6. celok: Algoritmy

6. celok - Lekcia 1
Algoritmy riešia problémov

Warm Up

●○○

K zamysleniu:

V čom spočíva “rovnakosť” programu?

Môžu byť dva programy “rovnaké” napriek tomu že sú iné?

Activity

Prejdi sa po učebni a zisti odpovede na tieto otázky.

Problems

1. Nájdi nekoho kto má narodeniny pred tebou.
2. Nájdi nekoho kto má narodeniny po tebe.
3. Nájdi toho kto má narodeniny najbližšie pred tebou.
4. Nájdi toho kto má narodeniny najbližšie po tebe.
5. Nájdi toho kto má narodeniny najbližšie k tvojim.
6. Nájdi toho kto má rovnaký počet ľudí s narodeninami pred svojimi a po svojich.
7. Nájdi dvojicu ktorá má narodeniny najbližšie k sebe.
8. Zisti najkratšiu dobu medzi dvomi narodeninami.
9. Zisti najkratšiu dobu medzi štvormi narodeninami.
10. Zisti najdlhšiu dobu kedy nikto nemá narodeniny.

K zamysleniu:

Povedzte si ako ste pristupovali k riešeniu týchto problémov.

U ktorých problémov bolo treba spraviť podobné kroky?

Pracujte v dvojici a posuďte ktoré programy sú “rovnaké”.

Algoritmus 1

```
MOVE_FORWARD()
TURN_RIGHT()
MOVE_FORWARD()
TURN_RIGHT()
MOVE_FORWARD()
TURN_RIGHT()
MOVE_FORWARD()
TURN_RIGHT()
```

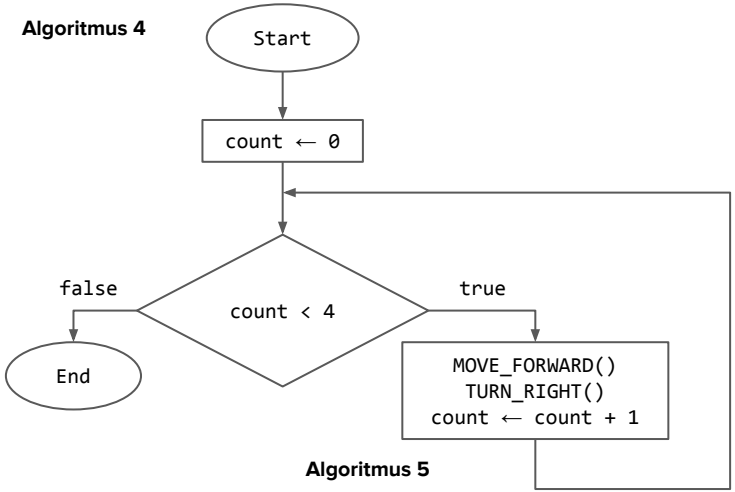
Algoritmus 2

```
REPEAT 2 TIMES
{
    MOVE_FORWARD()
    MOVE_FORWARD()
    TURN_RIGHT()
    MOVE_FORWARD()
    TURN_RIGHT()
}
```

Algoritmus 3

```
moves ← [“F” “R” “F” “R” “F” “R” “F” “R”]
FOR EACH move IN moves
{
    IF (move = “F”)
    {
        MOVE_FORWARD()
    }
    ELSE
    {
        TURN_RIGHT()
    }
}
```

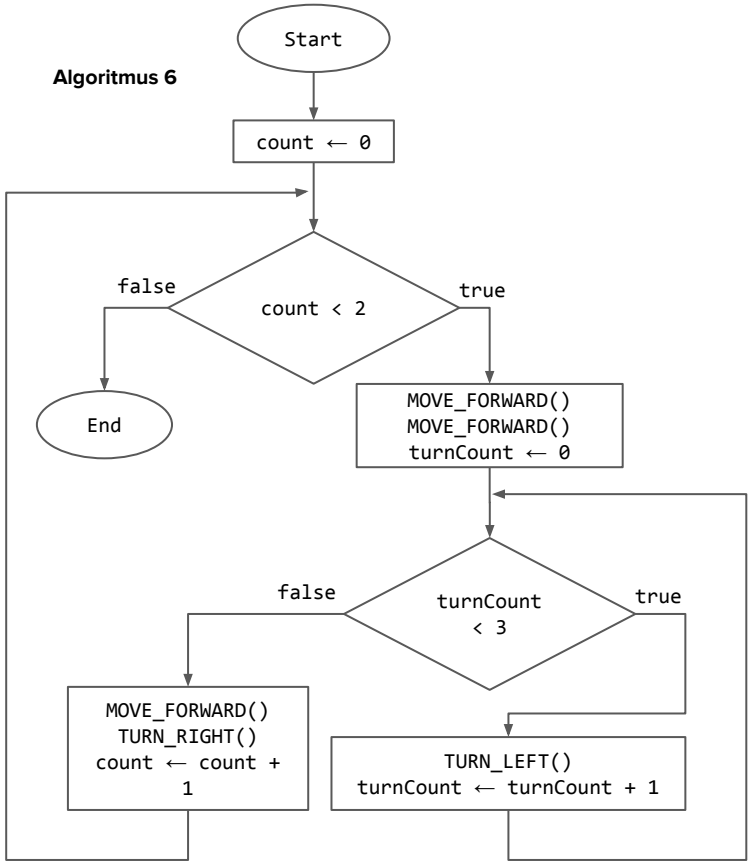
Algoritmus 4



Algoritmus 5

```
REPEAT 2 TIMES
{
    REPEAT 2 TIMES
    {
        MOVE_FORWARD()
    }
    REPEAT 3 TIMES
    {
        TURN_LEFT()
    }
    MOVE_FORWARD()
    REPEAT 3 TIMES
    {
        TURN_LEFT()
    }
}
```

Algoritmus 6



K zamysleniu:

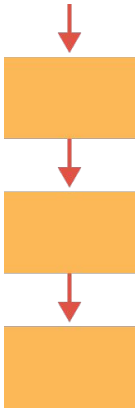
Diskutujte ktoré algoritmy sú “rovnaké”?

Ako to posúdiť?

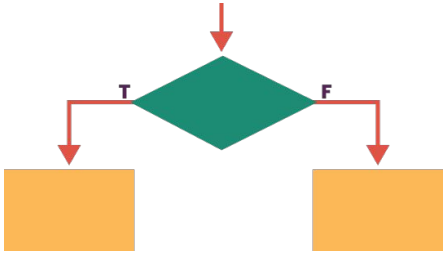
Wrap Up

Problém: všeobecný popis úlohy ktorú (ne)možno riešiť algoritmom
Algoritmus: konečná sada inštrukcií ktoré spĺňajú úlohu
K riešeniu jednoho problému väčšinou existuje veľa rôznych algoritmov a veľa spôsobov zápisu napr. Slovne pseudokódom vývojovým diagramom a programovacím jazykom.
Všetky algoritmy kombinujú tri základné prvky:

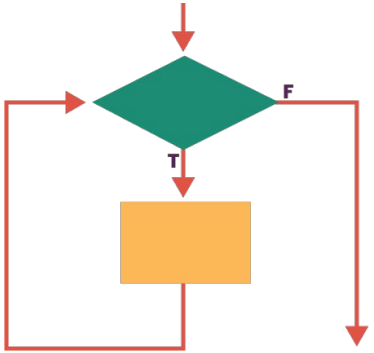
Postupnosť príkazov
Príkazy zoradené za sebou



Vetvenie
Podmienky určujúce ďalší postup



Cykly
Opakovanie určitých krokov v slúčke



K zamysleniu:

Zmenili nejak dnešné aktivity tvoj pohľad na algoritmy a problémy?

6. celok - Lekcia 2
Algoritmus Efficiency

Warm Up

●○○

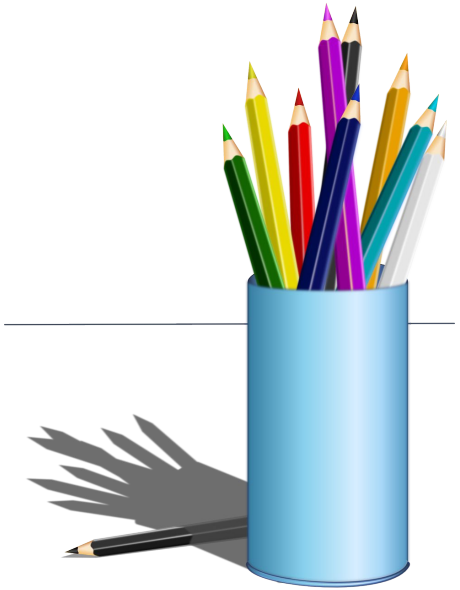
K zamysleniu:

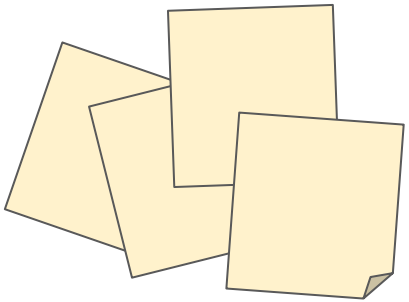
Stratili ste niekedy v ruksaku pero? Aké kroky treba urobiť aby ste našli pero?

