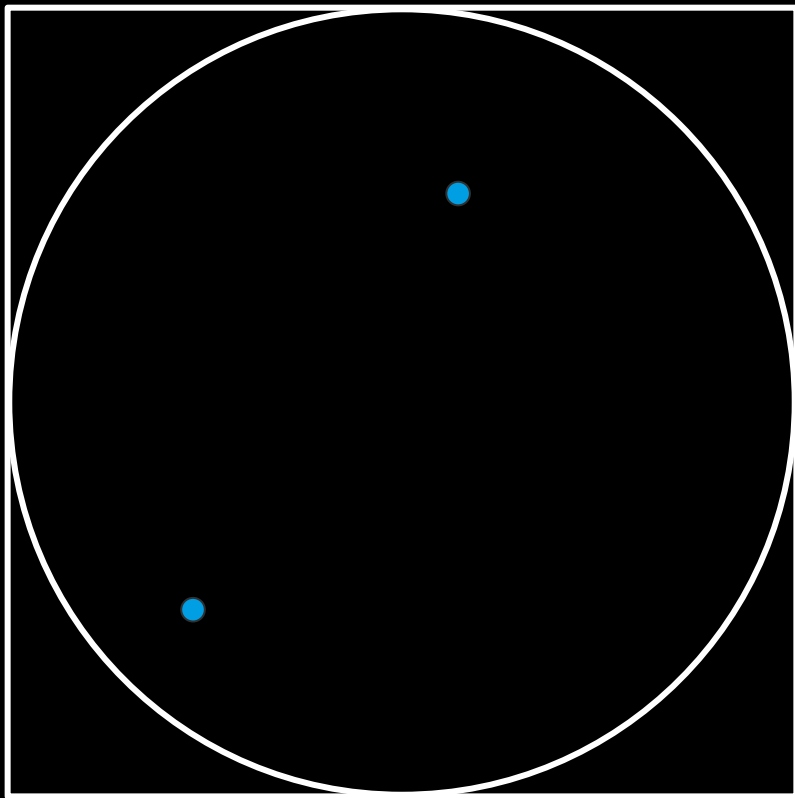
$$\frac{\text{Circle}}{\text{Square}} = \frac{\pi}{4}$$



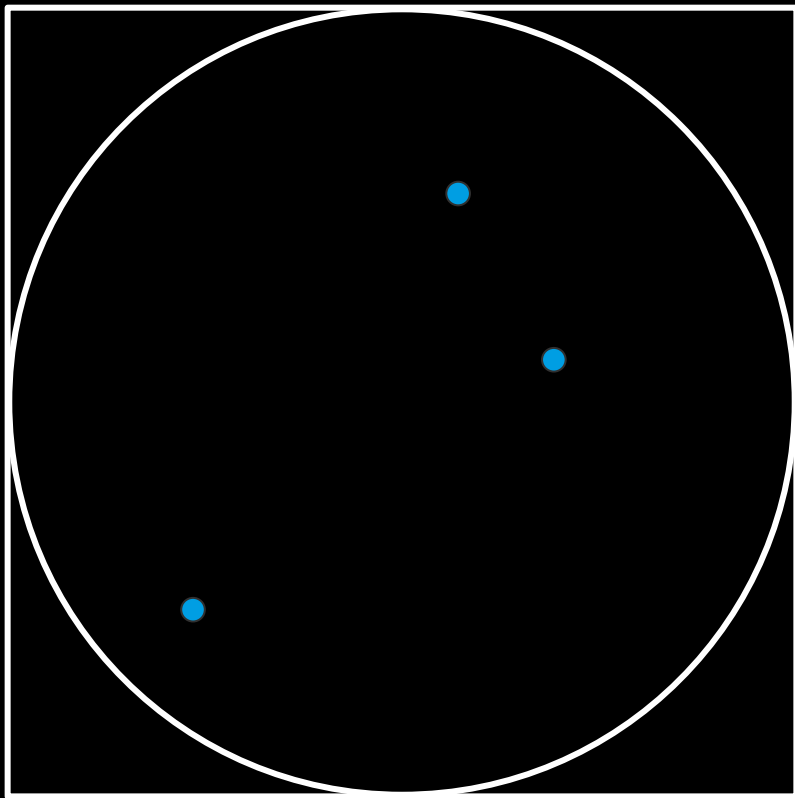
$$4 \frac{\text{circle}}{\text{square}} = \pi$$



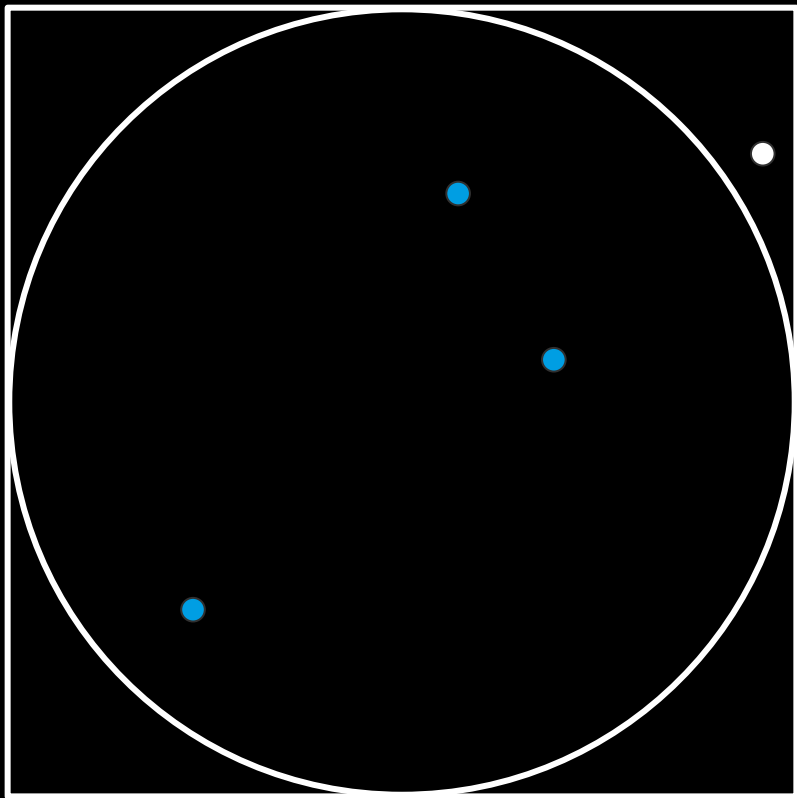
$$4 \frac{\text{○}}{\text{□}} = \pi$$
$$= 4$$



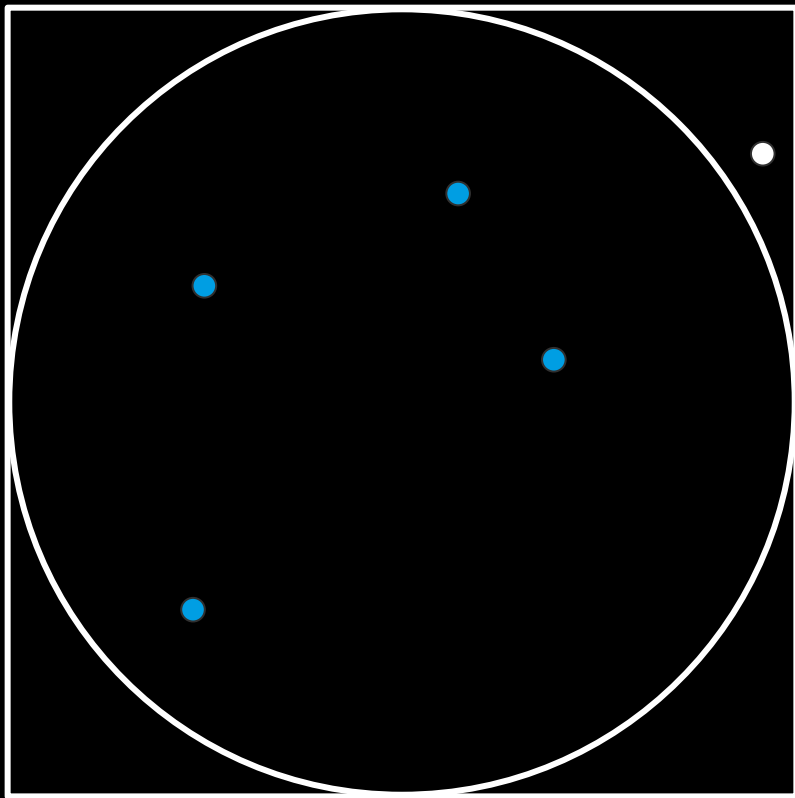
$$4 \frac{\text{○}}{\text{□}} = \pi$$
$$= 4$$



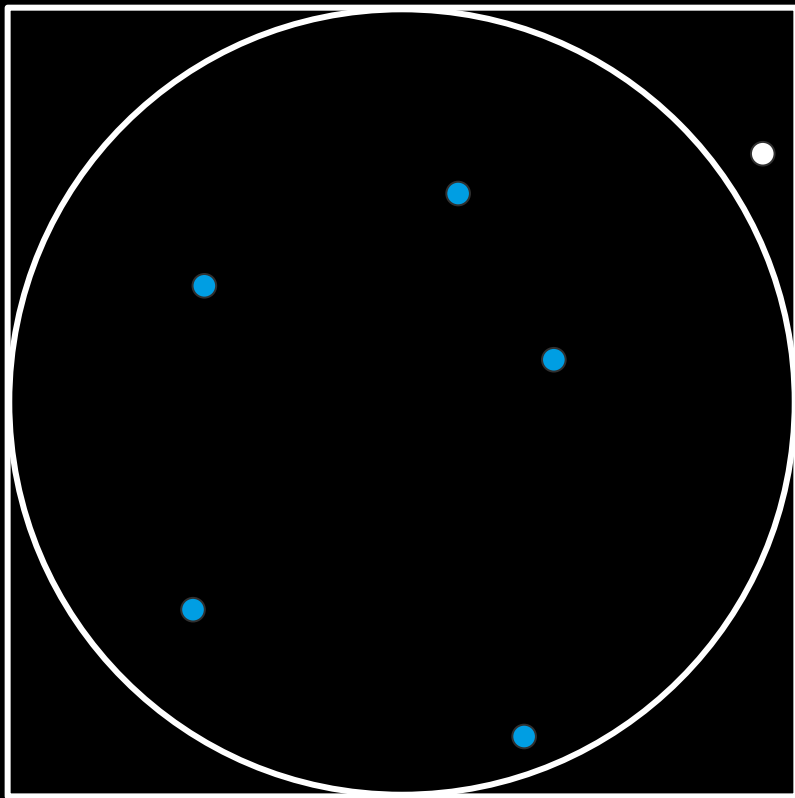
$$4 \frac{\text{circle}}{\text{square}} = \pi$$
$$= 4$$



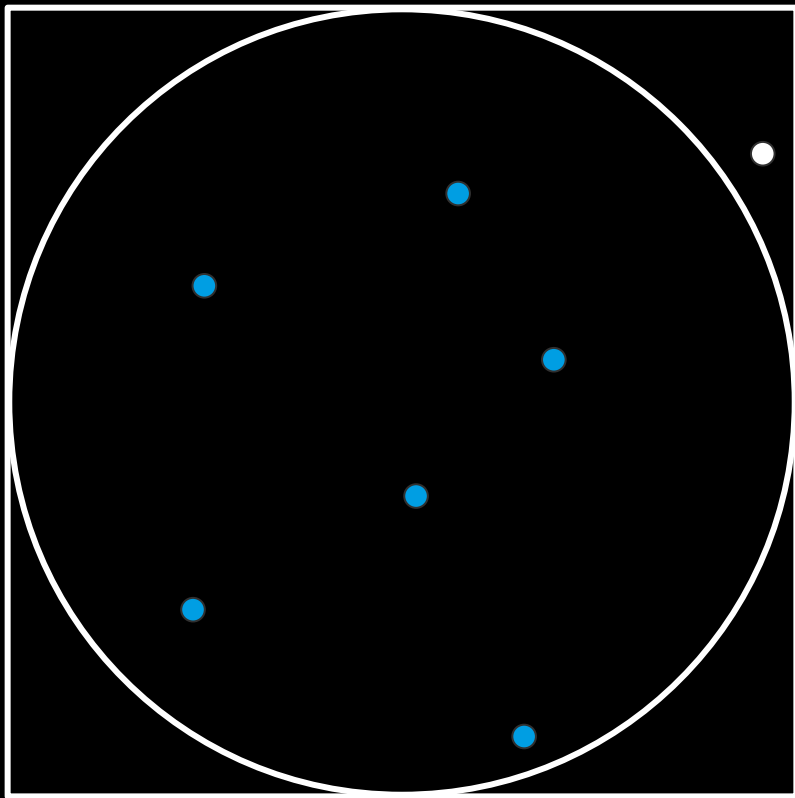
$$4 \frac{\text{Circle}}{\text{Square}} = \pi$$
$$= 3$$



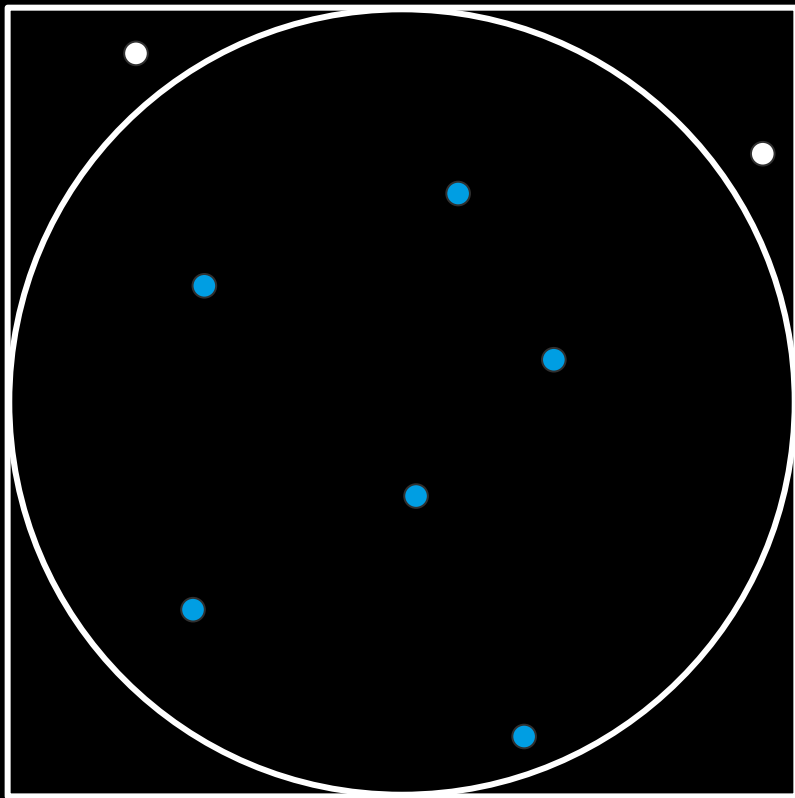
$$4 \frac{\text{circle}}{\text{square}} = \pi$$
$$= 3,2$$



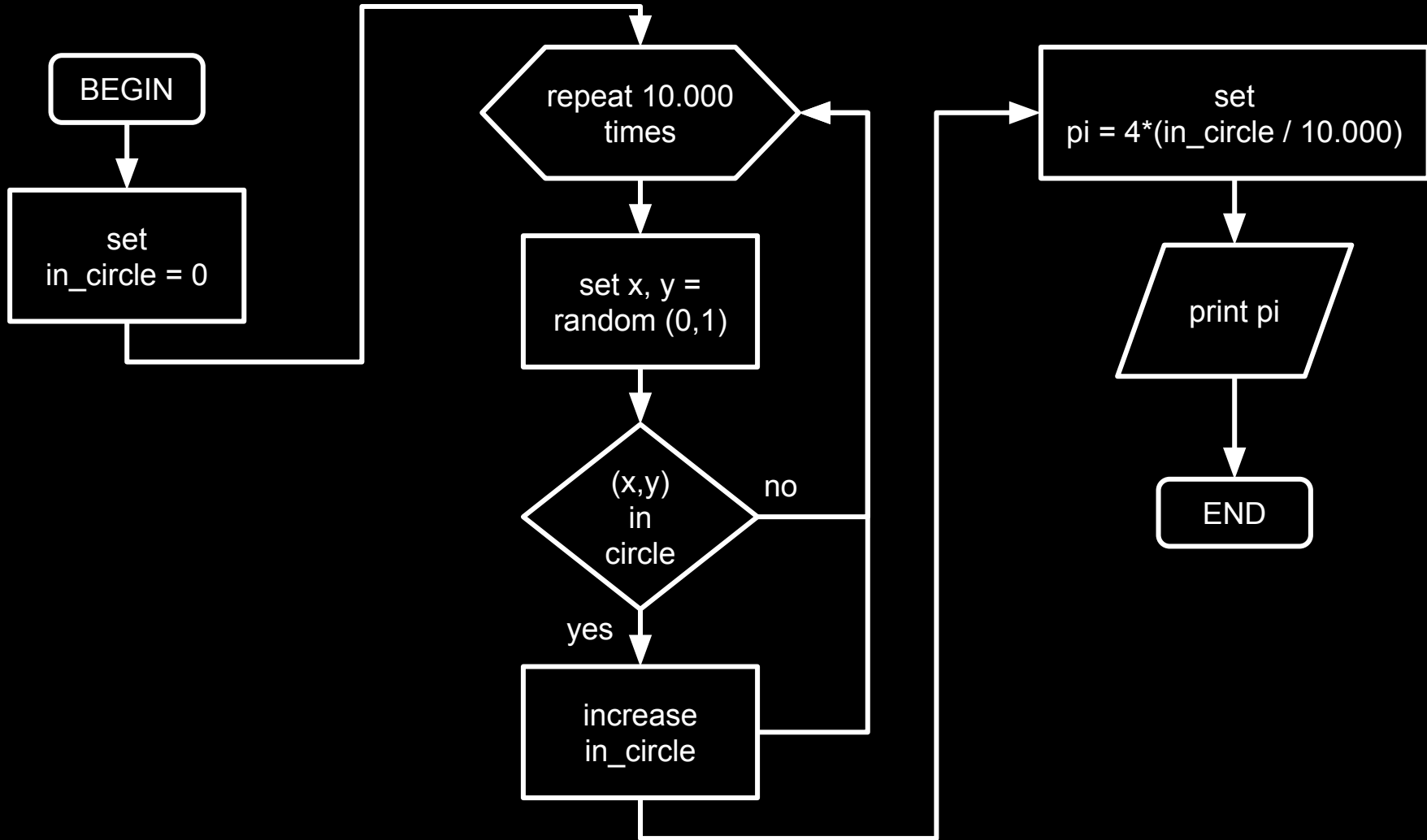
$$4 \frac{\text{○}}{\text{□}} = \pi$$
$$= 3,33$$



$$4 \frac{\text{○}}{\text{□}} = \pi$$
$$= 3,43$$



$$4 \frac{\text{Circle}}{\text{Square}} = \pi$$
$$= 3$$



gregory-leibniz series

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

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



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] \longrightarrow 9 > 5 ?