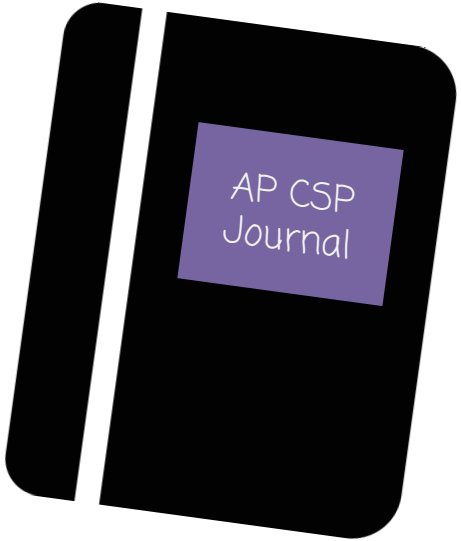
Activity

Efektívnosť algoritmu

Potrebujete:
lepiace papieriky
zápisník

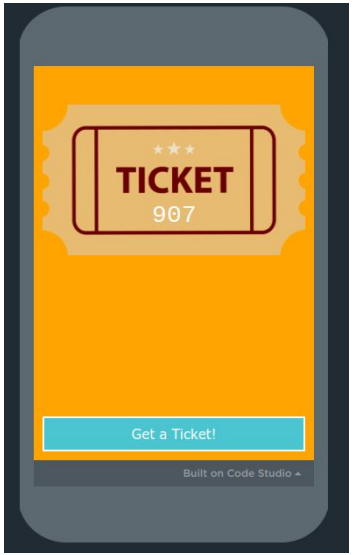


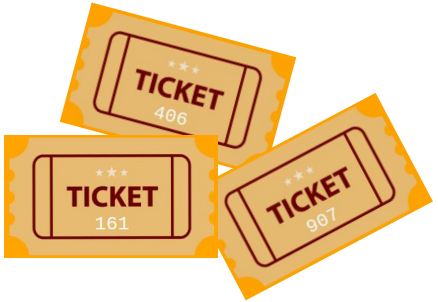




3 dobrovoľníci:

- Nájdi si 2. úroveň v Code Studio.
- Kliknutím si vygeneruj číslo do tomboly.
- Napíš si číslo na lepiaci papierik.
- Postav sa k tabuli.



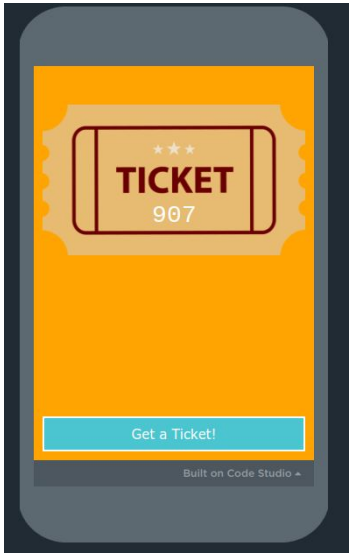


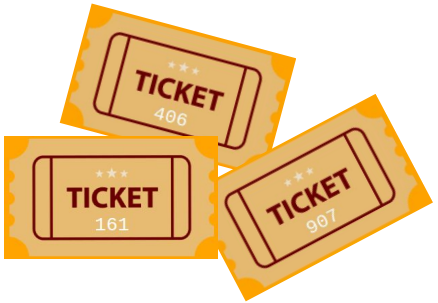
Problém: Zistiť či má niekto výherné číslo.

Zadanie: Zoznam lístkov a výherné číslo.

Skúsme zistiť či niekto vyhrál!

Postup: Overte výhry tak že postupne ukážete svoje číslo jeden za druhým.





K zamysleniu:

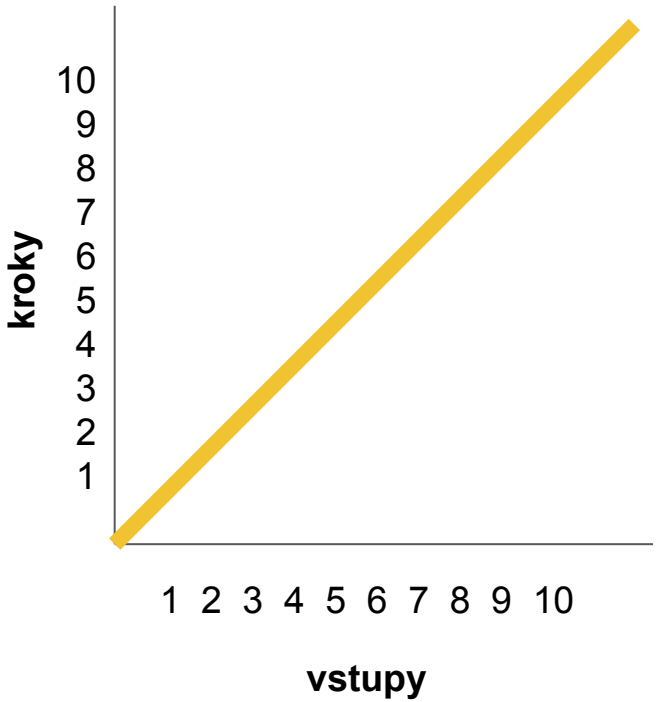
Koľko krokov sme potrebovali aby sme zistili či niekto vyhrál v tombole? Koľko krokov najviac môžeme spraviť aby sme splnili toto zadanie?

K zamysleniu:

Čo keby bolo dobrovoľníkov šesť? Celá trieda? Celá škola?

Vidíme tu nejaký opakujúci sa vzor?

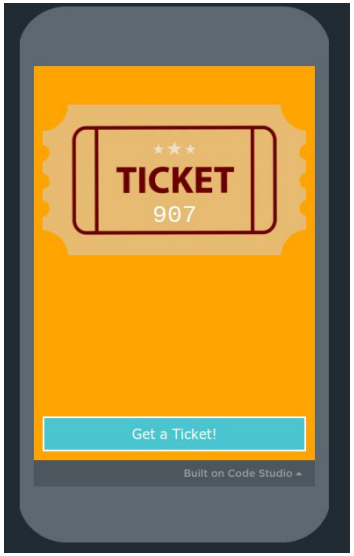
	vstupy	kroky
Prípad	3	3
Prípad	6	6
Prípad	10	10
Prípad	100	100



V dvojiciach...

Postup:

- Použite generátor na vygenerovanie 7 lístkov. Opíšte si čísla na papieriky.
- Zoraďte papieriky od najmenšího po najväčší.
- Jedno číslo opíšte na osobitný papierik. To bude hľadané číslo.



705

117 232 245 410 705 716 833

Výzva: Vytvorte algoritmus ktorý zistí či je dané číslo v danej sade čísiel.

- Hľadanie môže začať u ktoréhokoľvek papierika.
- Papieriky môžete preskakovať nemusíte dodržať poradie.
- Ďalší papierik k porovnaní určujte z toho ktorý ste práve spracovali.
- Pokúste úlohu splniť čo najmenším počtom krokov - ale nezabúdajte že hľadané číslo môže byť hocikde - ktoré miesto je najhoršie možné? Aký najväčší počet krokov by potreboval váš algoritmus v tom najhoršiom prípade?

Porovnajte navzájom: Spojte sa s inou skupinou. Ukážte si navzájom celý svoj algoritmus. Určite ktorá skupina má "rýchlejší" alebo ktorá urobí menej krokov.

Postup: Skúste teraz algoritmus nižšie a porovnajte.

1. Nájdi v sade číslo uprostred. Porovnaj s daným číslom. Ak je menšie ako dané číslo odstráň všetky papieriky naľavo (vrátane prostredného čísla). Ak je prostredné číslo väčšie odstráň všetky papieriky napravo (vrátane prostredného čísla).
2. Nájdi v skrátenej sade číslo. Opakuj 1. krok porovnávania.
3. Nájdi v skrátenej sade číslo. Opakuj 1. krok porovnávania.

Našli sme hľadané číslo!