[9, 5, 2, 1, 4, 7] \longrightarrow 9 > 5 ? $\xrightarrow{\text{yes}}$ [5, 9, 2, 1, 4, 7]

[9, 5, 2, 1, 4, 7] \longrightarrow 9 > 5 ? $\xrightarrow{\text{yes}}$ [5, 9, 2, 1, 4, 7]

[5, 9, 2, 1, 4, 7] \longrightarrow 9 > 2 ? $\xrightarrow{\text{yes}}$ [5, 2, 9, 1, 4, 7]

[9, 5, 2, 1, 4, 7] \longrightarrow 9 > 5 ? $\xrightarrow{\text{yes}}$ [5, 9, 2, 1, 4, 7]

[5, 9, 2, 1, 4, 7] \longrightarrow 9 > 2 ? $\xrightarrow{\text{yes}}$ [5, 2, 9, 1, 4, 7]

[5, 2, 9, 1, 4, 7] \longrightarrow 9 > 1 ? $\xrightarrow{\text{yes}}$ [5, 2, 1, 9, 4, 7]

[9, 5, 2, 1, 4, 7] \longrightarrow 9 > 5 ? $\xrightarrow{\text{yes}}$ [5, 9, 2, 1, 4, 7]

[5, 9, 2, 1, 4, 7] \longrightarrow 9 > 2 ? $\xrightarrow{\text{yes}}$ [5, 2, 9, 1, 4, 7]

[5, 2, 9, 1, 4, 7] \longrightarrow 9 > 1 ? $\xrightarrow{\text{yes}}$ [5, 2, 1, 9, 4, 7]

[5, 2, 1, 9, 4, 7] \longrightarrow 9 > 4 ? $\xrightarrow{\text{yes}}$ [5, 2, 1, 4, 9, 7]

[9, 5, 2, 1, 4, 7] \longrightarrow 9 > 5 ? $\xrightarrow{\text{yes}}$ [5, 9, 2, 1, 4, 7]

[5, 9, 2, 1, 4, 7] \longrightarrow 9 > 2 ? $\xrightarrow{\text{yes}}$ [5, 2, 9, 1, 4, 7]

[5, 2, 9, 1, 4, 7] \longrightarrow 9 > 1 ? $\xrightarrow{\text{yes}}$ [5, 2, 1, 9, 4, 7]

[5, 2, 1, 9, 4, 7] \longrightarrow 9 > 4 ? $\xrightarrow{\text{yes}}$ [5, 2, 1, 4, 9, 7]

[5, 2, 1, 4, 9, 7] \longrightarrow 9 > 7 ? $\xrightarrow{\text{yes}}$ [5, 2, 1, 4, 7, 9]

[5, 2, 1, 4, 7, 9] \longrightarrow 5 > 2 ? $\xrightarrow{\text{yes}}$ [2, 5, 1, 4, 7, 9]

[5, 2, 1, 4, 7, 9] \longrightarrow 5 > 2 ? $\xrightarrow{\text{yes}}$ [2, 5, 1, 4, 7, 9]

[2, 5, 1, 4, 7, 9] \longrightarrow 5 > 1 ? $\xrightarrow{\text{yes}}$ [2, 1, 5, 4, 7, 9]

[5, 2, 1, 4, 7, 9] \longrightarrow 5 > 2 ? $\xrightarrow{\text{yes}}$ [2, 5, 1, 4, 7, 9]

[2, 5, 1, 4, 7, 9] \longrightarrow 5 > 1 ? $\xrightarrow{\text{yes}}$ [2, 1, 5, 4, 7, 9]

[2, 1, 5, 4, 7, 9] \longrightarrow 5 > 4 ? $\xrightarrow{\text{yes}}$ [2, 1, 4, 5, 7, 9]

[5, 2, 1, 4, 7, 9] \longrightarrow 5 > 2 ? $\xrightarrow{\text{yes}}$ [2, 5, 1, 4, 7, 9]

[2, 5, 1, 4, 7, 9] \longrightarrow 5 > 1 ? $\xrightarrow{\text{yes}}$ [2, 1, 5, 4, 7, 9]

[2, 1, 5, 4, 7, 9] \longrightarrow 5 > 4 ? $\xrightarrow{\text{yes}}$ [2, 1, 4, 5, 7, 9]

[2, 1, 4, 5, 7, 9] \longrightarrow 5 > 7 ? $\xrightarrow{\text{no}}$ [2, 1, 4, 5, 7, 9]

[2, 1, 4, 5, 7, 9] \longrightarrow 2 > 1 ? $\xrightarrow{\text{yes}}$ [1, 2, 4, 5, 7, 9]

[2, 1, 4, 5, 7, 9] \longrightarrow 2 > 1 ? $\xrightarrow{\text{yes}}$ [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] \longrightarrow 2 > 4 ? $\xrightarrow{\text{no}}$ [1, 2, 4, 5, 7, 9]

[2, 1, 4, 5, 7, 9] \longrightarrow $2 > 1 ?$ $\xrightarrow{\text{yes}}$ [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] \longrightarrow $2 > 4 ?$ $\xrightarrow{\text{no}}$ [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] \longrightarrow $4 > 5 ?$ $\xrightarrow{\text{no}}$ [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] \longrightarrow 1 > 2 ? $\xrightarrow{\text{no}}$ [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] \longrightarrow 1 > 2 ? $\xrightarrow{\text{no}}$ [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] \longrightarrow 2 > 4 ? $\xrightarrow{\text{no}}$ [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] \longrightarrow 1 > 2 ? $\xrightarrow{\text{no}}$ [1, 2, 4, 5, 7, 9]

[1, 2, 4, 5, 7, 9] \longrightarrow 1 > 2 ? $\xrightarrow{\text{no}}$ [1, 2, 4, 5, 7, 9]

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

selection sort

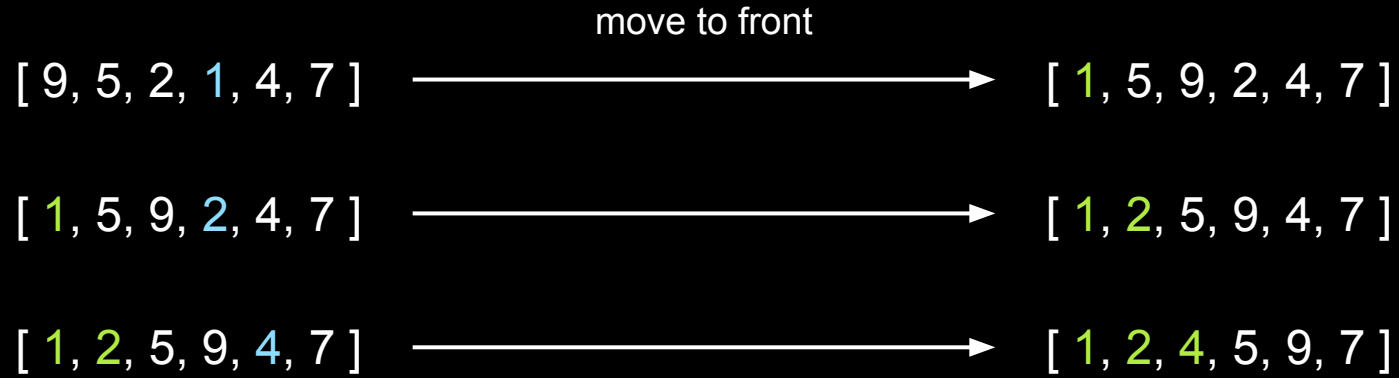
find the smallest element and move it to front. repeat for the rest of the elements.

[9, 5, 2, 1, 4, 7] move to front → [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]

[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]

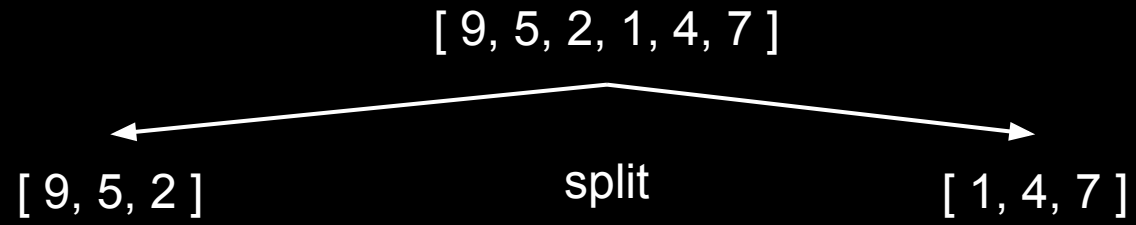
[1, 2, 4, 5, 9, 7] → [1, 2, 4, 5, 7, 9]

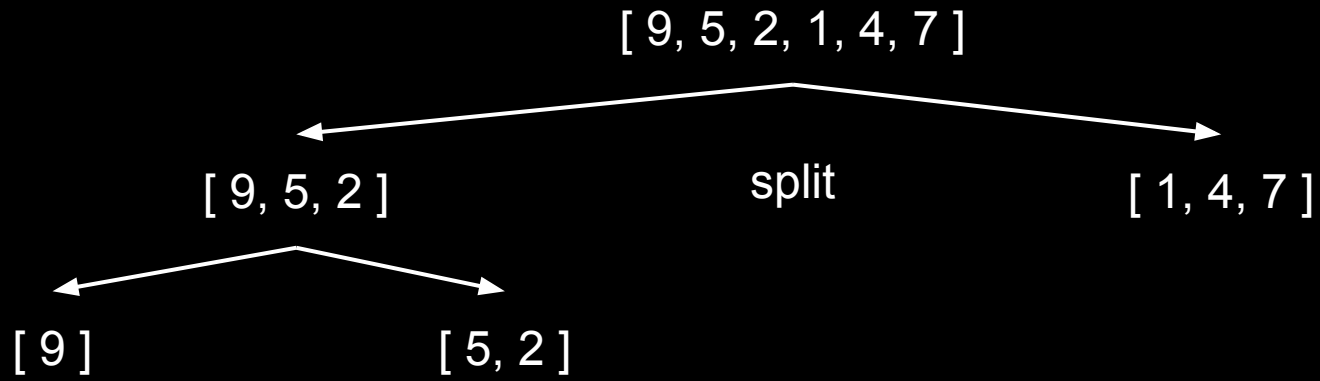
[1, 2, 4, 5, 7, 9] → [1, 2, 4, 5, 7, 9]

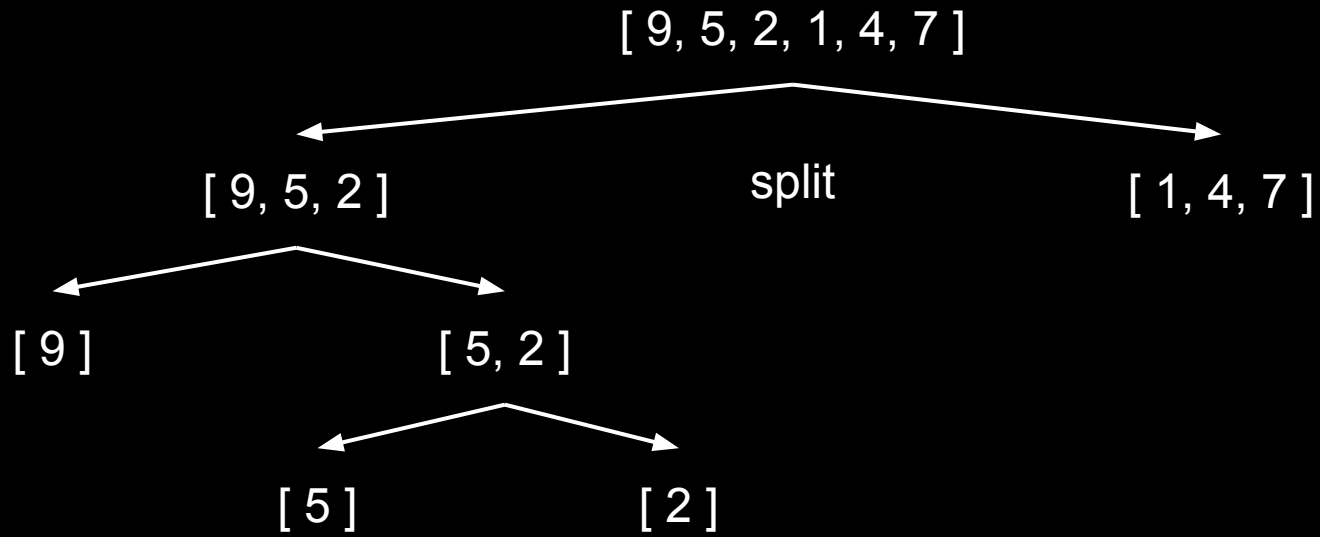
merge sort

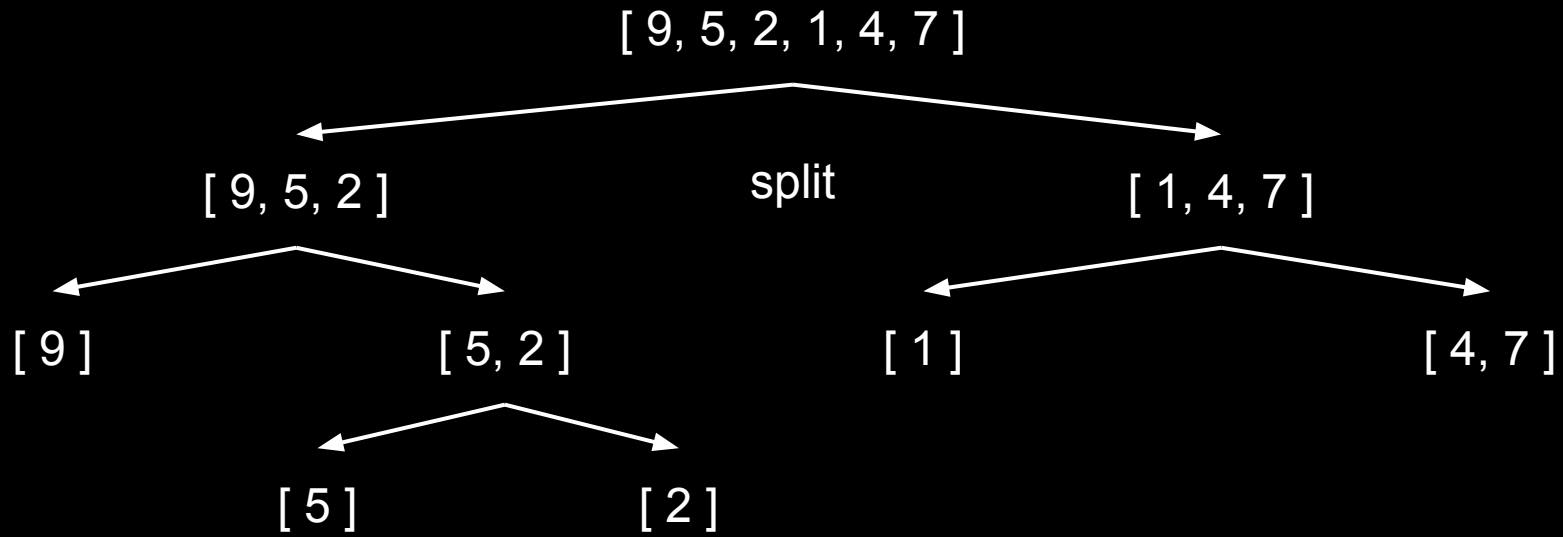
divide the elements recursively in two halves until only one element is left.
then merge the sorted halves back together.