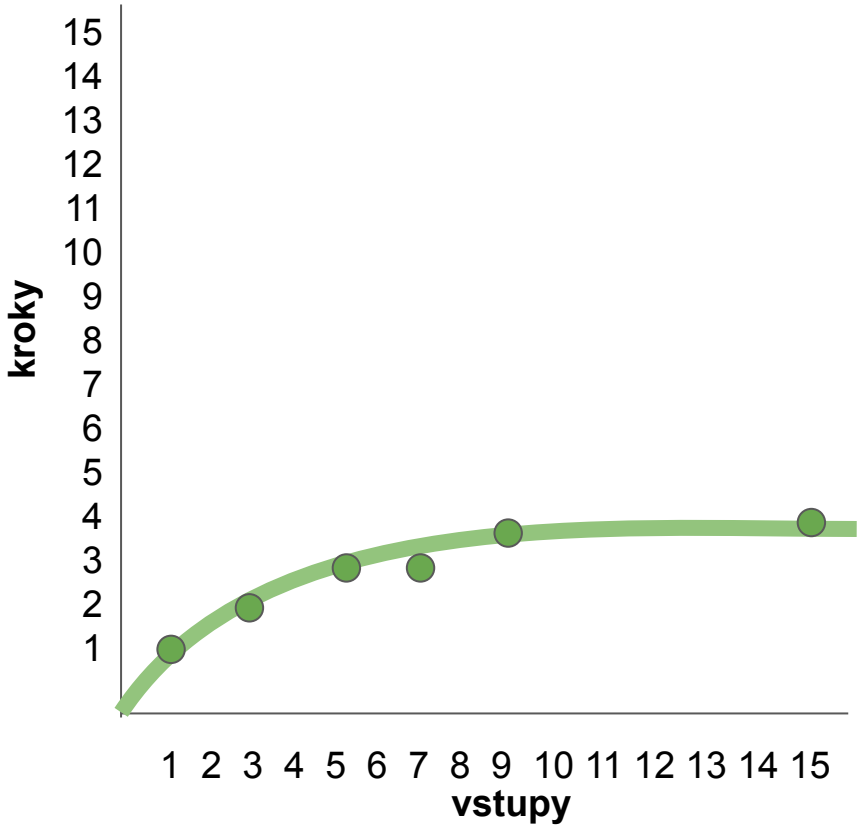
Binárne vyhľadávanie:

Našli sme
naše číslo!
705 v sade je.

Krok 1 Krok 3 Krok 2

Postup: Vyskúšajme binárne vyhľadávanie pre rôzne prípady. Urobili sme to za vás a výsledky sú v tabuľke!

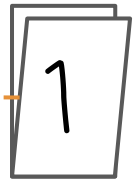
	vstupy	kroky
Prípad	1	1
Prípad	3	2
Prípad	5	3
Prípad	7	3
Prípad	9	4
Prípad	15	4



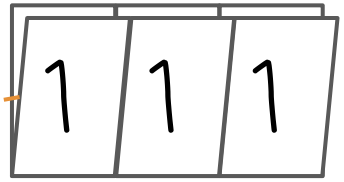
Dá sa na to pozrieť inak. Vidíte nejaký opakujúci sa vzor?

	inputs	steps
Prípad	1	1
Prípad	3	2
Prípad	5	3
Prípad	7	3
Prípad	9	4
Prípad	15	4

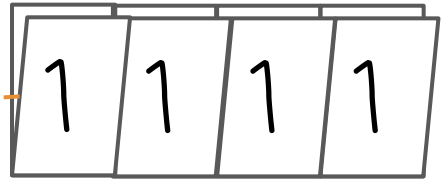
Koľko bitov treba k reprezentácii 1 binárne?

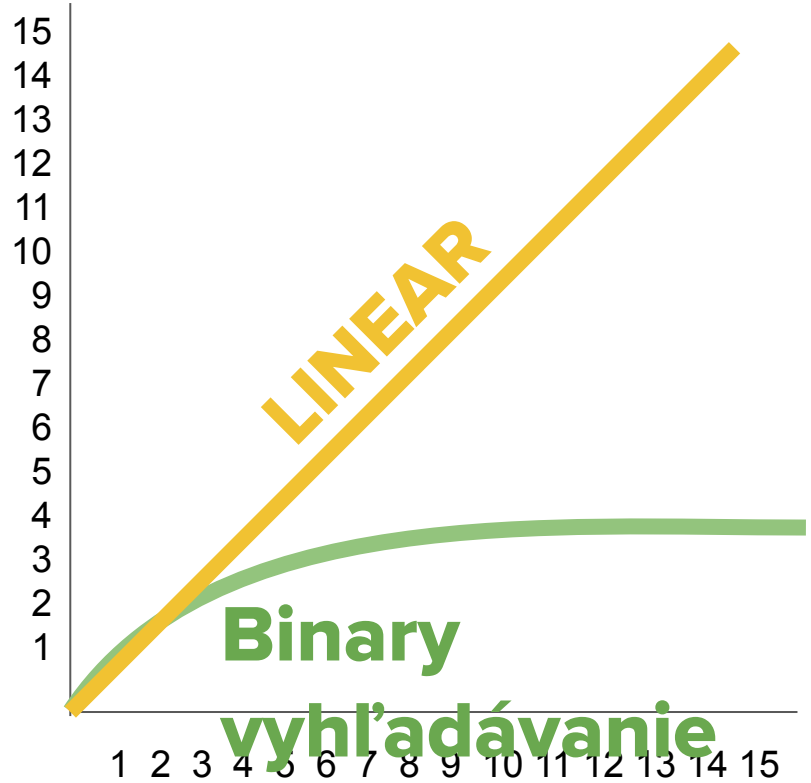


A čo pre 7?



A čo pre 15?





Here's our two search Algoritmuss we've explored. The first is **linear**. As we add more inputs the number of steps grows at the same rate.

The second represents what happens with Binary vyhľadávanie. Notice how it grows at a much slower rate! Binary vyhľadávanie is faster than Linear search BUT the data must be sorted.

Wrap Up

K zamysleniu:

Ak by sme mali jeden vstup ktorý algoritmus by potreboval najmenej krokov k riešeniu?

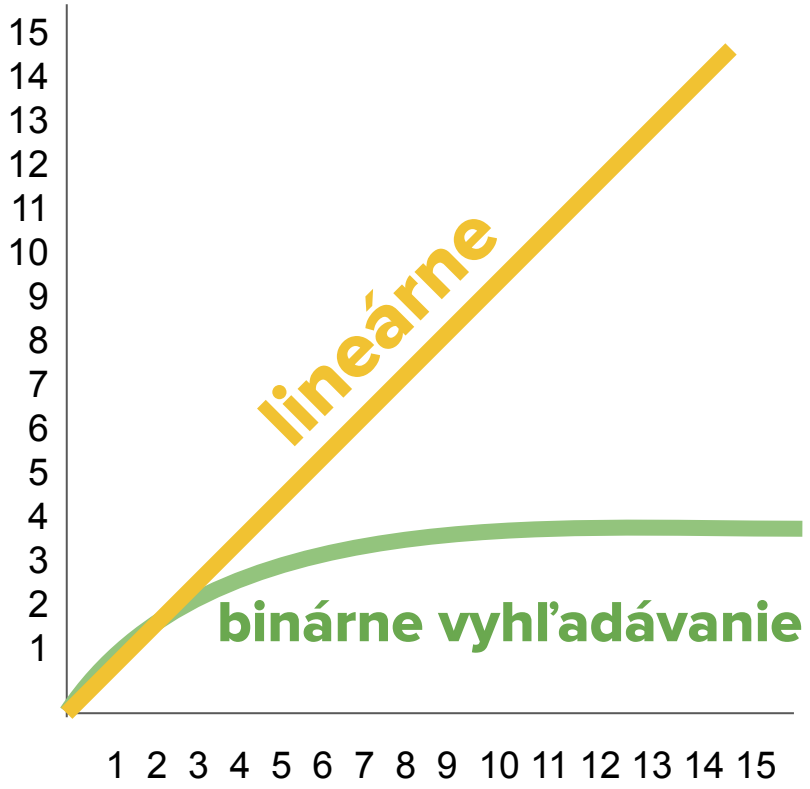
A pre päť vstupov?

A čo pre sto vstupov?

Efektívnosť: merítko podľa počtu krokov k dokončeniu algoritmu

Lineárne vyhľadávanie: vyhľadávací algoritmus ktorý prechádza každú položku kým nenájde hľadaný prvok alebo nedosiahne konca.

Binárne vyhľadávanie: vyhľadávací algoritmus ktorý začne uprostred usporiadanej sady čísel a odstráni polovicu dát; proces opakuje kým nenájde hľadaný prvok alebo nedosiahne konca.



6. celok - Lekcia 3
Neprimeraný čas

Warm Up

●○○

K zamysleniu:

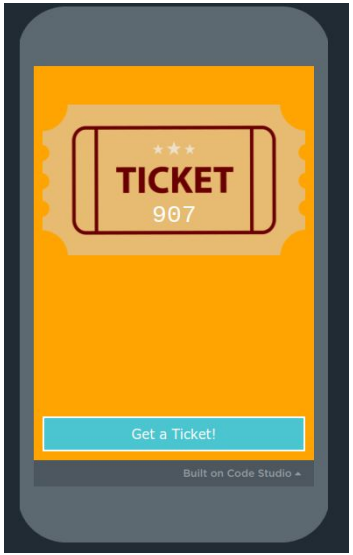
Čo sa mieni tým že jeden algoritmus je “efektívnejší” ako druhý?