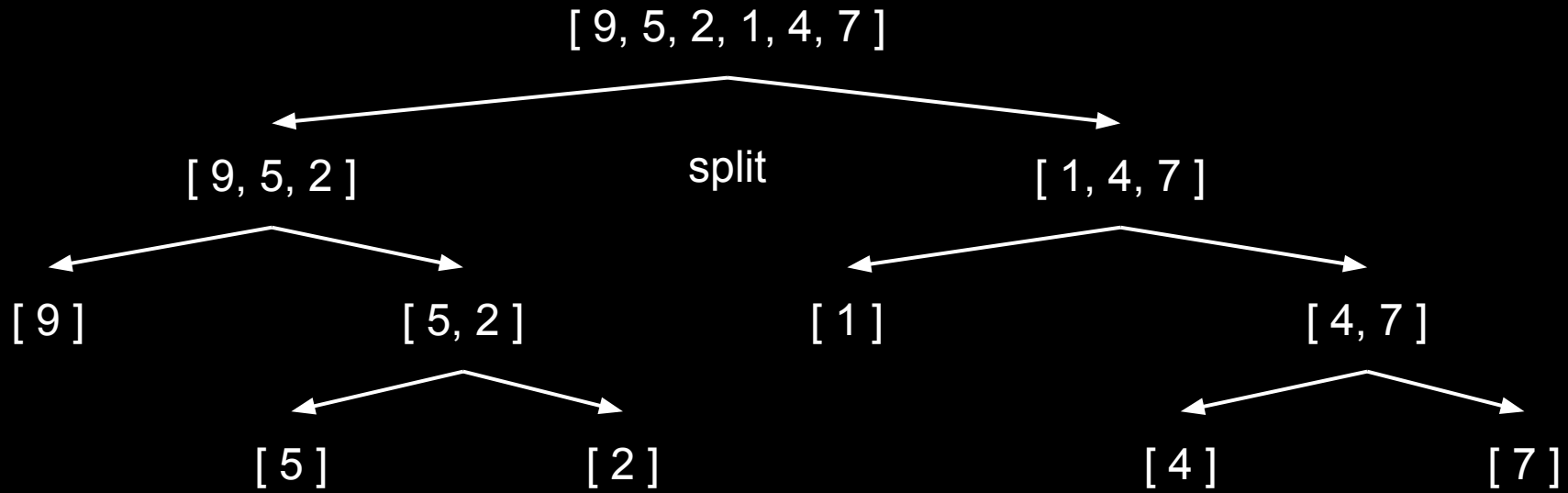
divide the elements **recursively** in two halves until only one element is left.
then merge the sorted halves back together.