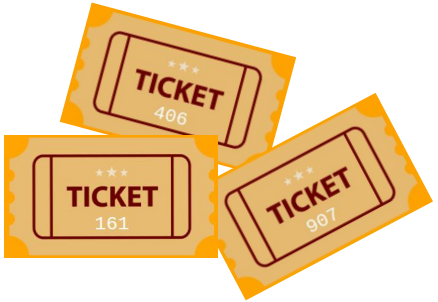
Activity

Tombola dvojíc
Vyhrávajú dva lístky ktorých súčet je víťazné číslo.

Víťazné číslo je 1000.

Postup
Vygeneruj si lístok.
V tichosti sa prejdi po učebni.
Zisti či tvoje číslo s nekým dokopy vyhrálo!





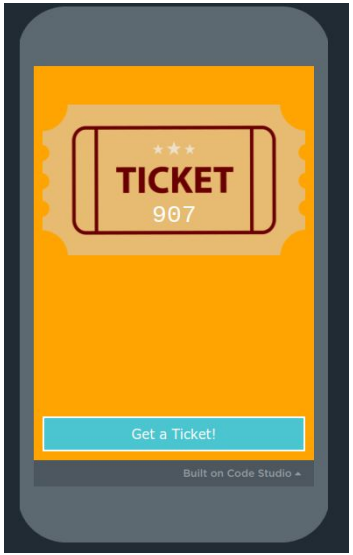
Tombola skupín

Vyhráva hocijaká skupina (od jedného po všetky lístky) ktorých súčet je víťazné číslo.

Víťazné číslo je 2500.

Postup

Vygeneruj si lístok.
Prejdi sa po triede - môžeš hovoriť.
Zisti či tvoje číslo s nekým dokopy vyhrálo!



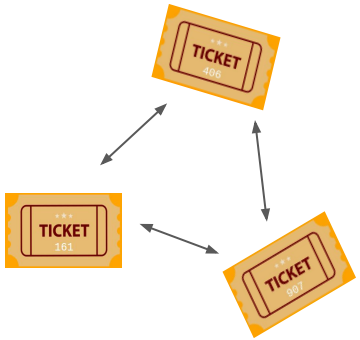
K zamysleniu

Ktorú tombolu bolo ťažšie vyhodnotiť? Prečo?

Zapíšme si algoritmus ktorý vyhodnotí každý pár či každú skupinu v tombole (dvojíc či skupin).

Zistime koľkokrát bude treba vyhodnotiť výhru!

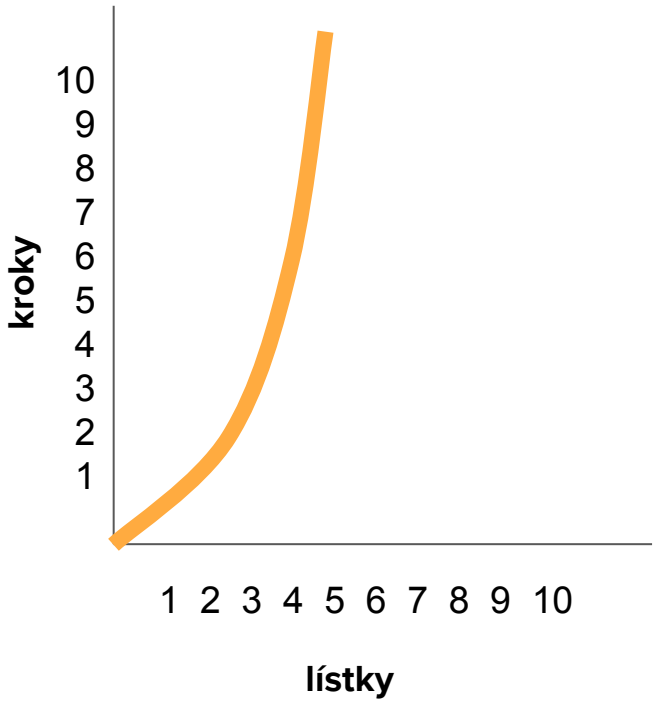
V dvojici vyplňte tabuľku v zošite.



Share your responses with another group!

lístky	kroky
2	1
3	3
4	6
5	10
8	28

Tombola dvojíc



Vzorec pre tento vzťah

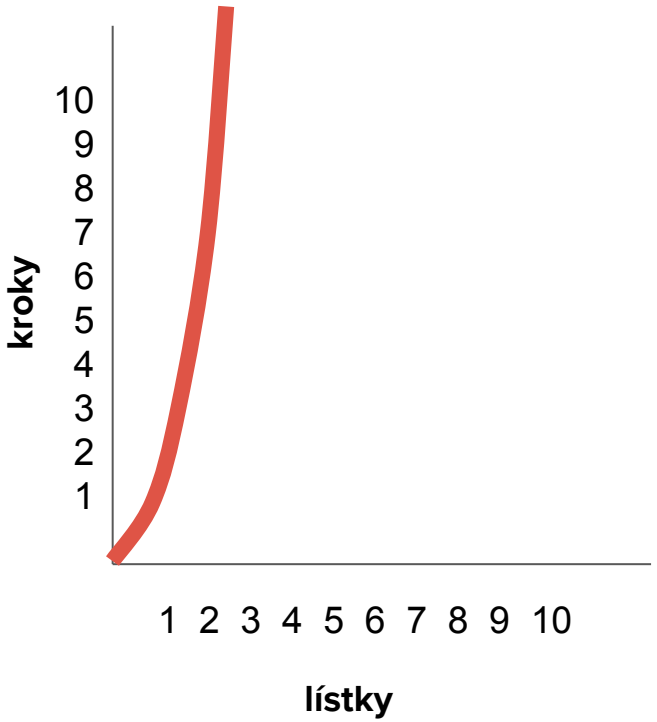
$(n^2 - n)/2$

Vzorec nemusíme poznať ale z toho že je v ňom n^2 by sme mali poznať aký bude graf.

Algoritmus s efektivnosťou n^2 n^3 n^4 ... sa volá **polynomiálny.**

lístky	kroky
2	3
3	7
4	15
5	31
8	255

Tombola skupín



Vzorec pre tento vzťah

$2^n - 1$

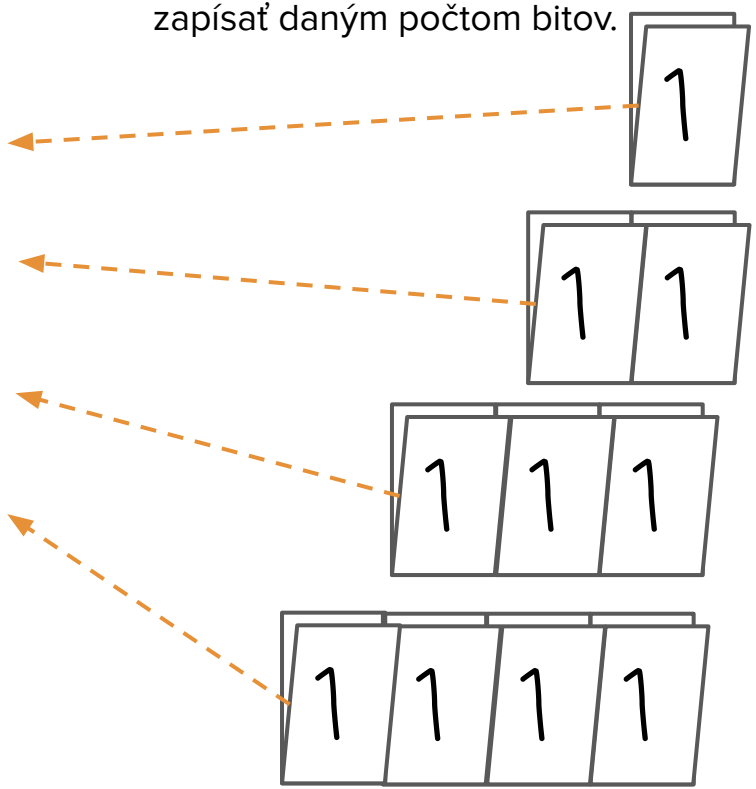
Vzorec nemusíme poznať ale z toho že je v ňom 2^n by sme mali vedieť že rýchlo rastie.