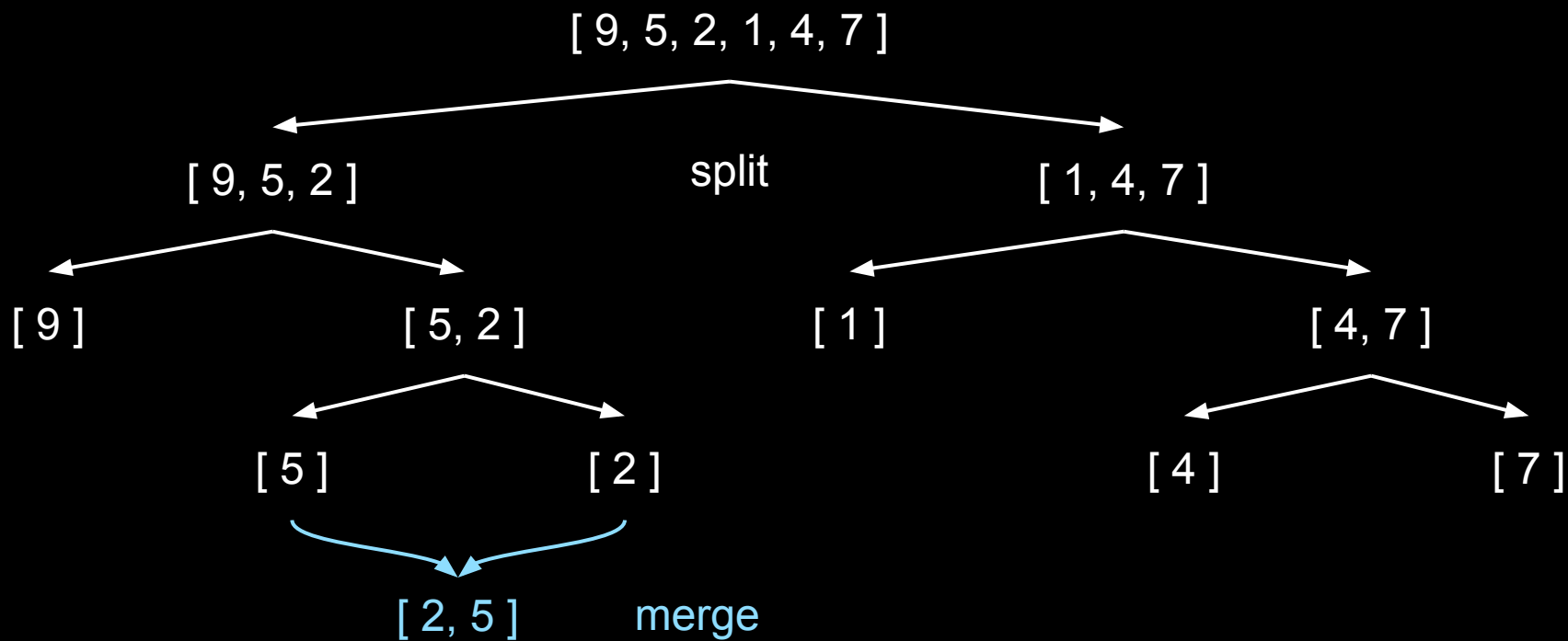


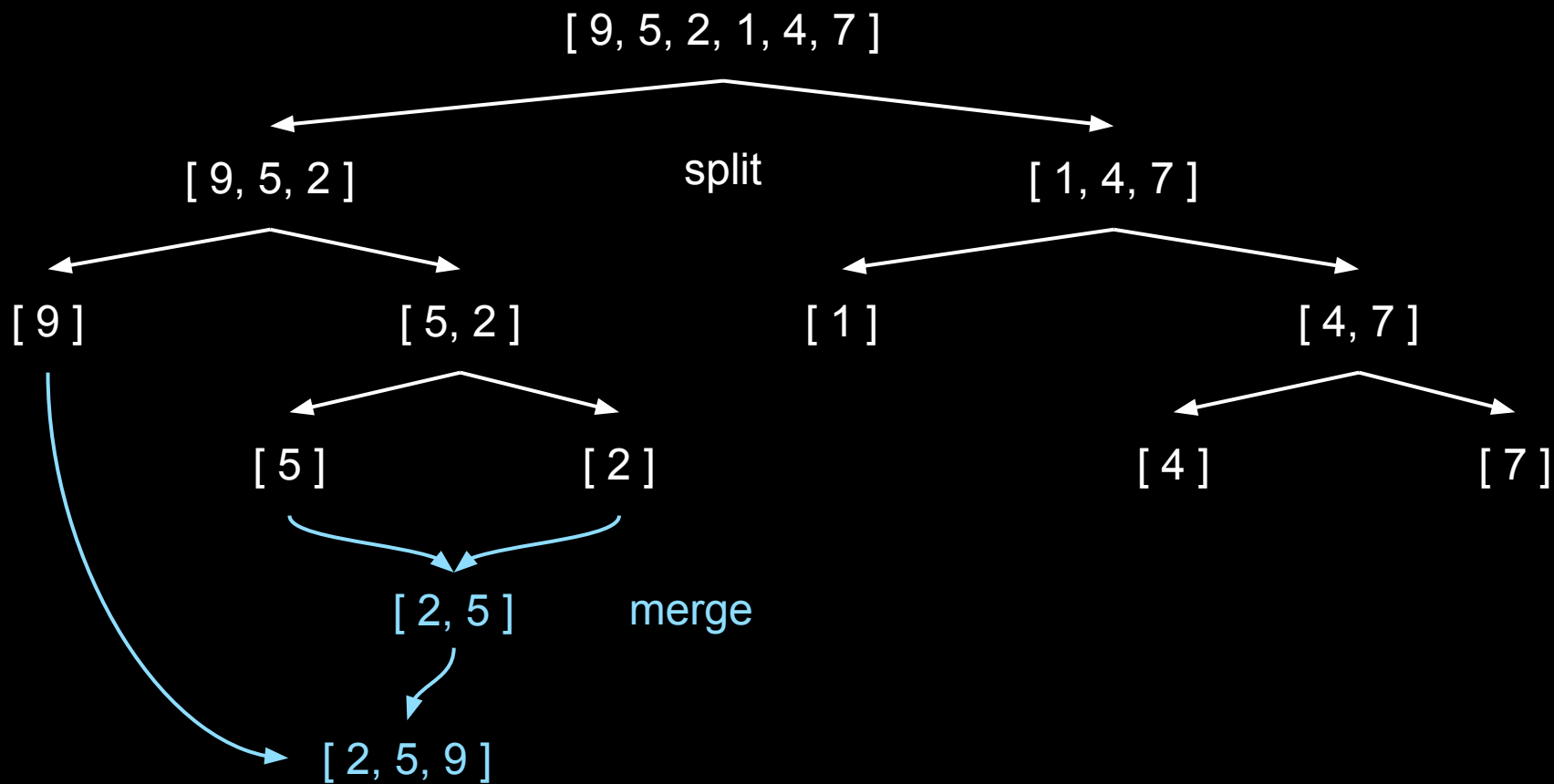


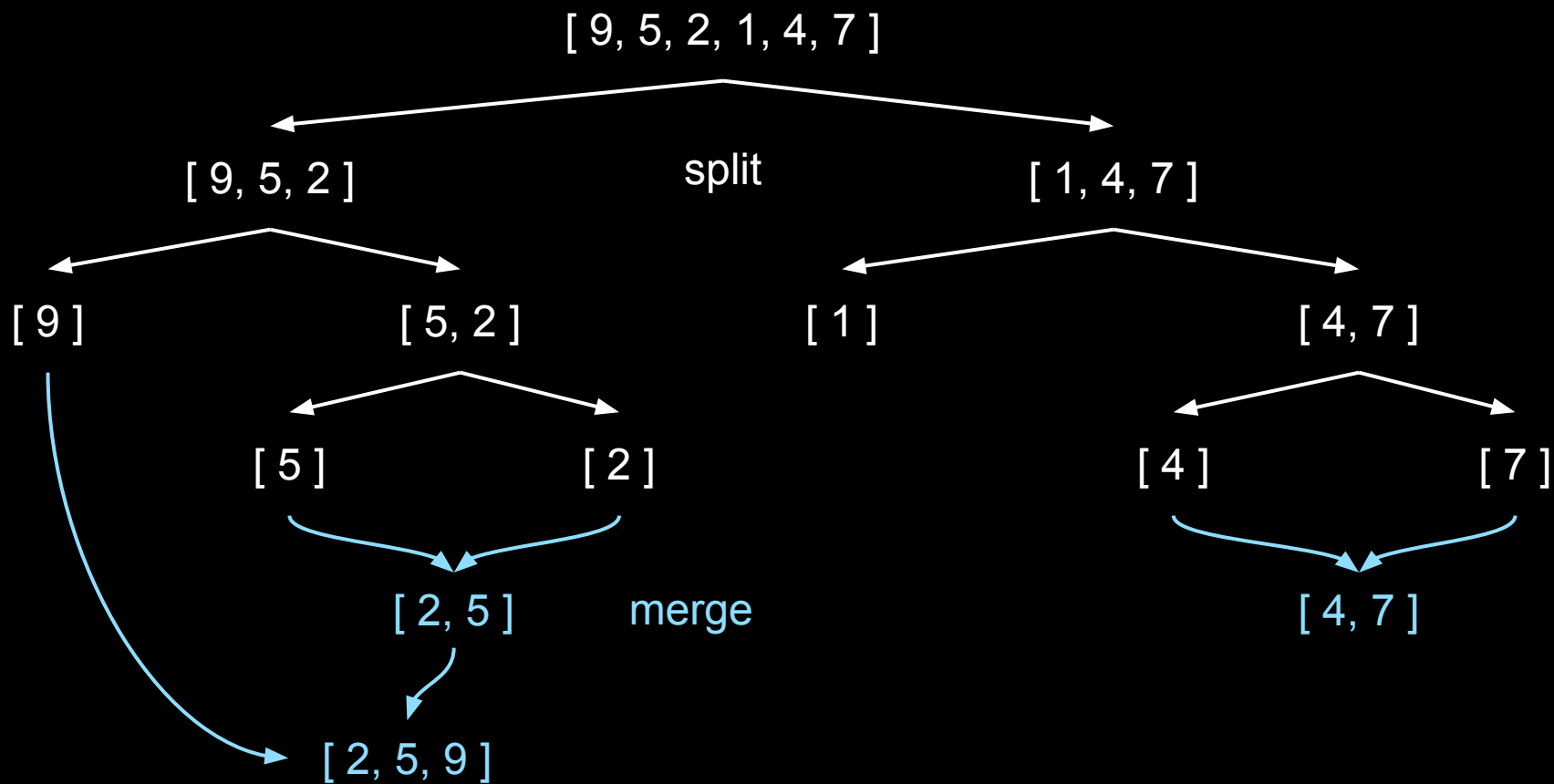


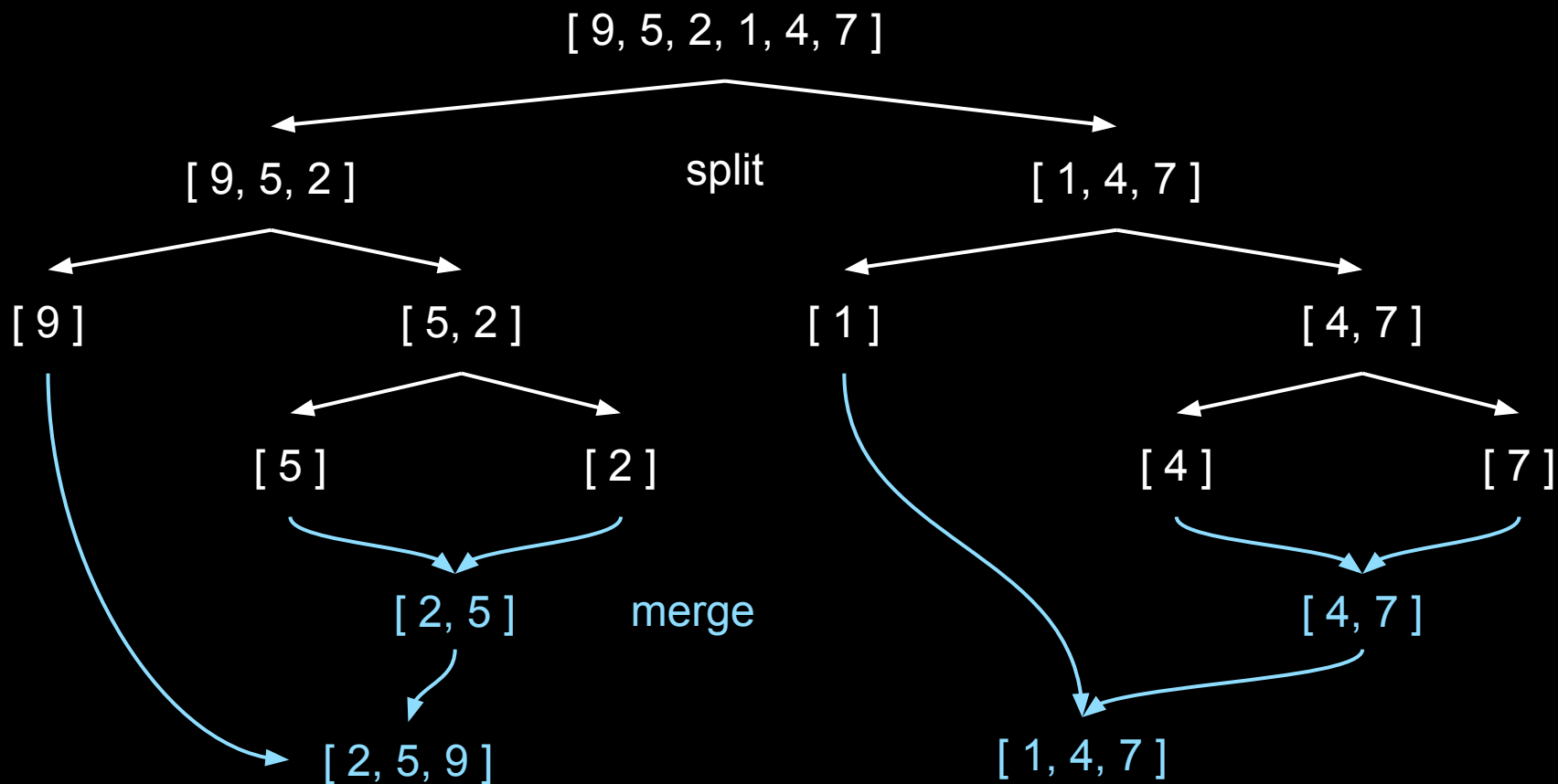


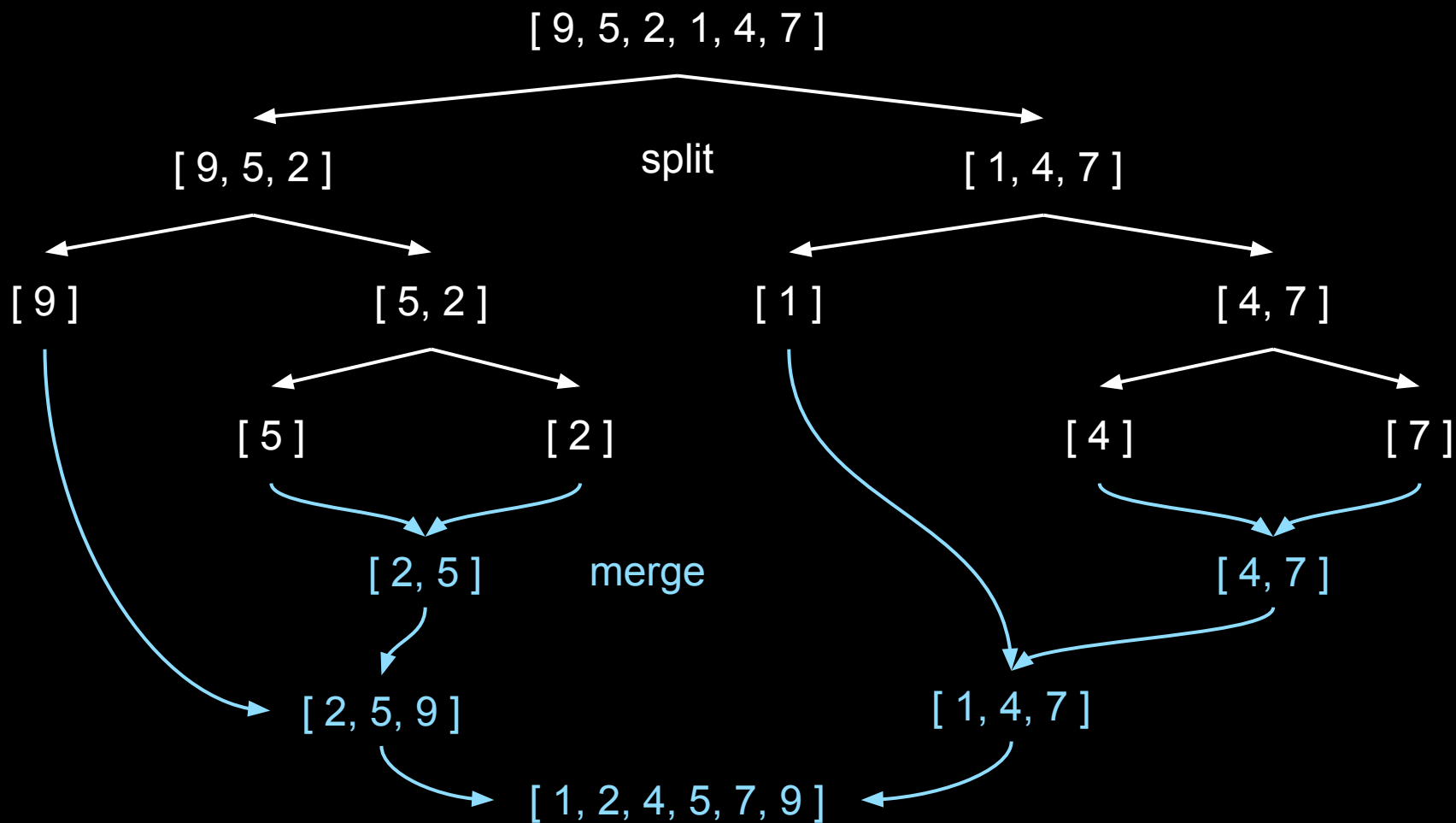




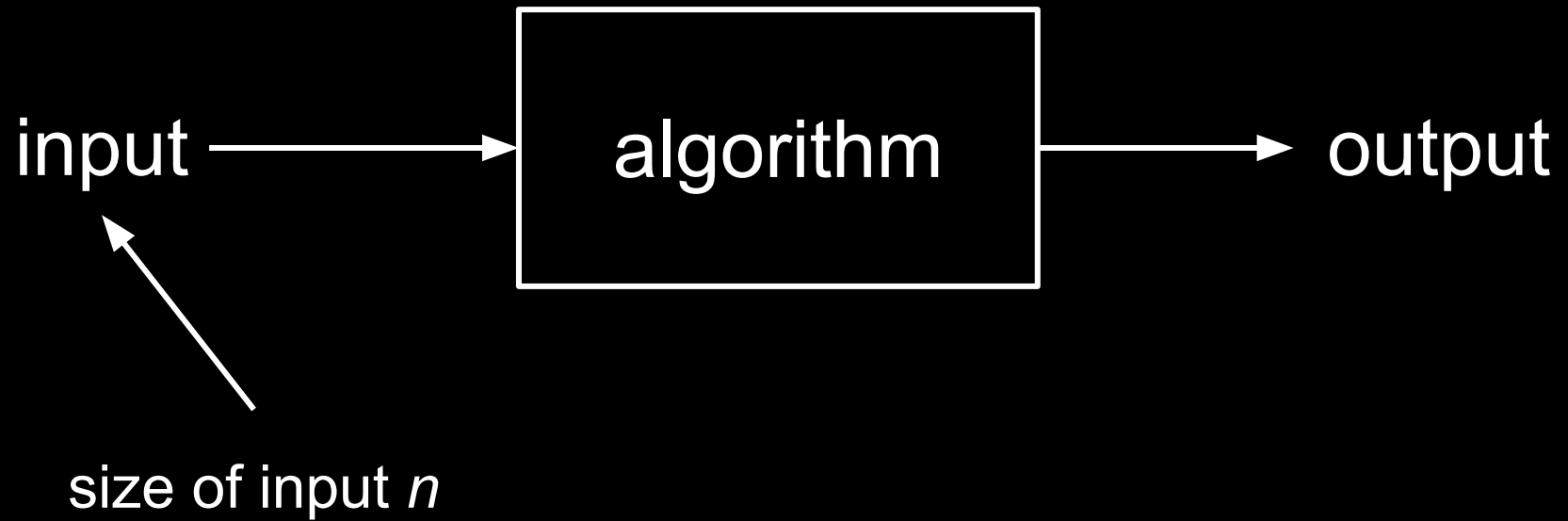


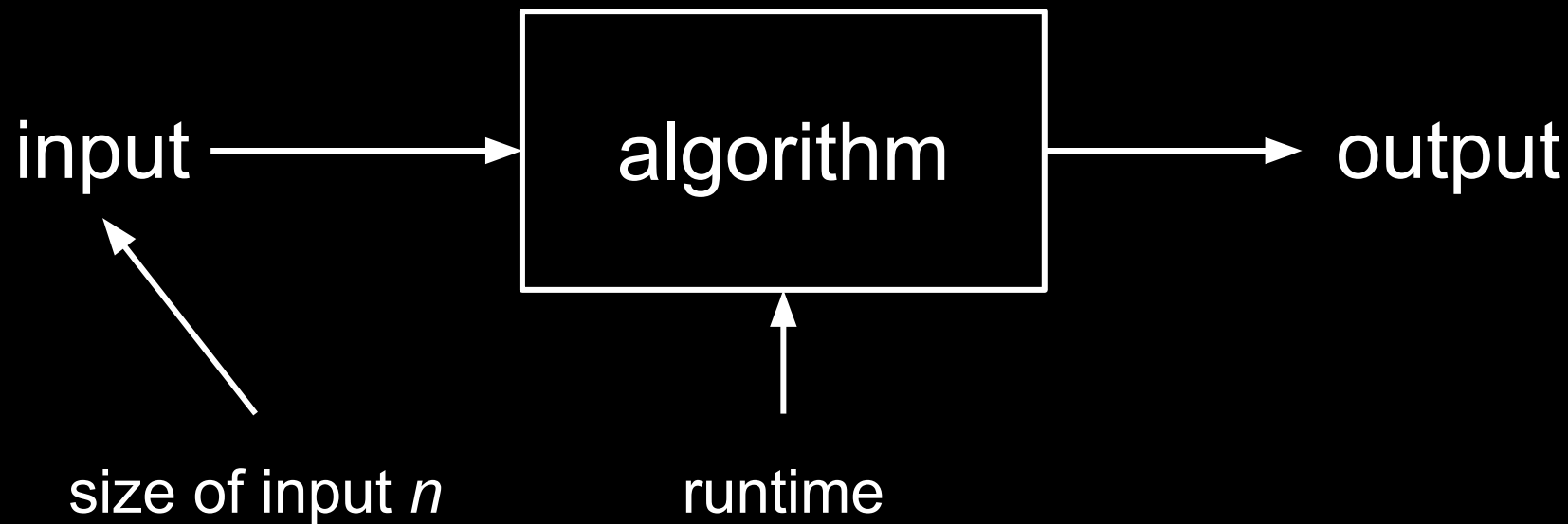




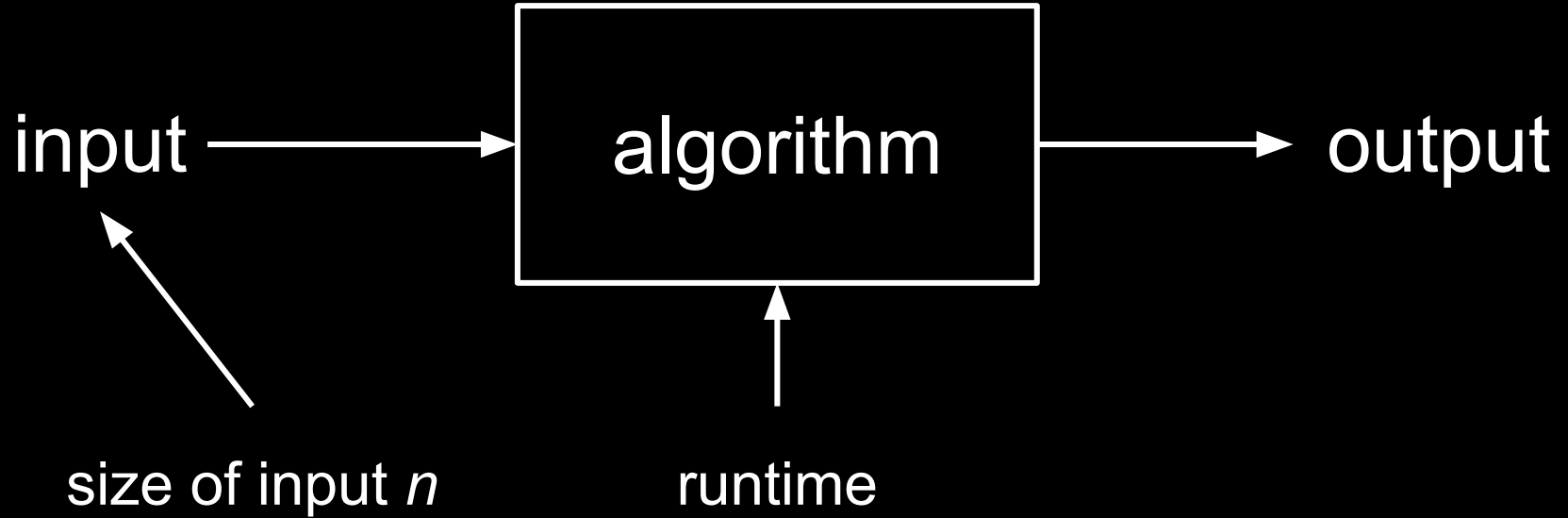


complexity

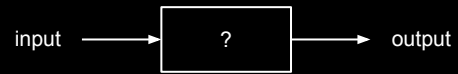




$O(n)$



$O(1)$	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

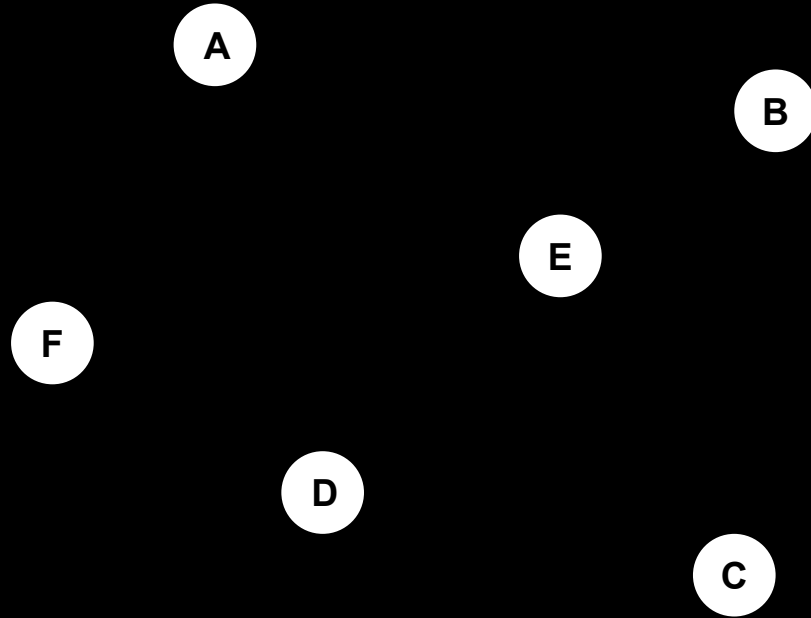


optimization

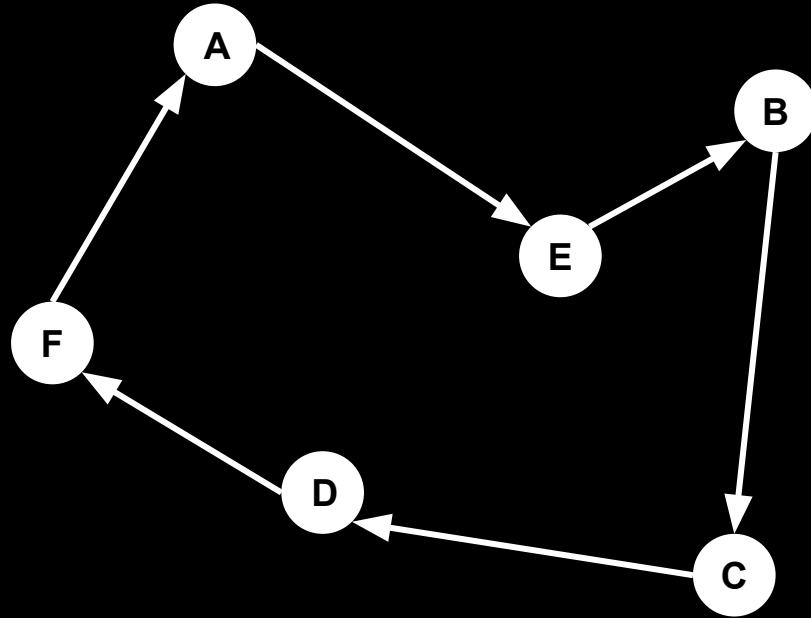


traveling salesmen

shortest tour?



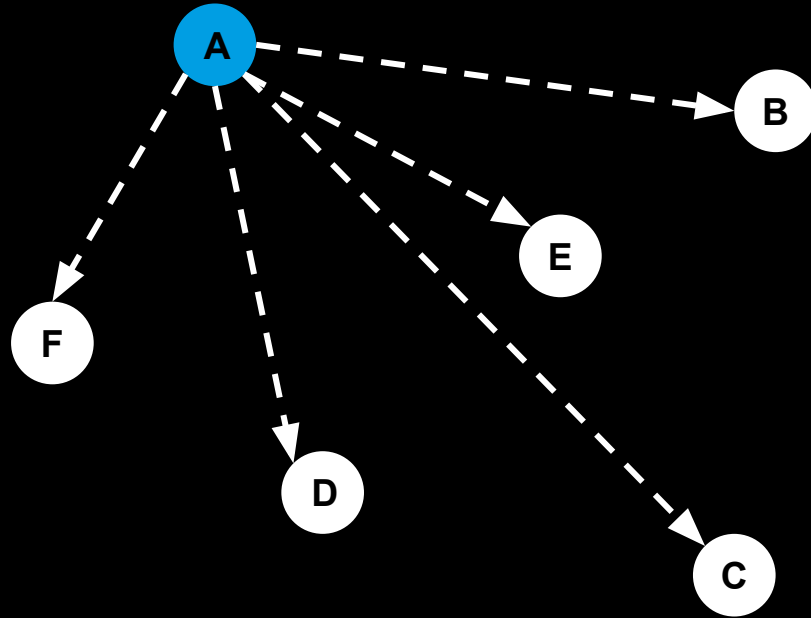
shortest tour?



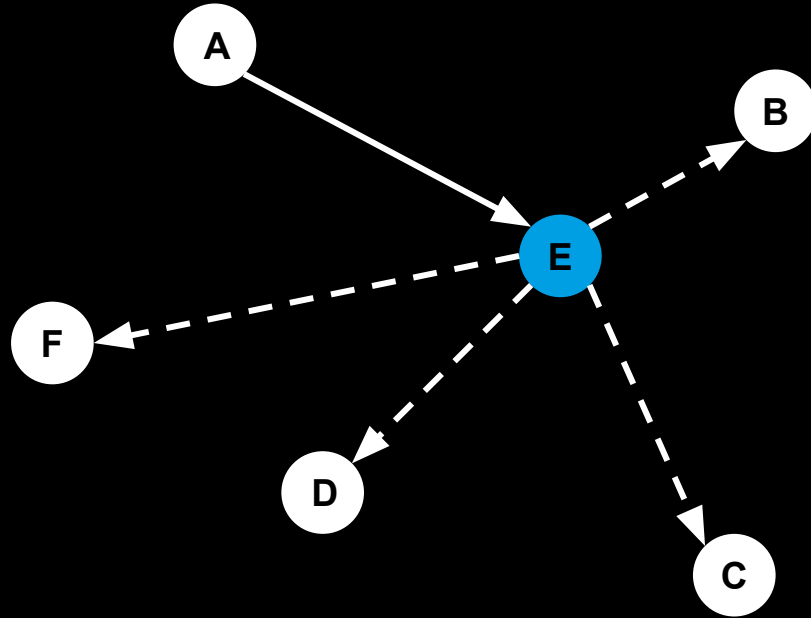
brute force

$$O(n) = n!$$

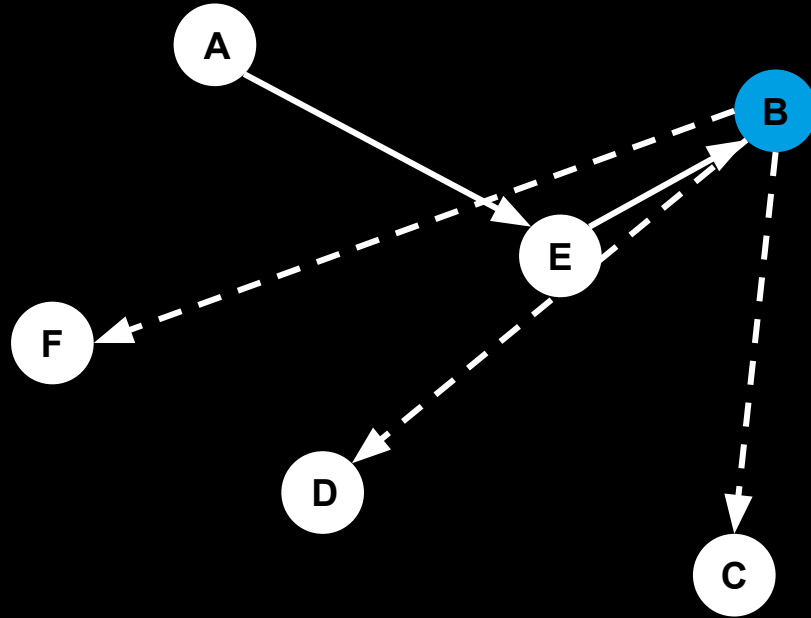
5 possible cities



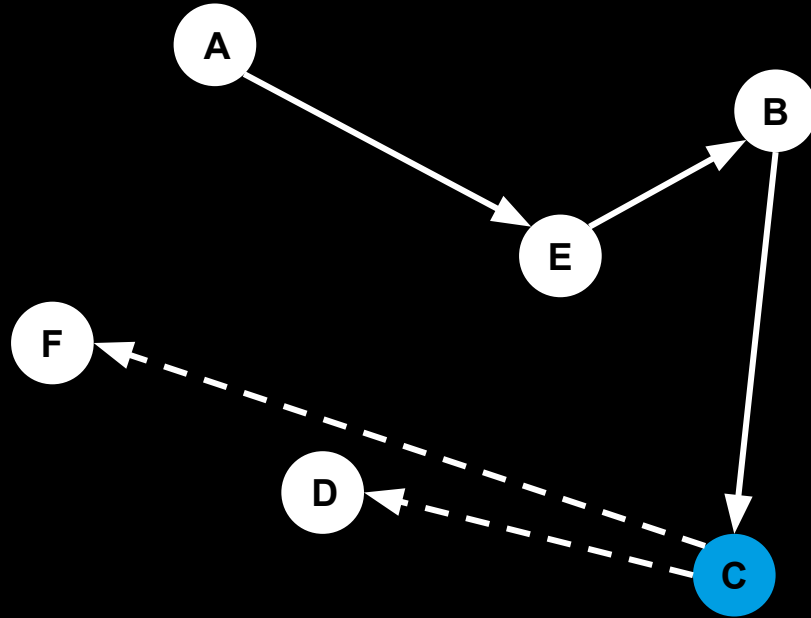
~~5~~ 4 possible cities



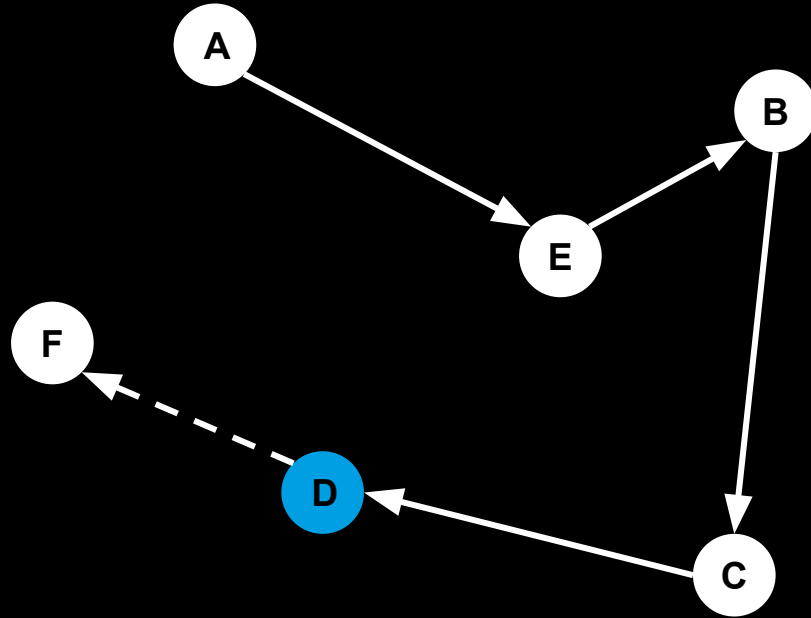
5 4 3 possible cities



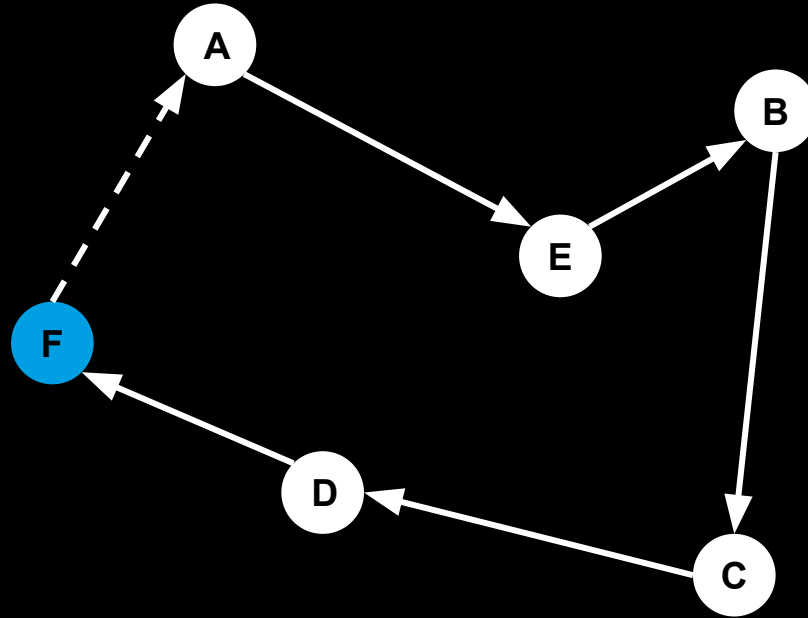
~~5~~ 4 3 2 possible cities



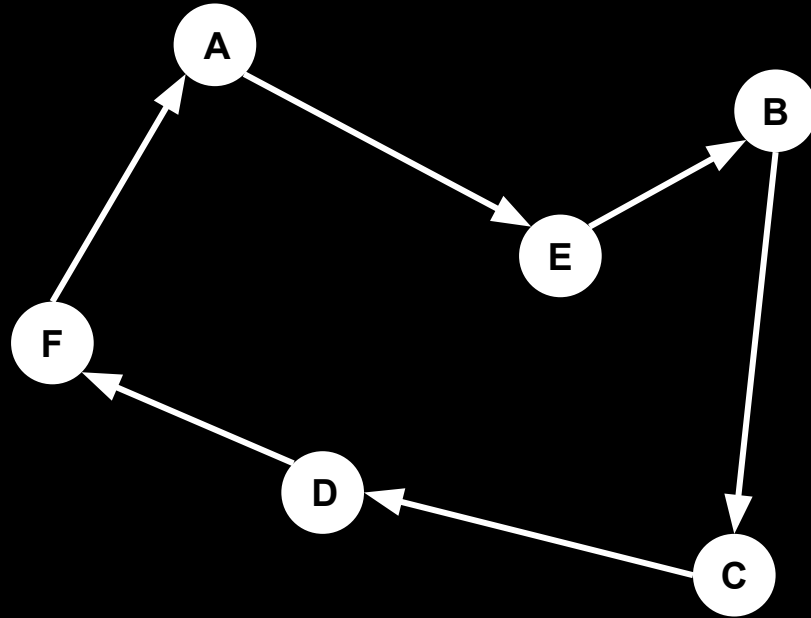
5 4 3 2 1 possible city



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

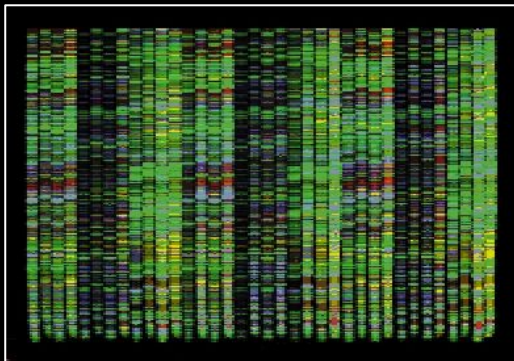


Image source: [IEEE](#)



Image source: [VDI Nachrichten](#)



Image source: [Wikimedia](#)

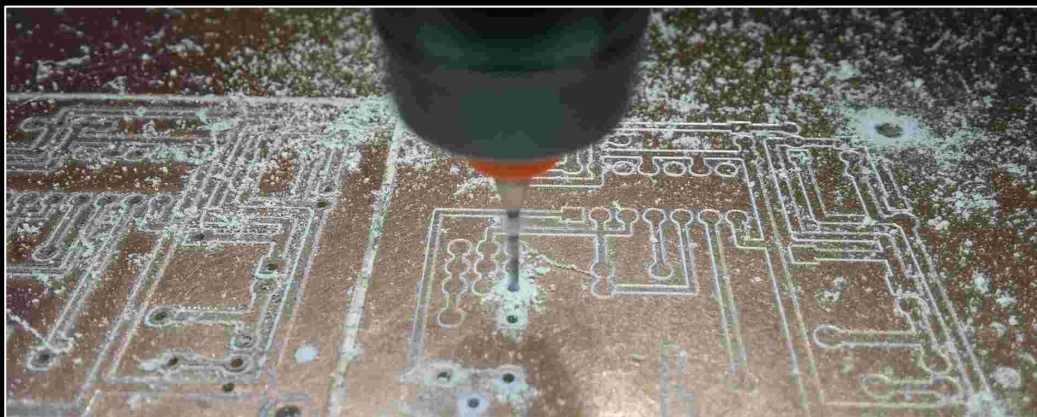
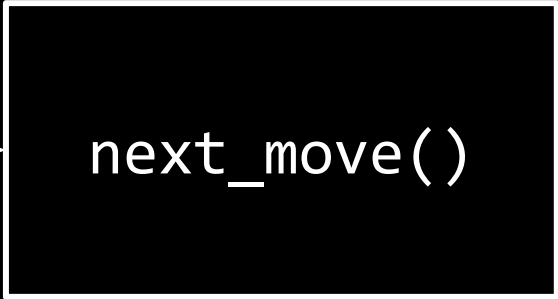


Image source: <https://github.com/meiyi1986/tutorials/blob/master/notebooks/img/pcb-drilling.jpeg>



Image source: [IAS Observatory](#)

but sometimes, brute force works perfectly

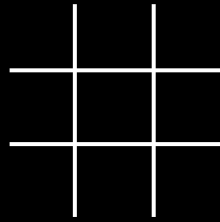


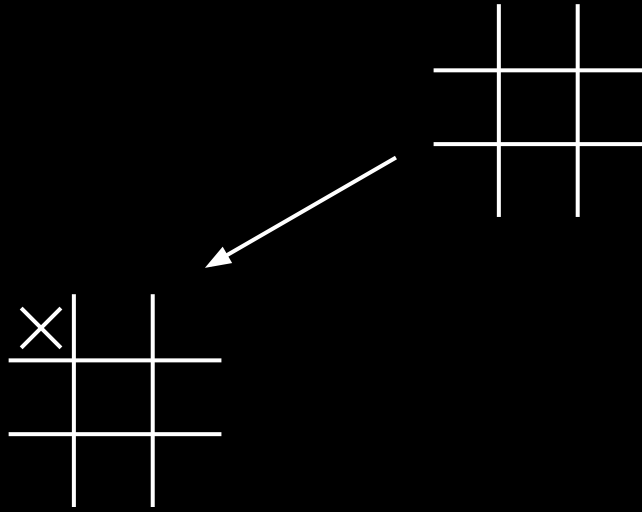
(2, 2)

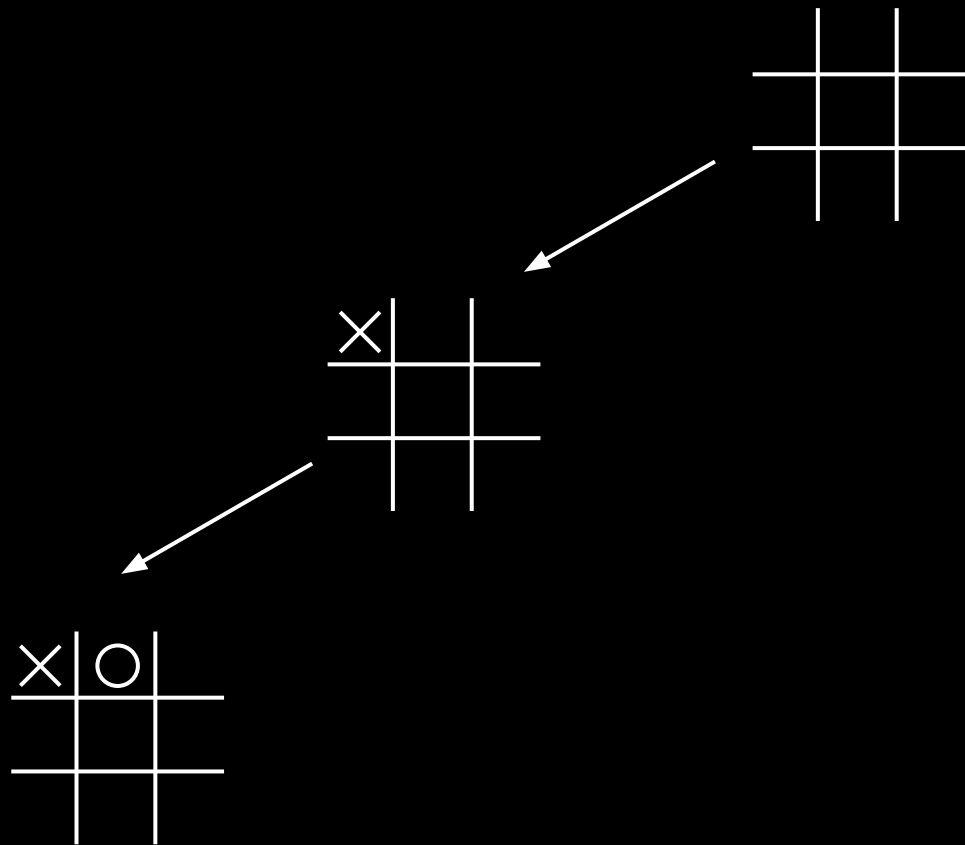
brute-force search:

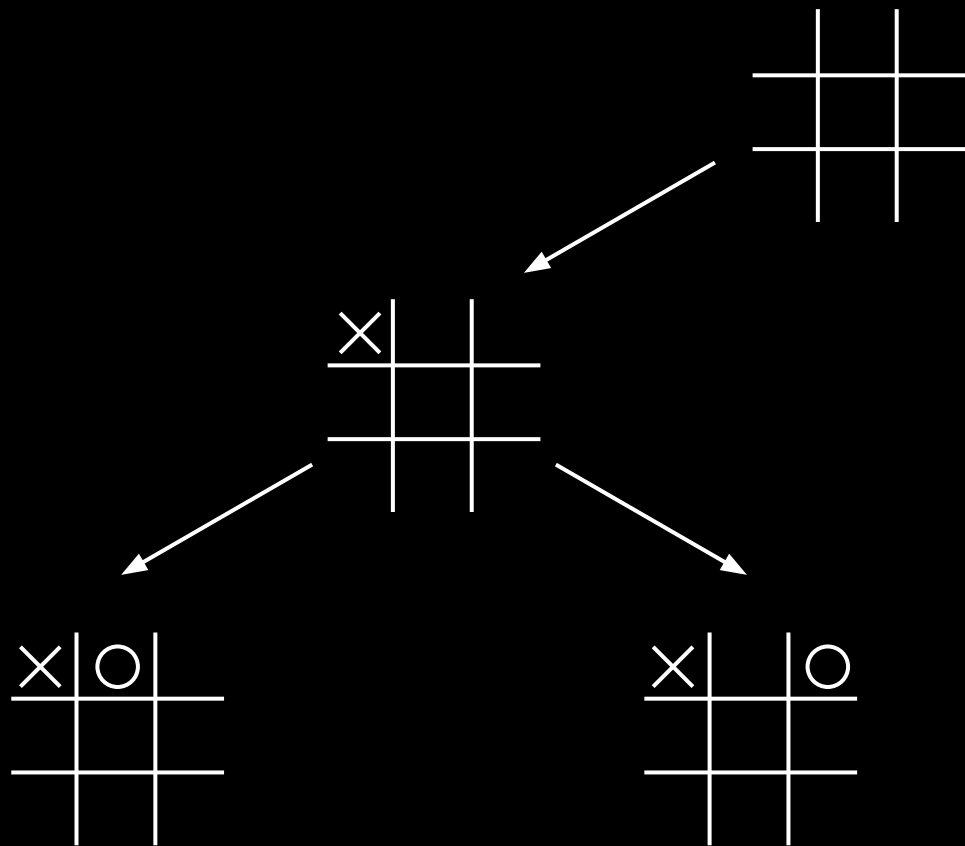
evaluate all possible move sequences.