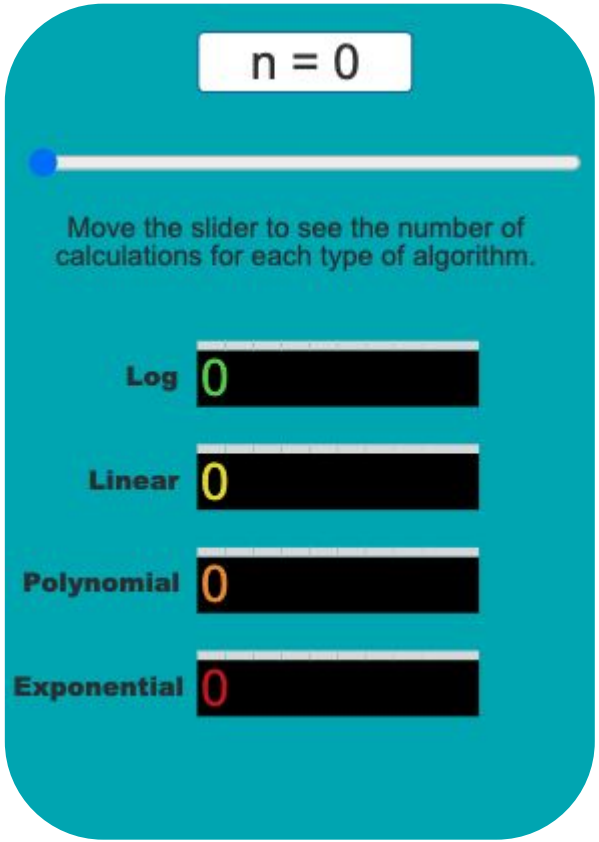
Algoritmus s efektívnosťou 2^n 3^n 4^n ... sa volá **exponenciálny.**

Dá sa na to pozrieť inak.

lístky	kroky
1	1
2	3
3	7
4	15
5	31
8	255

Počeť porovnaní odpovedá číslu ktoré možno zapísať daným počtom bitov.





Primeraný či neprimeraný čas?

- Pustite si aplikáciu tlačítkom Run.
- Posuvníkom zisti ako narastá počet krokov.

← Tombola podľa poradí

← Tombola

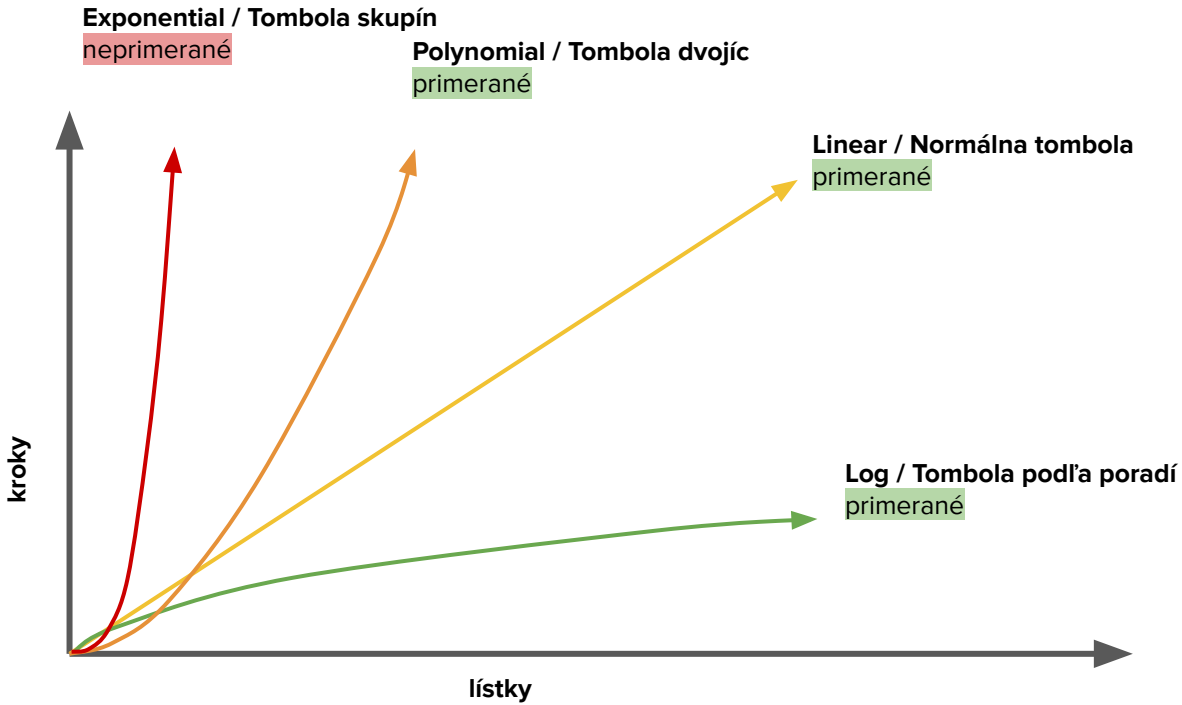
← Tombola dvojíc

← Tombola skupín

Diskutujte v dvojiciach:

a. Ktorý z algoritmov potrebuje primeraný čas?
b. Ktorý neprimerane veľa času?

Polynómy i exponenciála majú zakrivený rast.
Prečo považujeme len exponenciálny rast za neprimeraný?



Lístky	**Podľa poradí** logaritmické	**Normálna tombola** lineárne	**Tombola dvojíc** polynomiálne	**Tombola skupín** exponenciálne
10	4 porovnaní	10 porovnaní	100 porovnaní	1 024 porovnaní
20	5 porovnaní	20 porovnaní	400 porovnaní	1 048 576 porovnaní
100	7 porovnaní	100 porovnaní	10 000 porovnaní	$1.26 * 10^{30}$ porovnaní
1000	10 porovnaní	1 000 porovnaní	1 000 000 porovnaní	$1.07 * 10^{301}$ porovnaní
10 000	14 porovnaní	10 000 porovnaní	10^{9} porovnaní	$2,00 * 10^{3010}$ porovnaní
100 000	17 porovnaní	100 000 porovnaní	10^{12} porovnaní	$9,99 * 10^{30102}$ porovnaní

Viac porovnaní než atomov vo vesmíre

Polynomiálne je zlé, ale exponenciálne narastá neprimerane rýchlo.

Wrap Up

primeraný čas: Algoritmy s polynomiálnou či nižšou efektívnosťou (konštantnou lineárnou kvadratickou atď.) potrebujú primeraný čas.

neprimeraný čas: Algoritmy s exponenciálnou či faktoriálnou efektívnosťou sú príkladom algoritmov ktoré potrebujú neprimerane veľa času.

K zamysleniu:

Vaša škola plánuje usporiadať tombolu skupín počas školskej akademie.

Napíšte si stručne aké odporúčanie byste dali školskej rade.

6. celok - Lekcia 4
The Limits of Algoritmuss

Warm Up

●○○

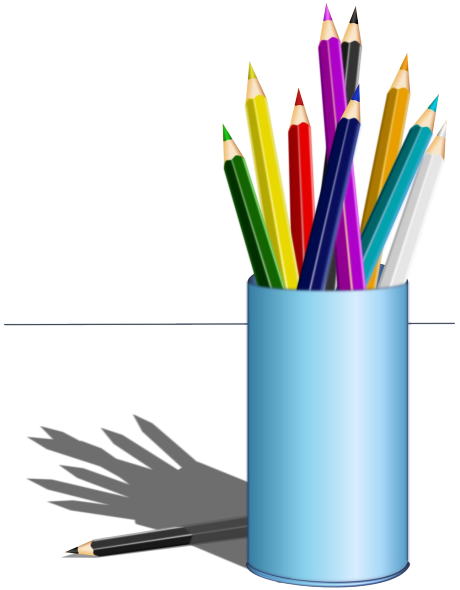
K zamysleniu:

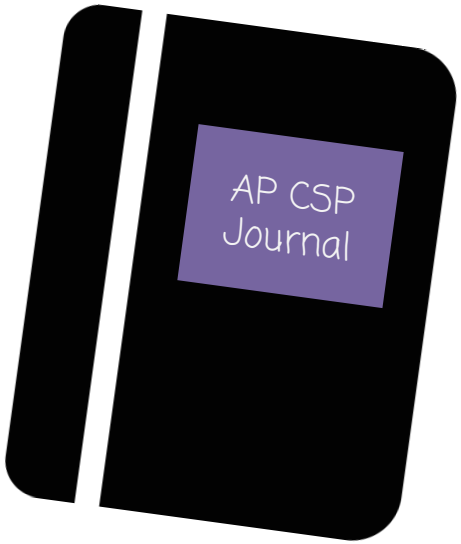
Aký je rozdiel v algoritmoch ktorým treba primeraný a neprimeraný čas?