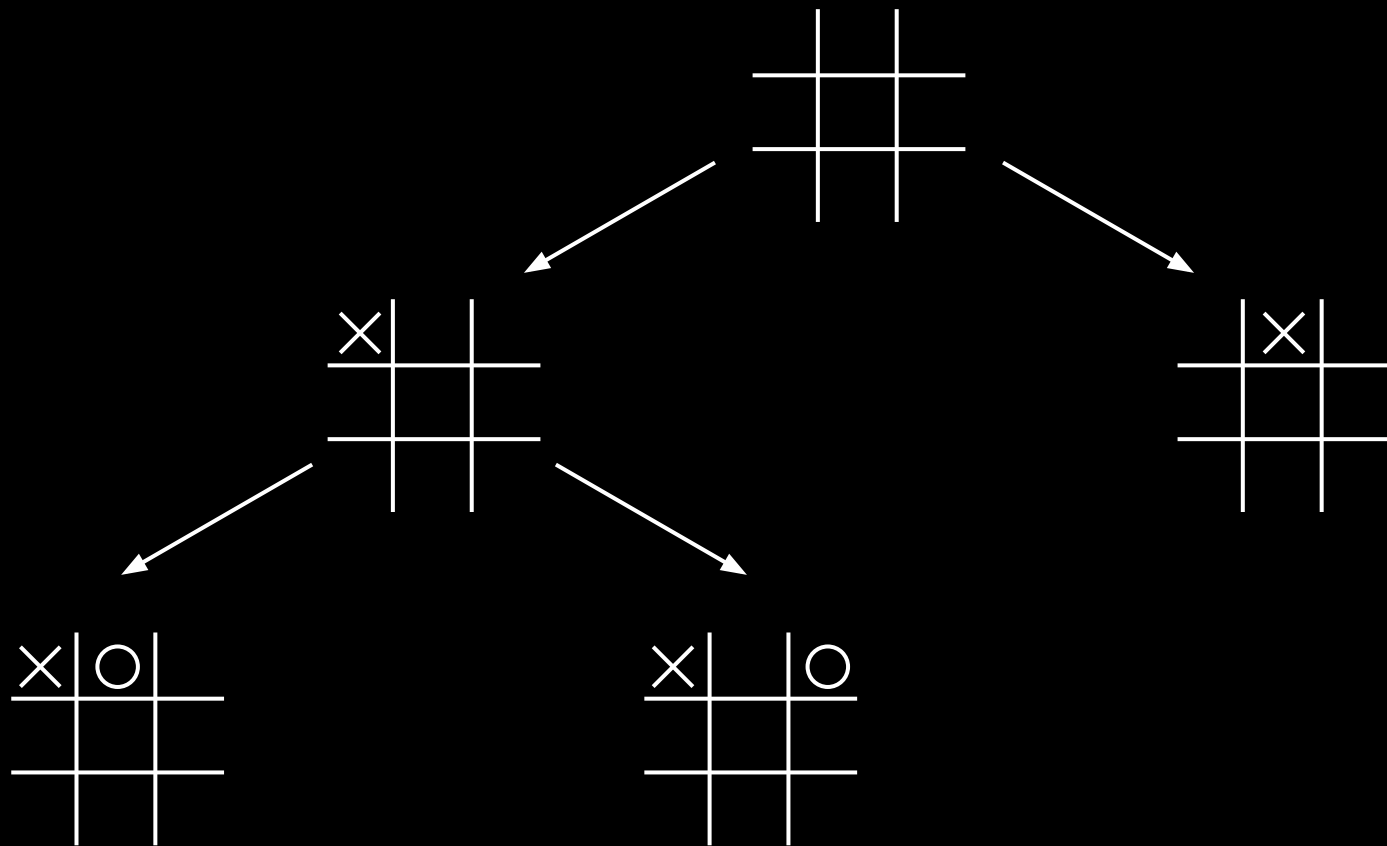
choose the move with the best value.

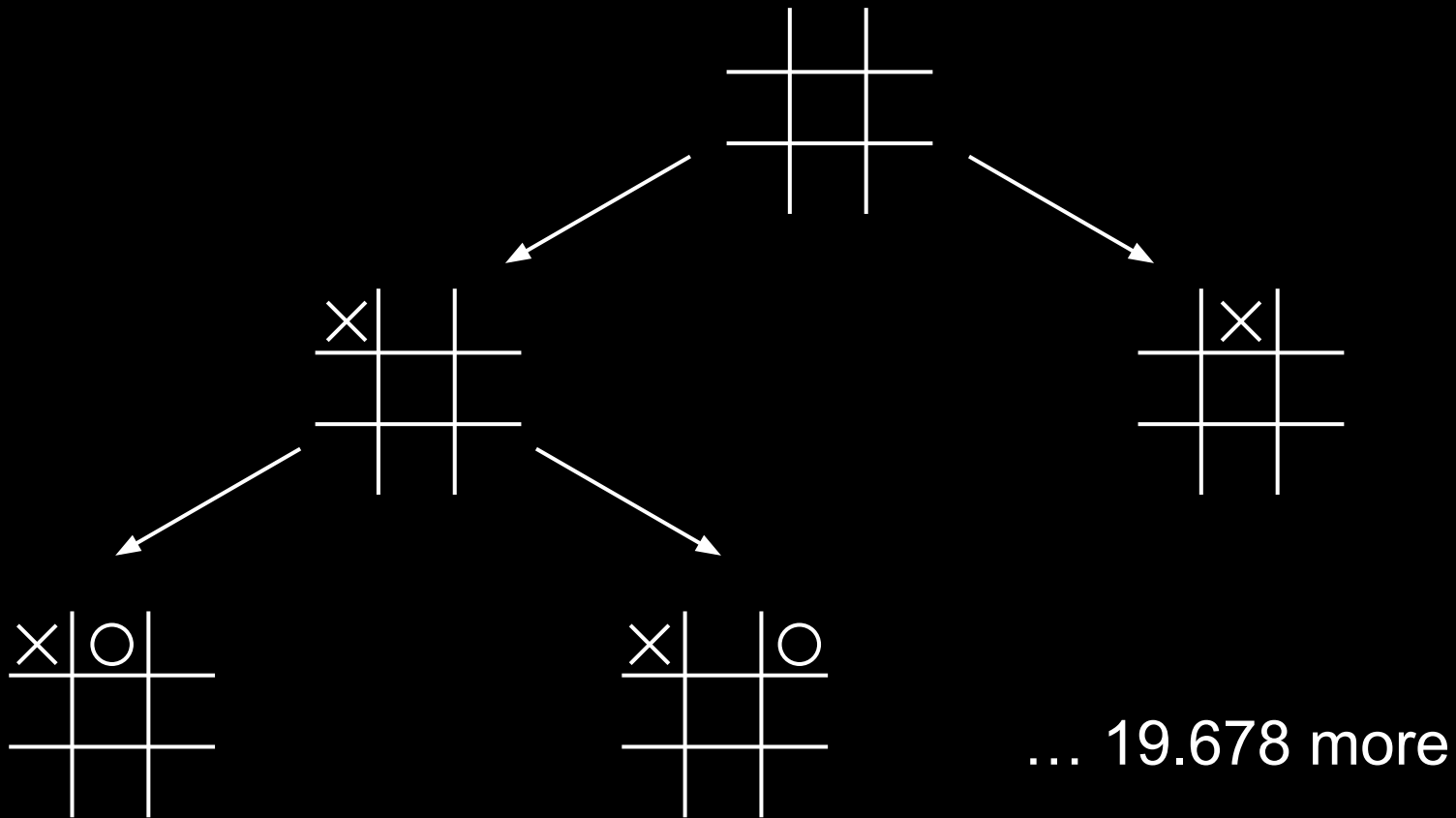






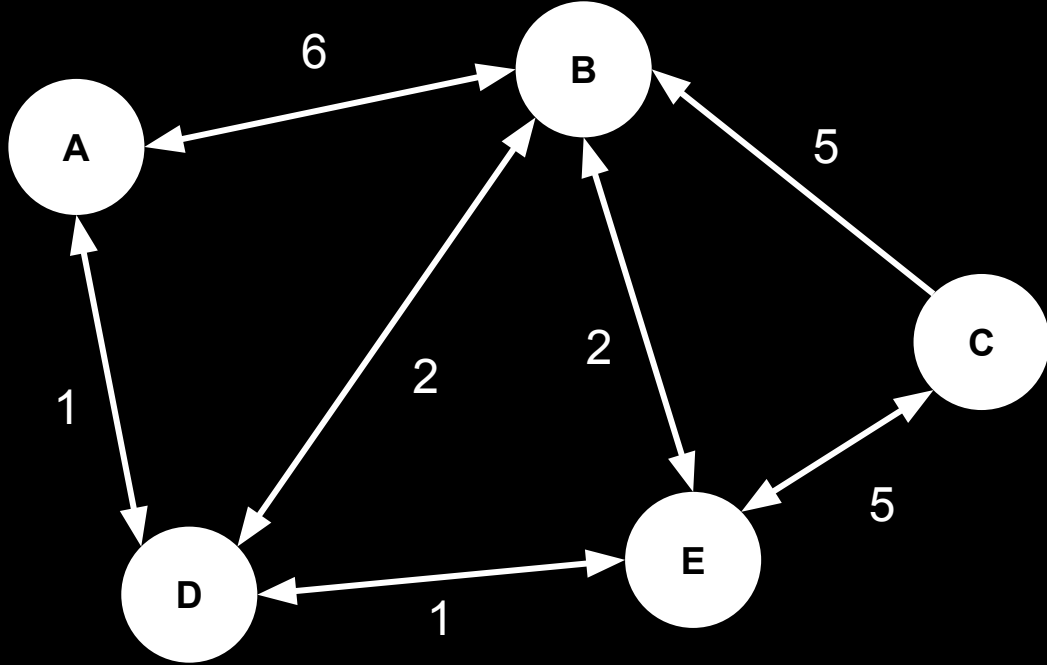








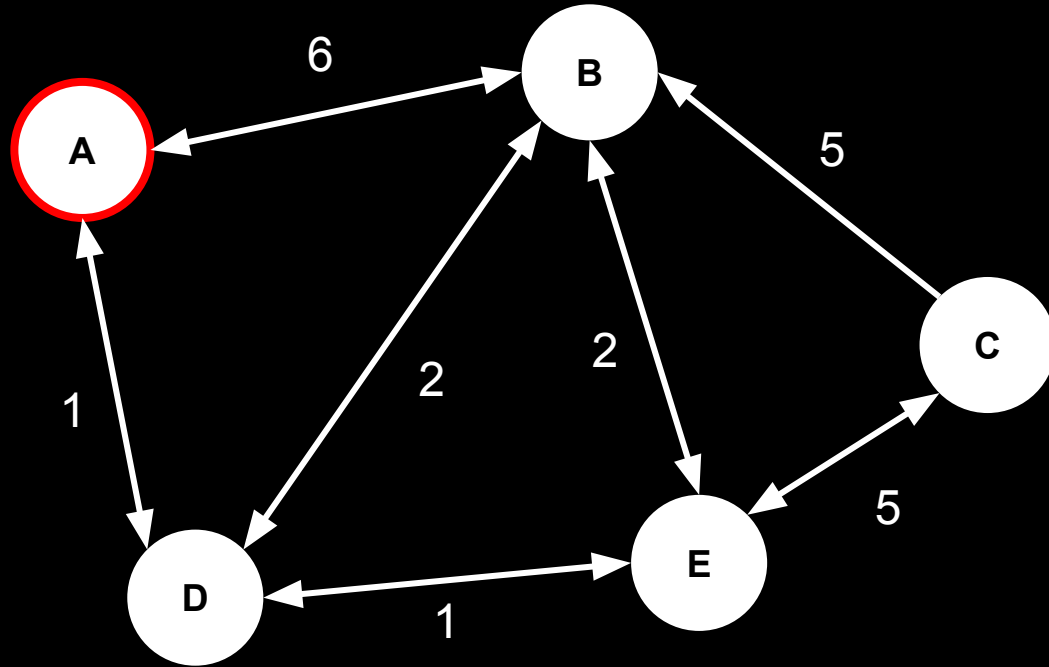
shortest paths



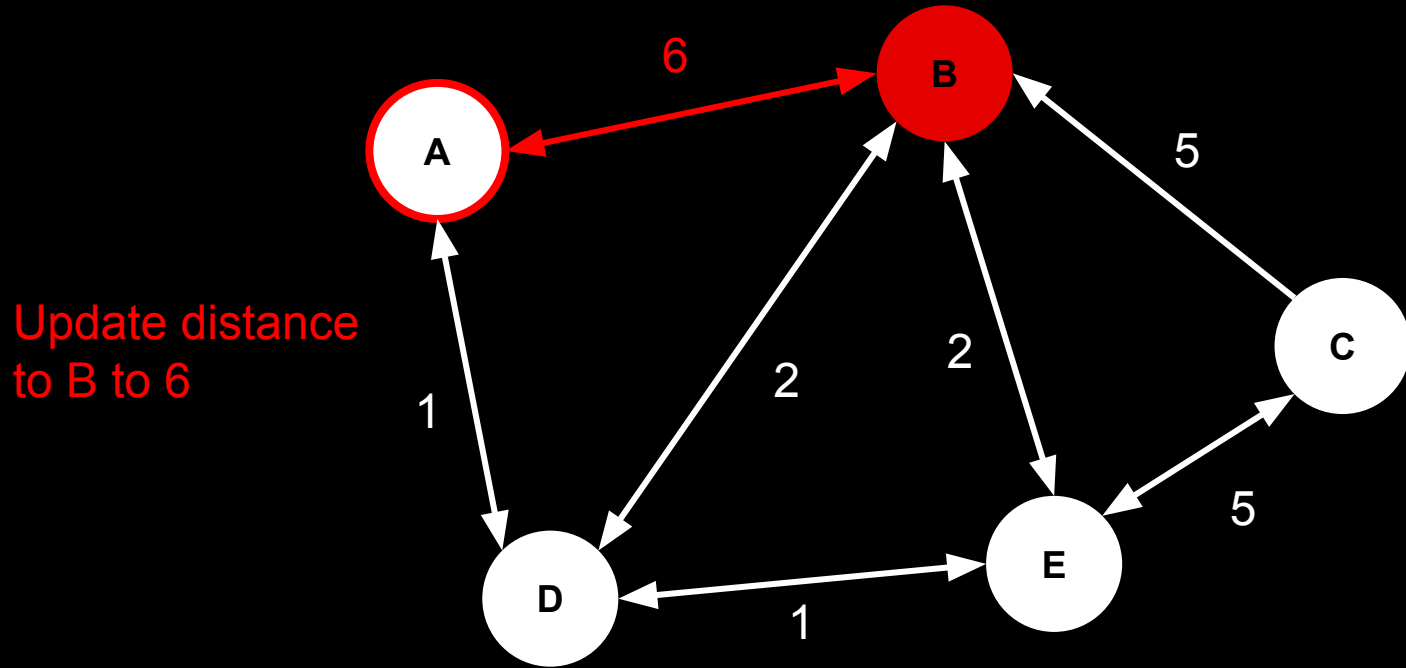
dijkstra's algorithm

Distances: $A = 0$, $B = \infty$, $C = \infty$, $D = \infty$, $E = \infty$

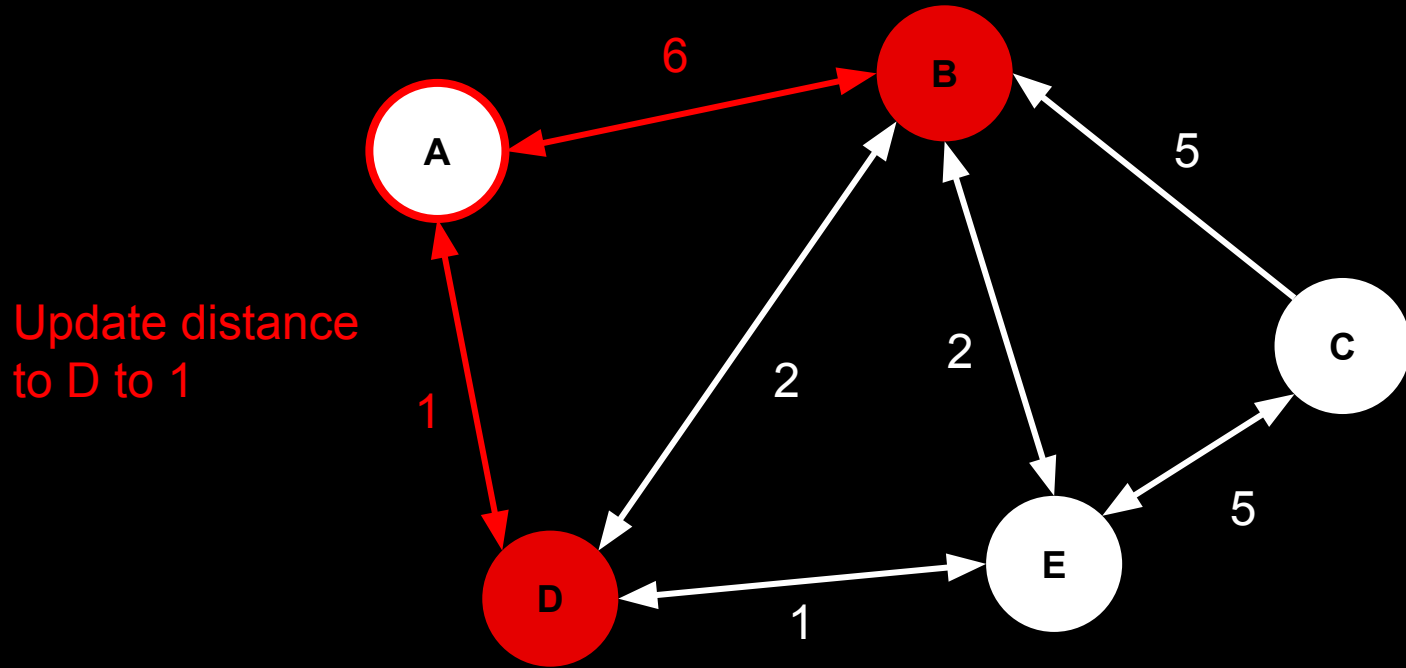
Set A as first
location



Distances: $A = 0$, $B = 6$, $C = \infty$, $D = \infty$, $E = \infty$



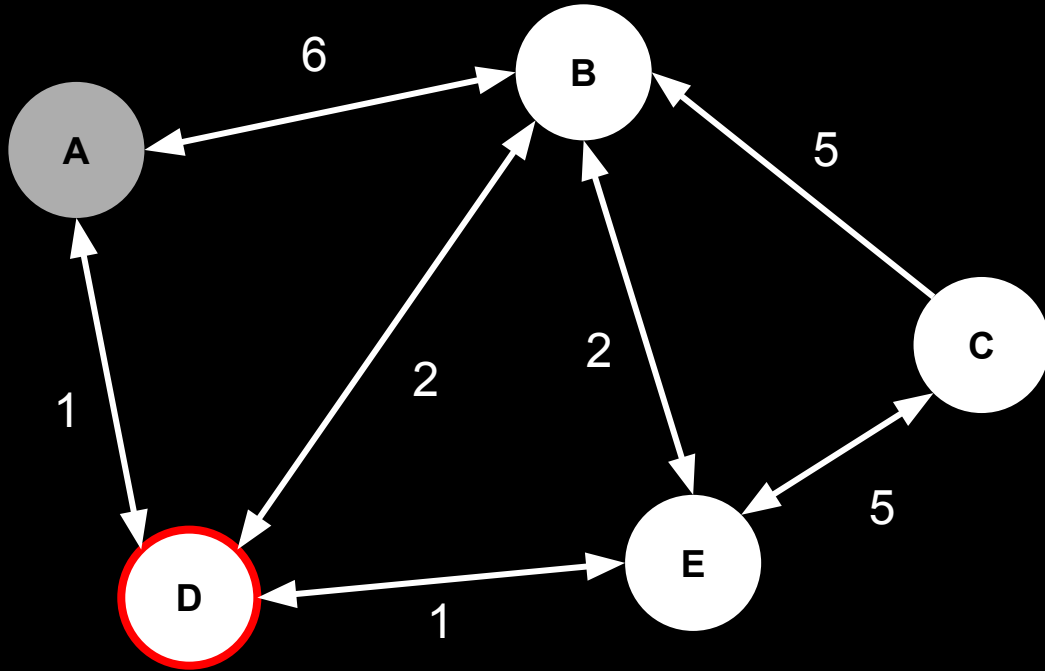
Distances: $A = 0$, $B = 6$, $C = \infty$, $D = 1$, $E = \infty$



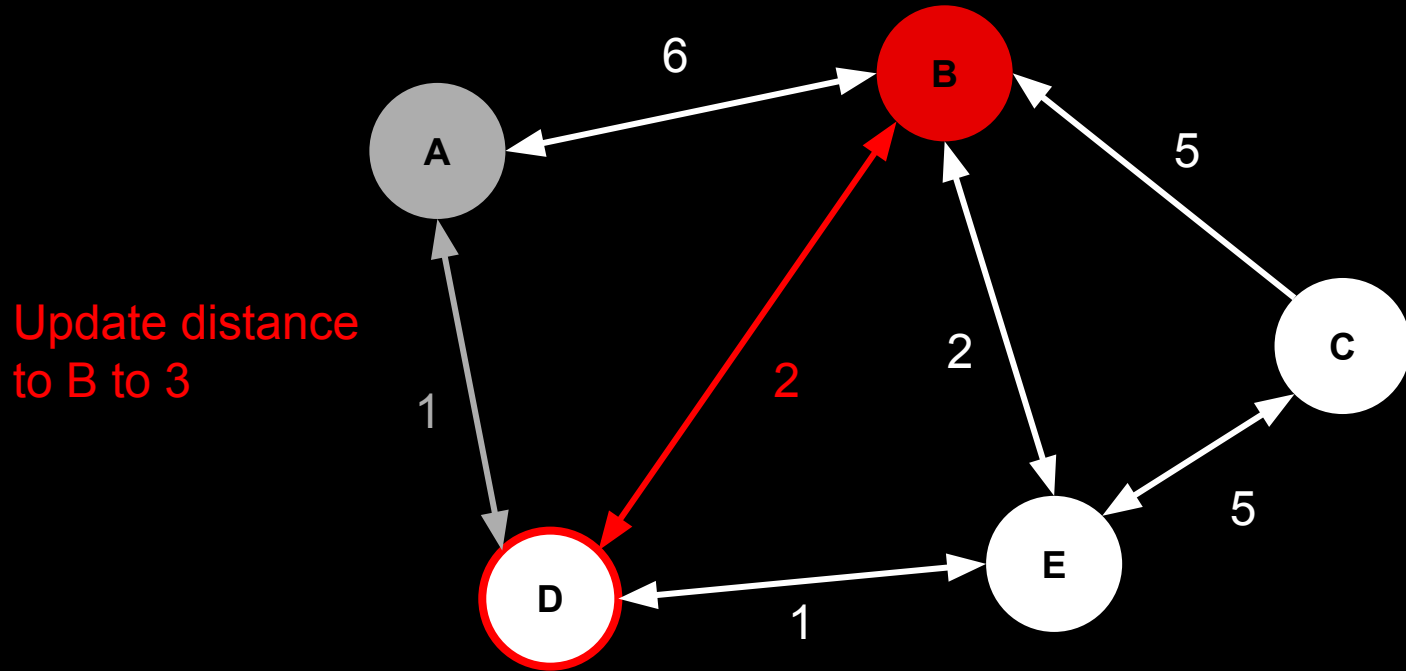
Distances: A = 0, B = 6, C = ∞ , D = 1, E = ∞

Move to D as
next location

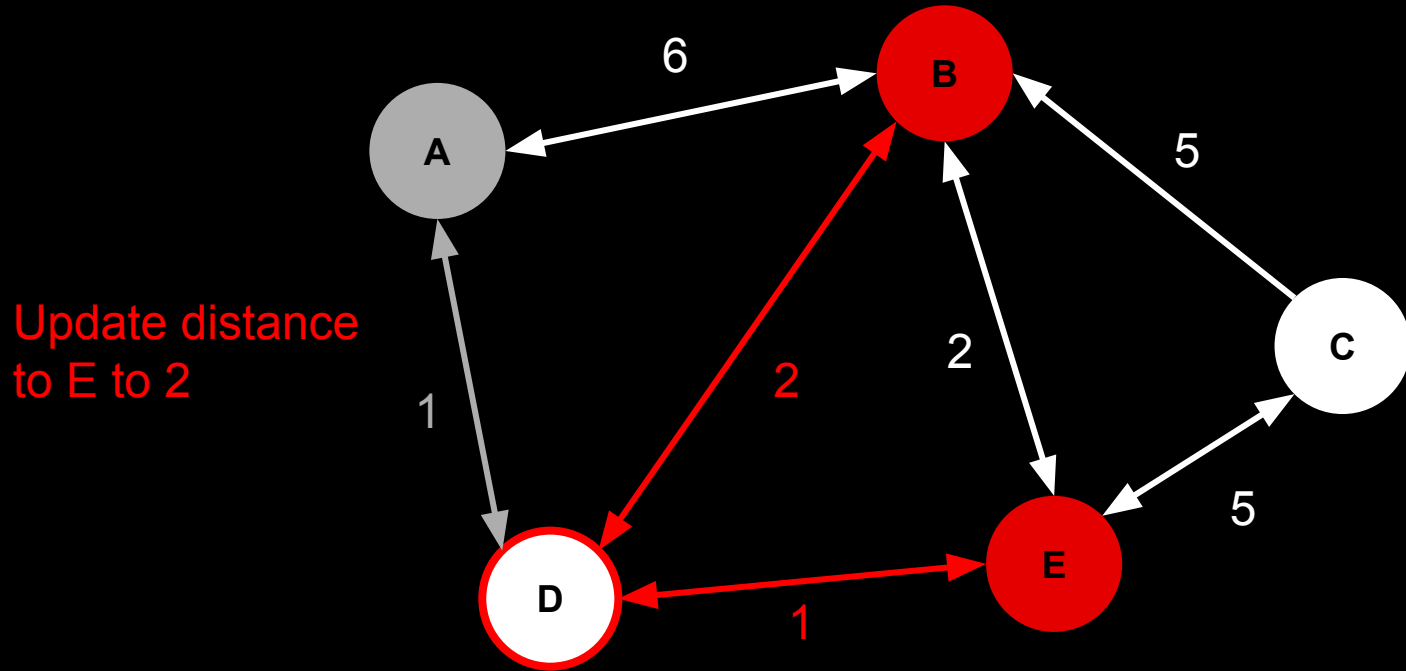
Mark A as
visited



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



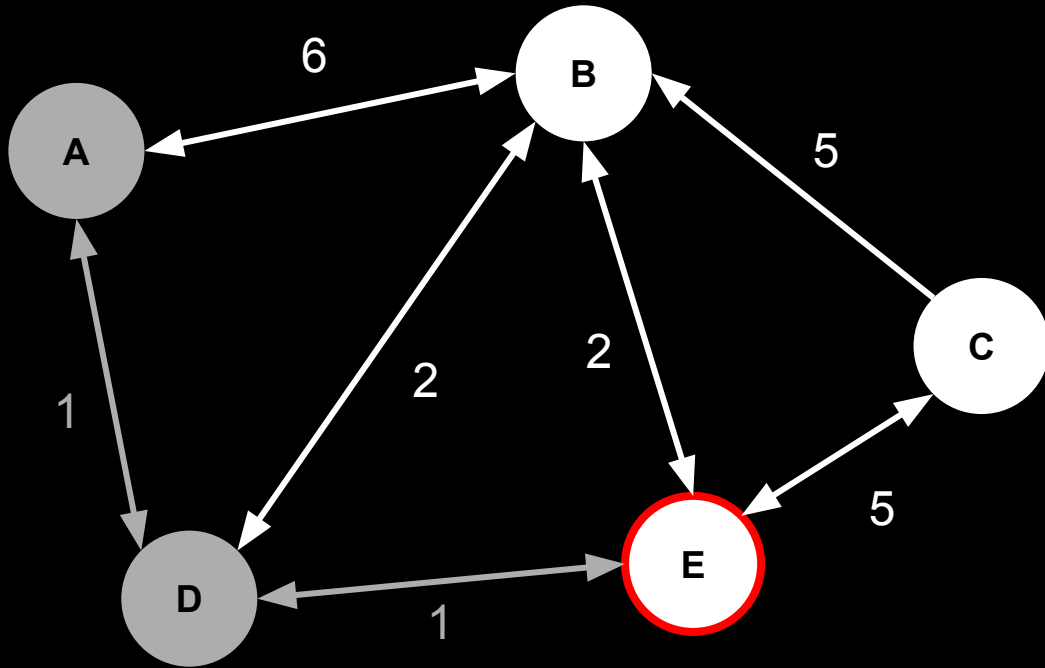
Distances: A = 0, B = 3, C = ∞ , D = 1, E = 2



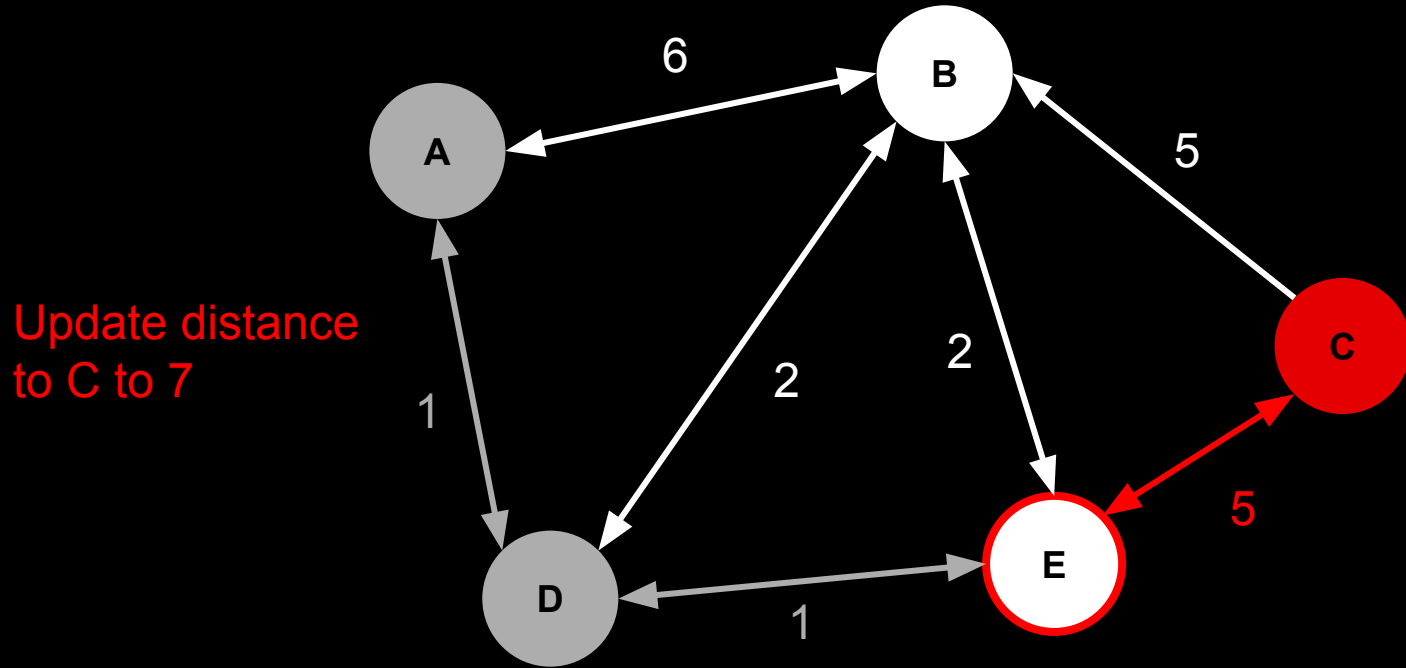
Distances: A = 0, B = 3, C = ∞ , D = 1, **E = 2**