Activity

Traveling Salesman

You should have:
Your journal
Pen/pencil

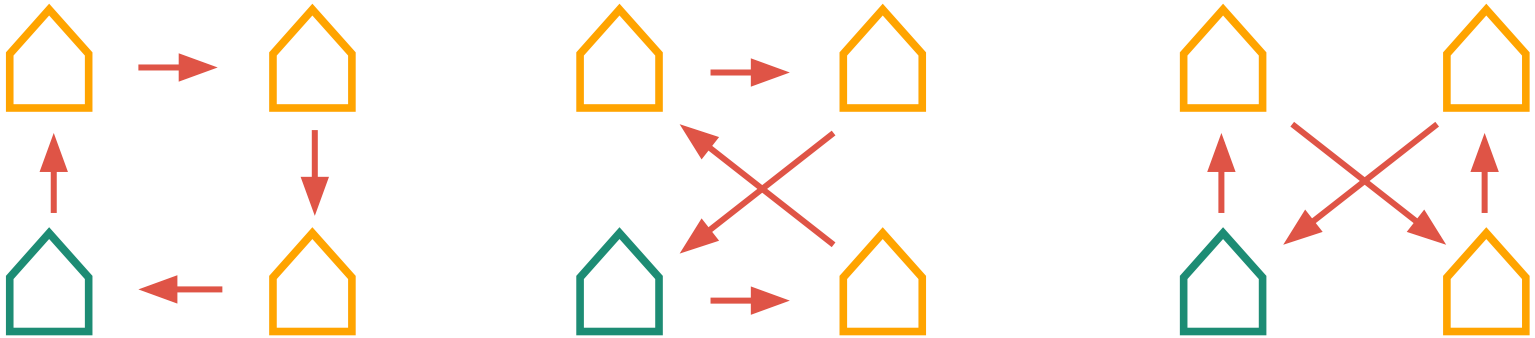




K zamysleniu: How many different paths can you find to visit all of your friends' houses?

Rules:

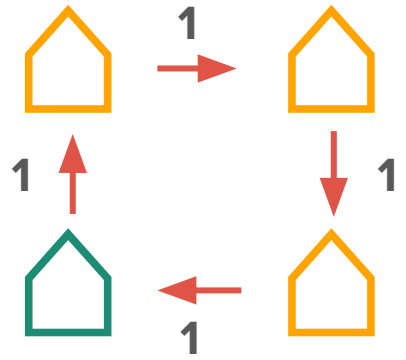
- You must start and end at your own house.
- You can only visit each house once.



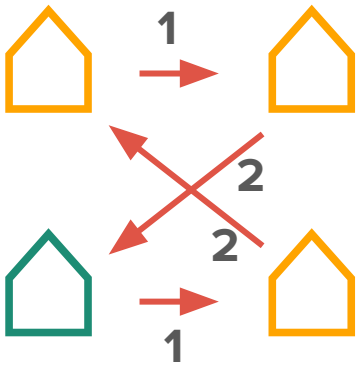
Here are a few different paths you might take.

K zamysleniu:

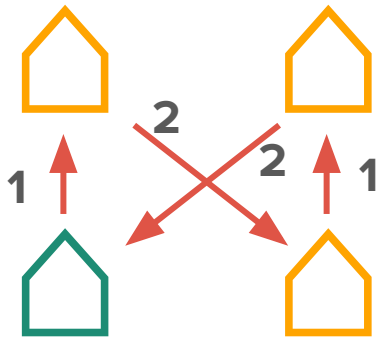
What do you need to know to determine the best path?



Total: 4



Total: 6



Total: 6

Distance!

K zamysleniu: What if we had a lot more places to visit? How would we determine the best path?

This is known as the **Traveling Salesman Problem**.

For every new place to visit the number of options for possible paths increases factorialy.

Number of houses to visit	Number of steps to check for the "best" path
1	1
2	2
3	6
4	24
5	120
6	720
7	5 040
8	40 320
9	362 880
10	3 628 800

Factorial fun: n!

Here's how n! works:

Multiply all whole numbers from the given number down to the number 1.

For example:
Instance: 4 houses to visit

4 x 3 x 2 x 1 = 24

Instance: 7 houses to visit

7 x 6 x 5 x 4 x 3 x 2 x 1 = 5 040

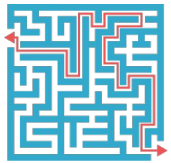
Number of houses to visit	Number of steps to check for the "best" path
1	1
2	2
3	6
4	24
5	120
6	720
7	5 040
8	40 320
9	362 880
10	3 628 800

That's a lot of possible paths to check for only 10 houses!

Problems

Any task that may (or may not) be solved with an Algoritmus.
Sorting a list is a problem. Sorting the list (2 3 1 7) is an instance of that problem.

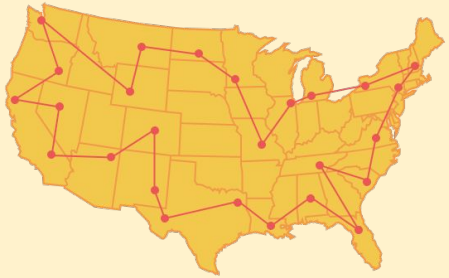
Decision Problems

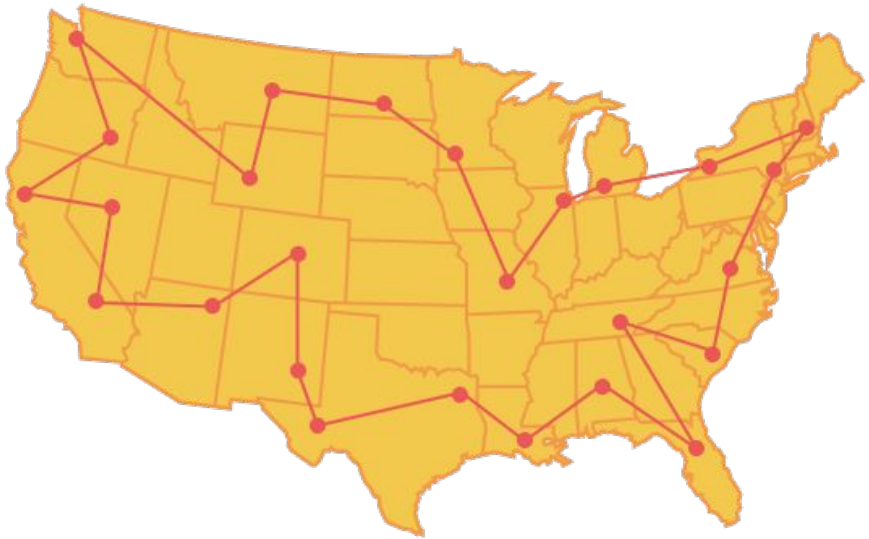


"Is there a path?"

Optimization Problems

"What's the shortest path"?





The **Traveling Salesman Problem** can be solved with an Algoritmus which porovnaní each possible option.

BUT it would take massive amounts of computing power to compare every single option especially as the number of homes to visit (otherwise known as *nodes*) increases.

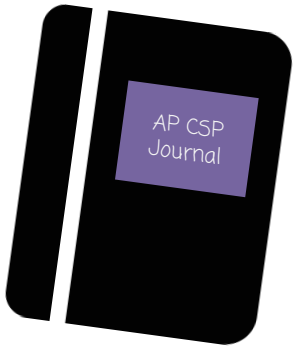
Therefore it would take an **unreasonable** amount of time for the solution to be calculated for most instances of the problem.

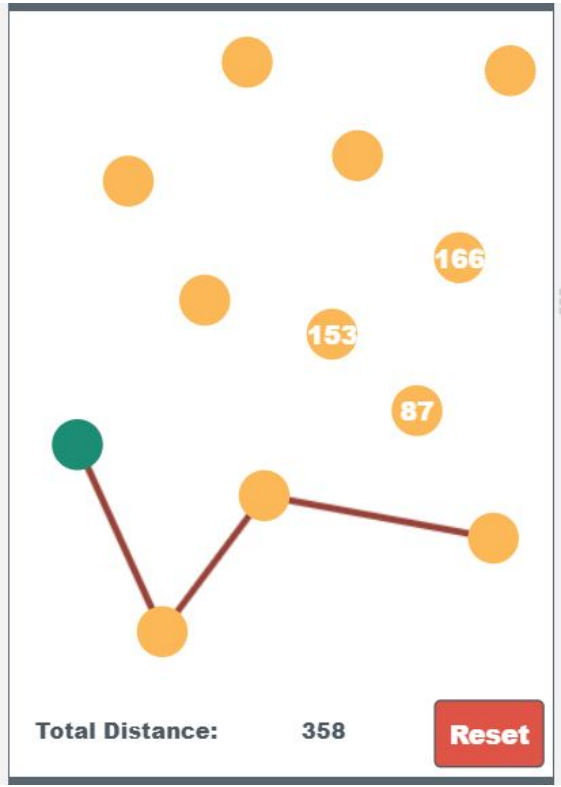
Welcome to **heuristics!**

- Provide a "good enough" solution to a problem when an actual solution is impractical or impossible

Postup:

- Navigate to Level 1 on Code Studio
- Try to find the "best" path to visit all nodes.
- Write down a plan or **heuristic** for choosing a good path.
    - Note: your **heuristic** may not always find the best path but it should be close enough

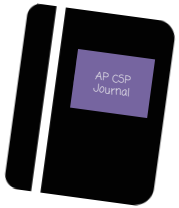




Postup:

- Navigate to Level 2 on Code Studio
- Test your heuristic on three different levels.
- Write down the distance for the path your heuristic finds.
- Try to find the best version not using the heuristic (brute force). Can you find a better path? Is your heuristic on average pretty good? Should you update your heuristic?

Distance (Heuristic)	Distance (Brute Force)
1225	1215



K zamysleniu:

- How did you create your heuristic?
- Did you change your heuristic after testing it out?

Share Out:

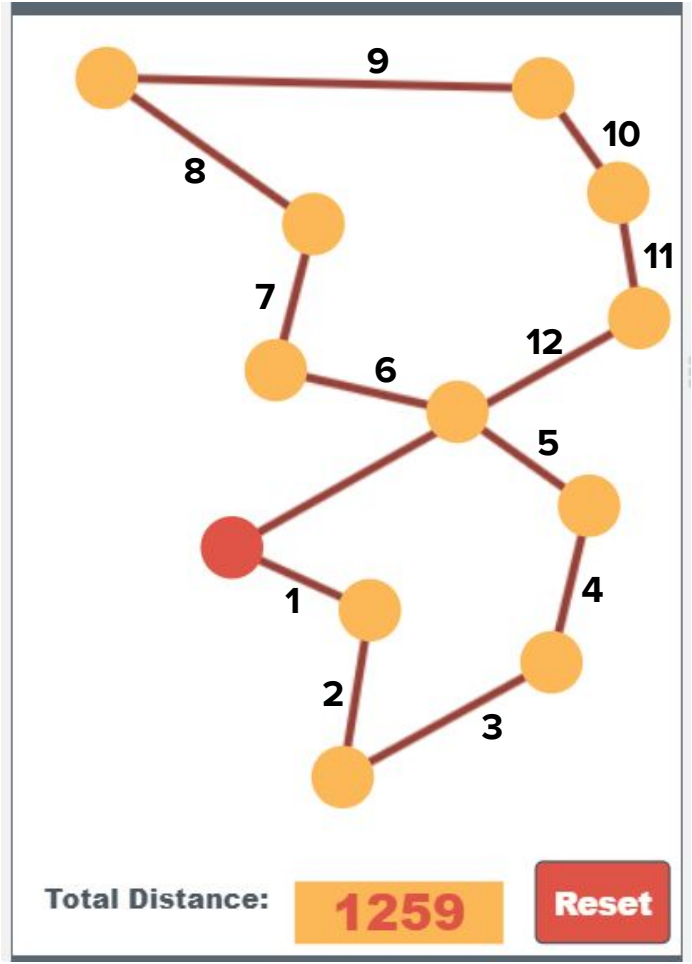
Explain your heuristic.

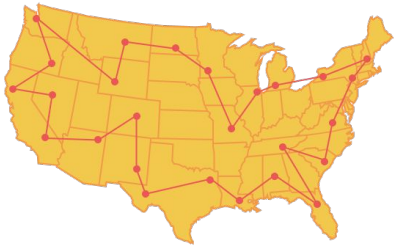
As a class which do we think is best?

Sample Heuristic:

At each node travel to the next closest node

Is this "good enough"?





Takeaways:

The Traveling Salesman Problem is an **optimization** problem. We are attempting to find the best path.

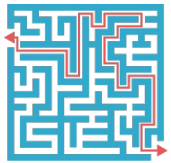
It is also **unreasonable** because there is not an Algoritmus that can solve the problem in a reasonable amount of time.

We need to use a **heuristic** to come up with a solution that is "good enough" for most instances of the problem.

Problems

Any task that may (or may not) be solved with an Algoritmus.
Sorting a list is a problem. Sorting the list (2 3 1 7) is an instance of that problem.

Decision Problems



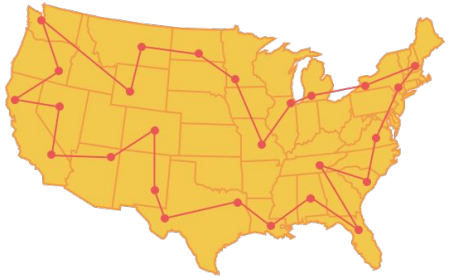
"Is there a path?"

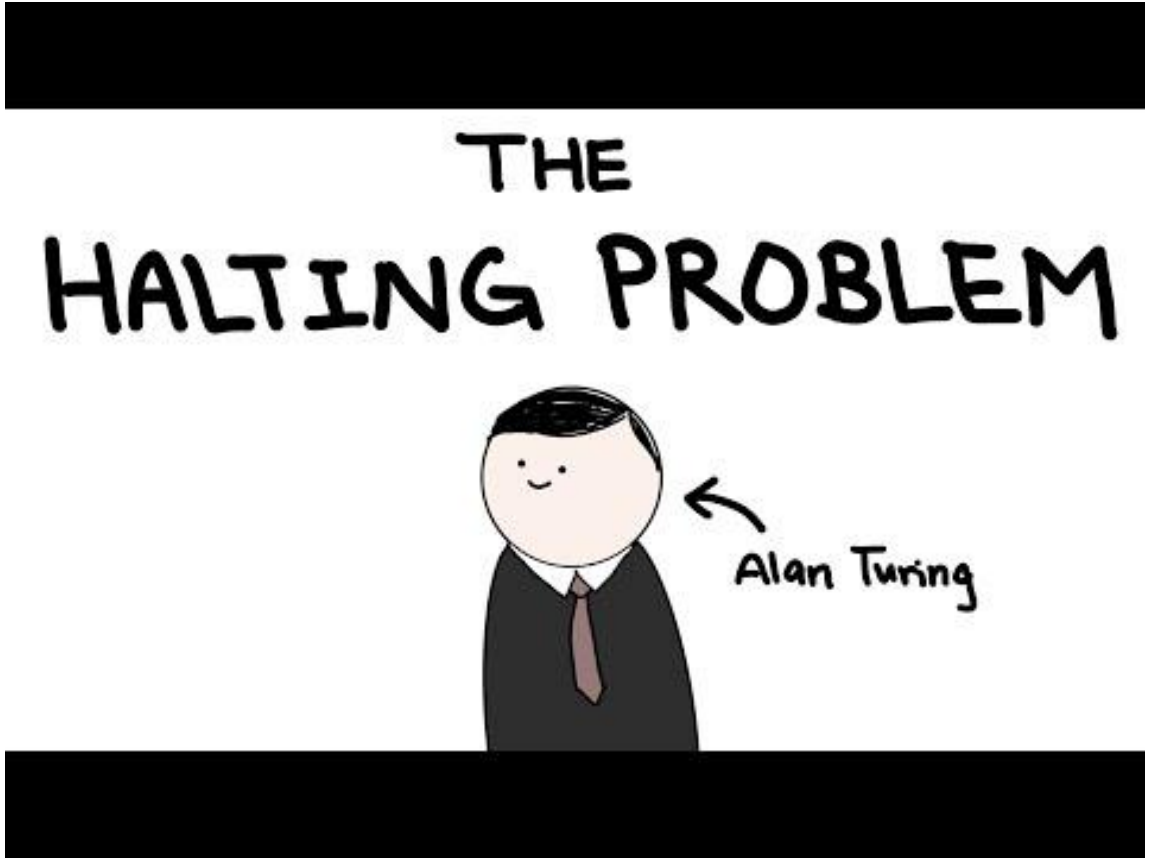
Undecidable Problems

"Will this code work?"

Optimization Problems

"What's the shortest path"?




THE
HALTING PROBLEM
Alan Turing

Takeaways:

There are some problems we've proven that no computer will ever be able to solve. The Halting Problem is a very famous example and in general we call these problems **undecidable**.

Wrap Up

K zamysleniu:

Why is a heuristic acceptable when it doesn't always produce the "best" result?

Heuristic: provides a "good enough" solution to a problem when an actual solution is impractical or impossible

Undecidable Problem: a problem for which no Algoritmus can be constructed that is always capable of providing a correct yes-or-no answer

6. celok - Lekcia 5
Distributed Algoritmuss

Warm Up

●○○

K zamysleniu:

Brainstorm a task that you can complete faster if you get other people to help.

What’s the most number of people you’d want to help you and why?

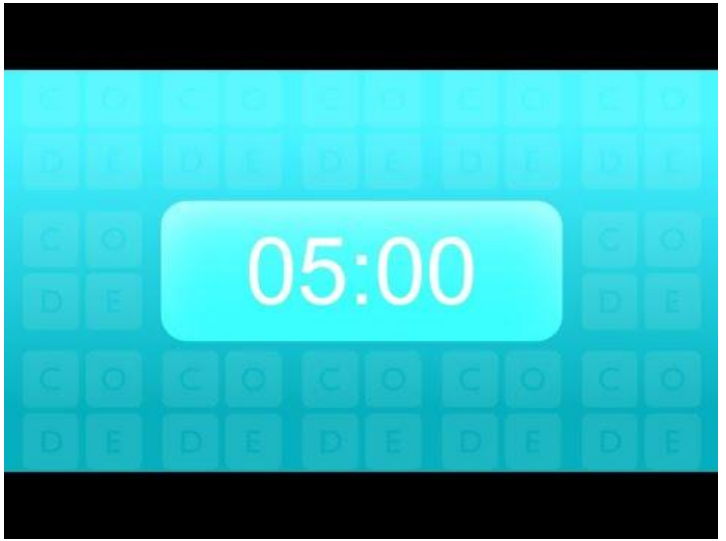
Activity

Parallel Algoritmuss and Speedup

Groups: Get into groups of 3 or 4
Each group should have a deck of cards

Challenge One - One Person Sort

Shuffle the cards
Put them in a neat stack face down
As quickly as you can get the cards sorted so all the red cards are at the bottom and all the black cards are at the top.
Time stops when you have the cards sorted and back in a neat stack.