**Move to E as
next location**

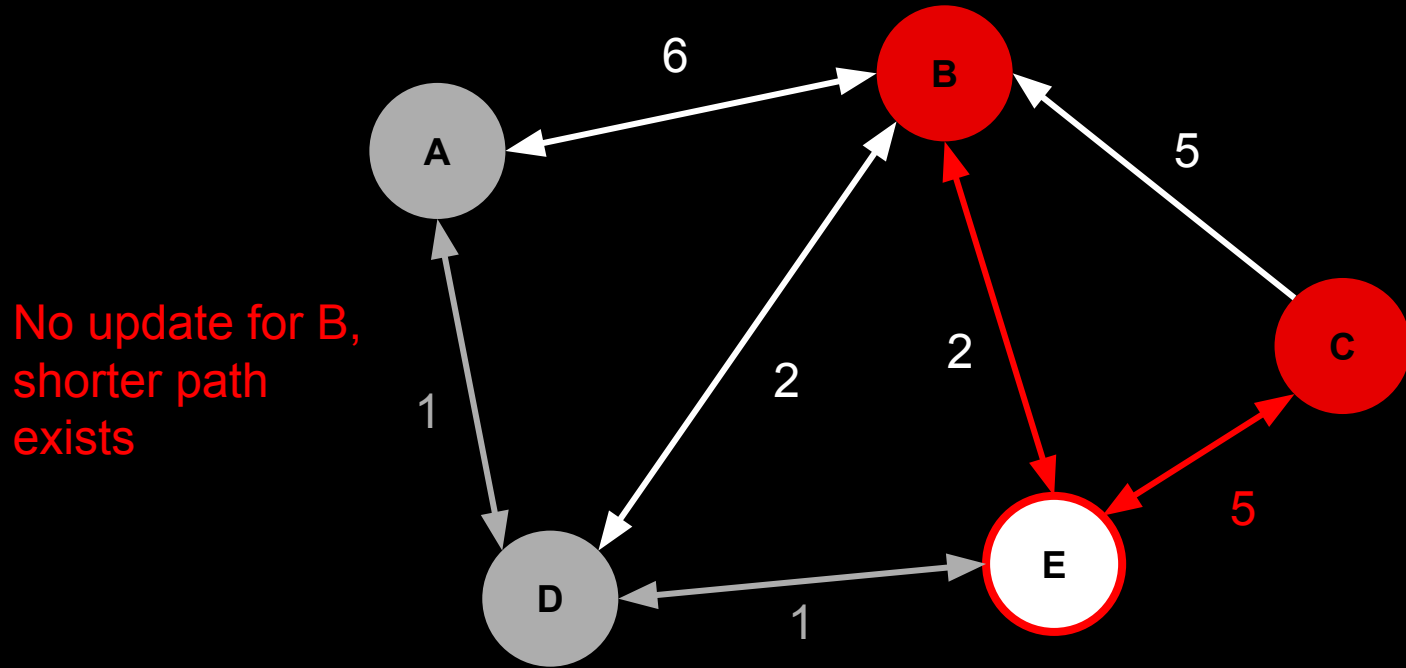
Mark D as
visited



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



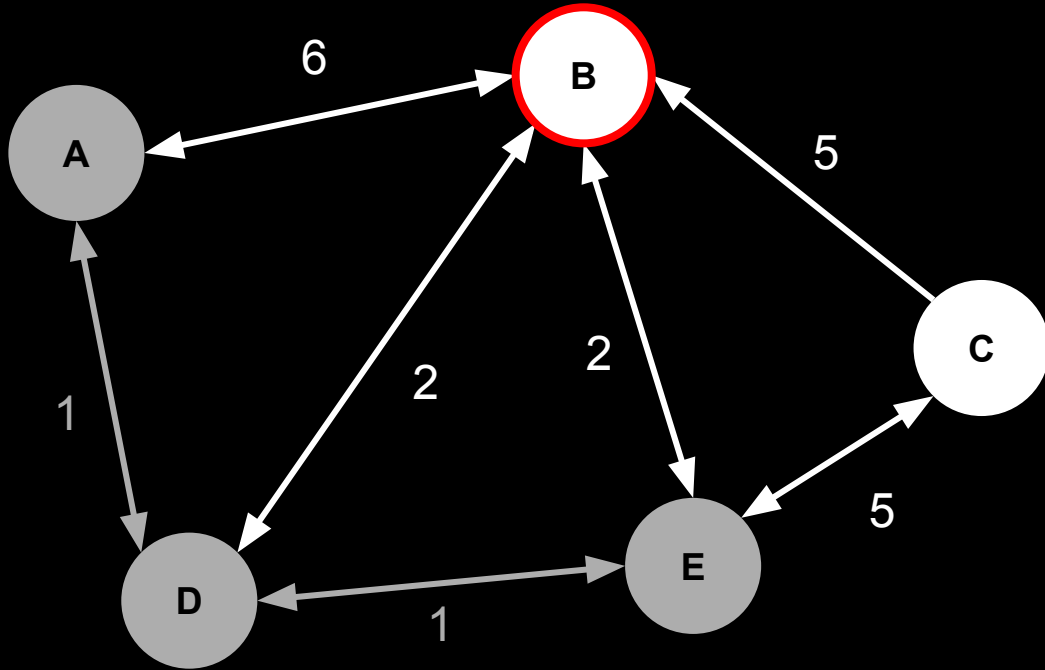
Distances: A = 0, B = 3, C = 7, D = 1, E = 2



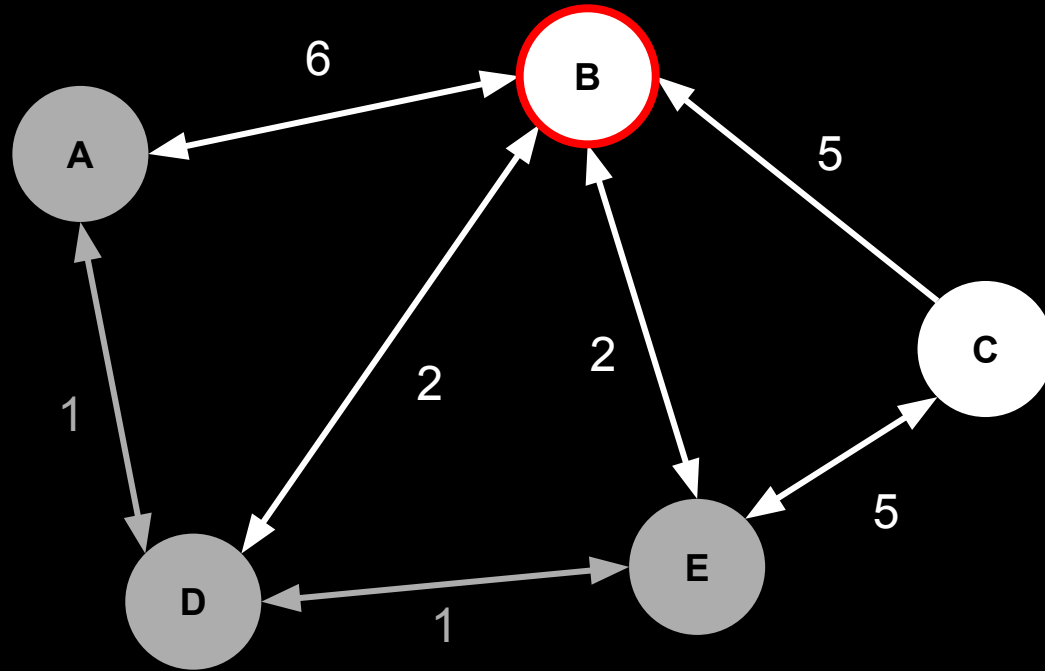
Distances: A = 0, B = 3, C = 7, D = 1, E = 2

Move to B as
new location

Mark E as
visited



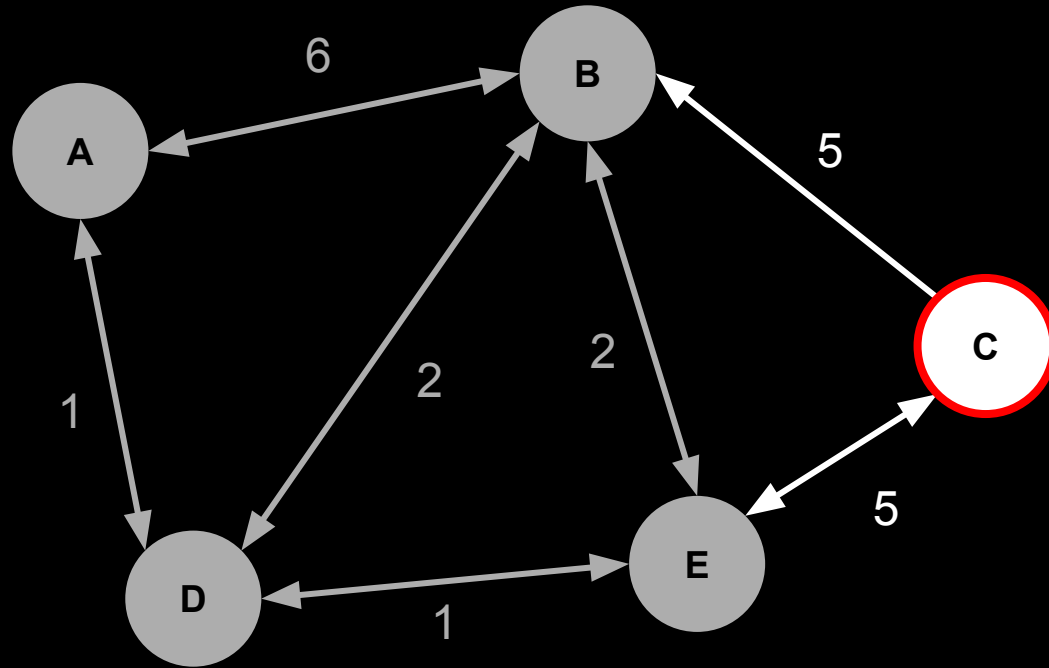
Distances: A = 0, **B = 3**, C = 7, D = 1, E = 2



No further
locations
reachable
from B

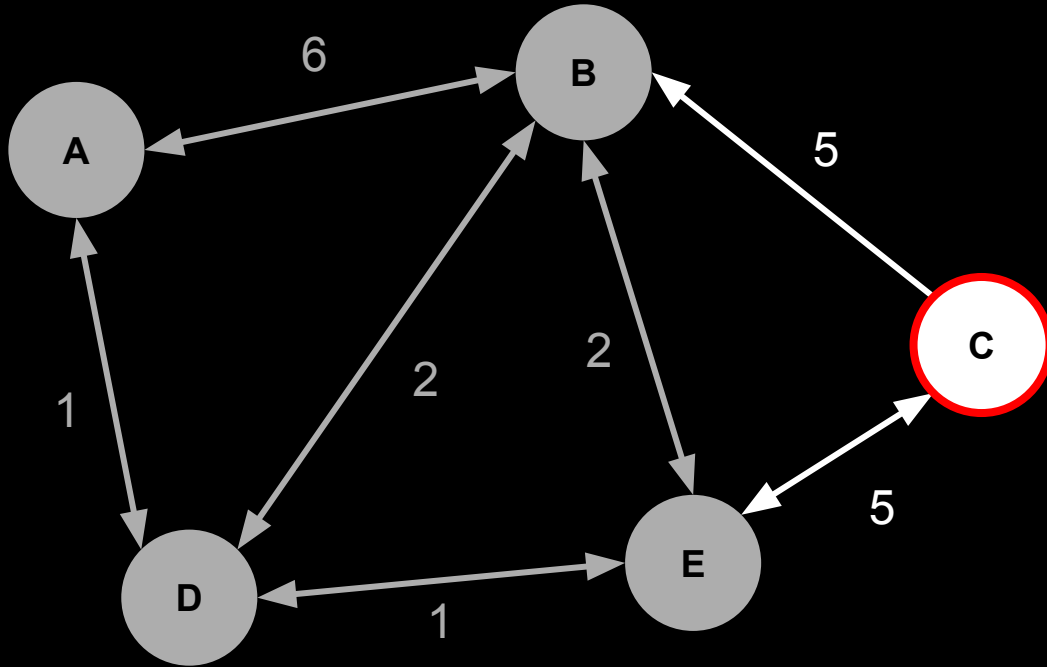
Distances: A = 0, B = 3, **C = 7**, D = 1, E = 2

Move to C as
new location



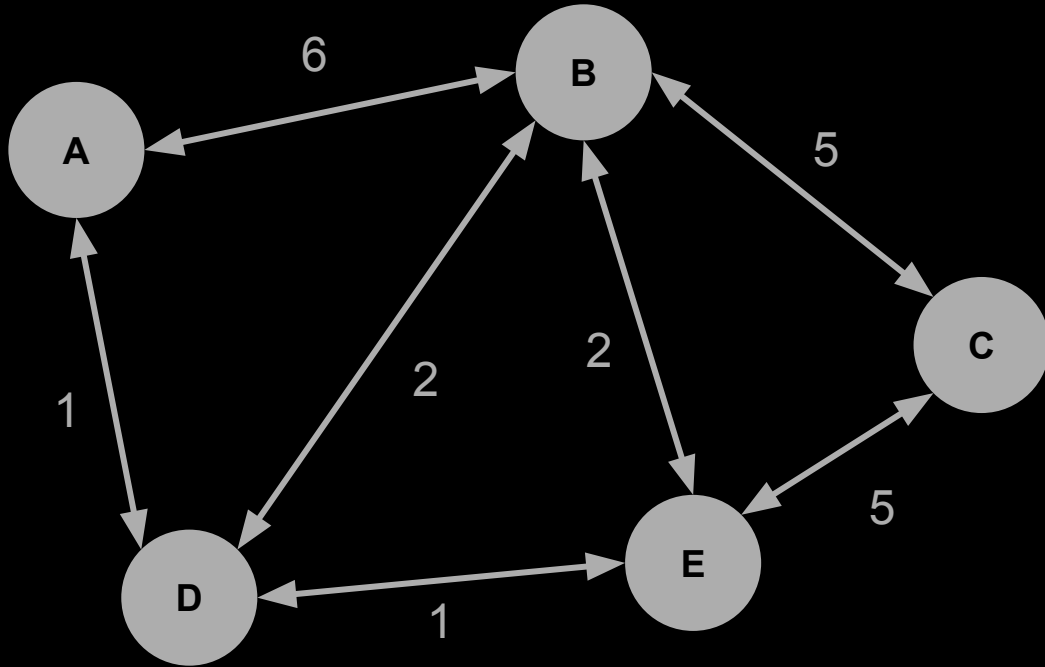
Distances: A = 0, B = 3, **C = 7**, D = 1, E = 2

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

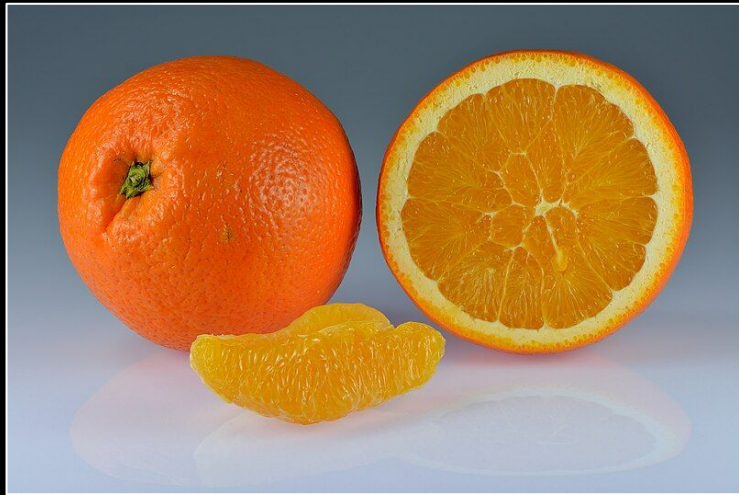


Image source: [Wikimedia](#)

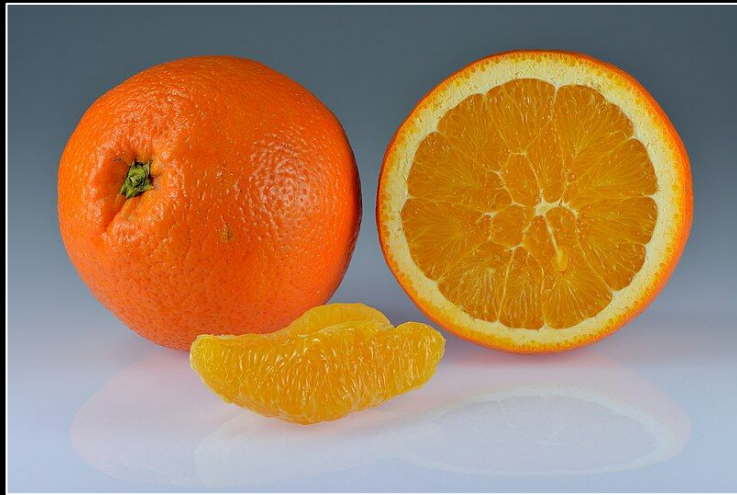


Image source: [Wikimedia](#)

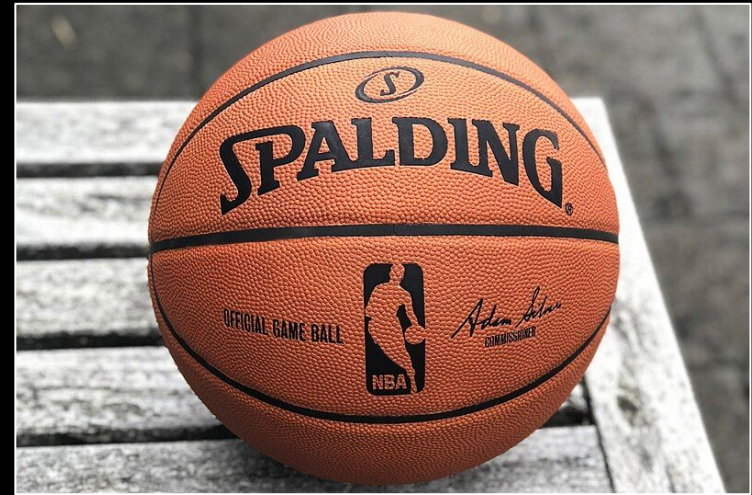


Image source: [Wikimedia](#)

what set of rules can solve this?

machine learning algorithms