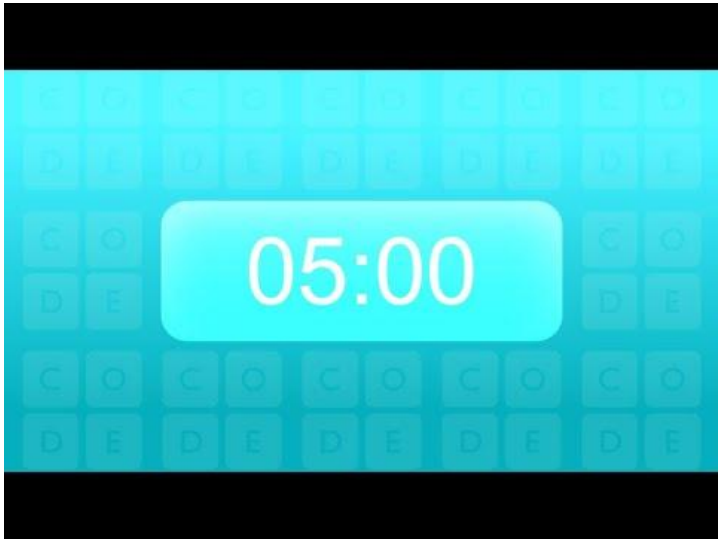


Record the best time in your group

Challenge Two - Two Person Sort

Shuffle the cards
Put them in a neat stack face down
As quickly as you can get the cards sorted so all the red cards are at the bottom and all the black cards are at the top.
Time stops when you have the cards sorted and back in a neat stack.

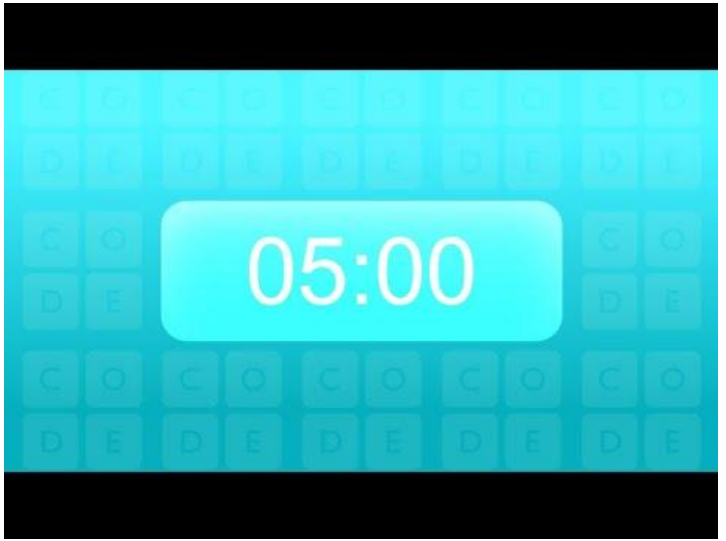
This time two people can sort the cards.
Record the best time in your group



Challenge Three - Full Group Sort

Shuffle the cards
Put them in a neat stack face down
As quickly as you can get the cards sorted so all the red cards are at the bottom and all the black cards are at the top.
Time stops when you have the cards sorted and back in a neat stack.

This time your entire group (three or four people) can sort the cards.
Record the best time in your group

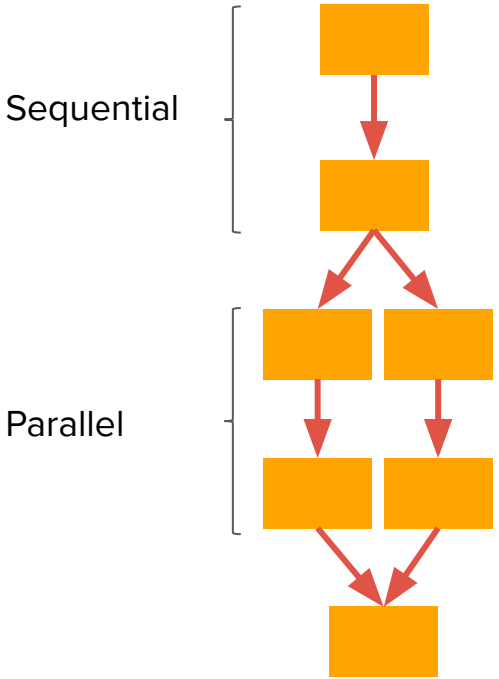


Sequential

Steps are performed in order one at a time.

Parallel

Some steps are performed at the same time.



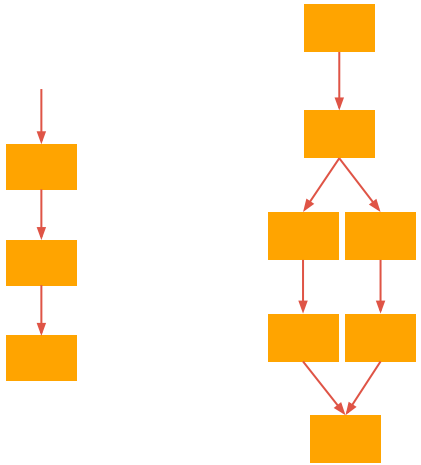
K zamysleniu

What portions of your Algoritmuss for Challenges 2 and 3 were parallel?

What makes things complicated or slows you down during parallel portions of your Algoritmus?

Speedup

Sequential time divided by parallel time



60 seconds

40 seconds

60 seconds / 40 seconds = 1.5
The speedup of this parallel solution is 1.5

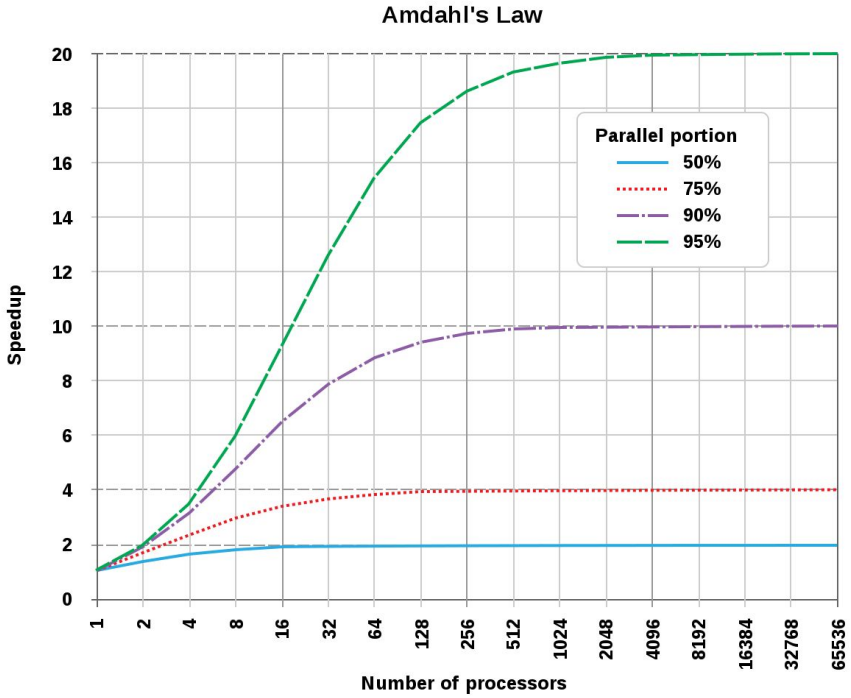
K zamysleniu: What was your group's speedup in Challenge 2?

What about in Challenge 3?

Are you surprised?

It’s not just you! Speed-up is never equal to the number of processors.

Some portions of your Algoritmus can’t be made parallel. Each additional processor helps a little less. Eventually the speedup reaches a limit.



K zamysleniu

As you watch this video write down

- Why is the type of computing presented “distributed”?
- Why is distributed computing used to solve the problem?

K zamysleniu

As you watch this video write down

- Why is the type of computing presented “distributed”?
- Why is distributed computing used to solve the problem?

Wrap Up

Sequential Computing: programs run in order one command at a time.

Parallel Computing: programs are broken into small pieces some of which are run simultaneously

Distributed Computing: programs are run by multiple devices

Speedup: the time used to complete a task sequentially divided by the time to complete a task in parallel

K zamysleniu:

Based on today's activities what are the pros and cons of parallel and distributed computing?

6. celok - Lekcia 6
Assessment Day

Activity

Unit Assessment