Algorithm
Project 2

Graph Algorithms

01

Algorithm Applied

Breadth First Search

02

Shortest Distance

Top k-nearest hospitals, random graph generator

03

Time Complexity

“How long?”

04

Empirical Analysis

Comparing em’

01

Algorithm

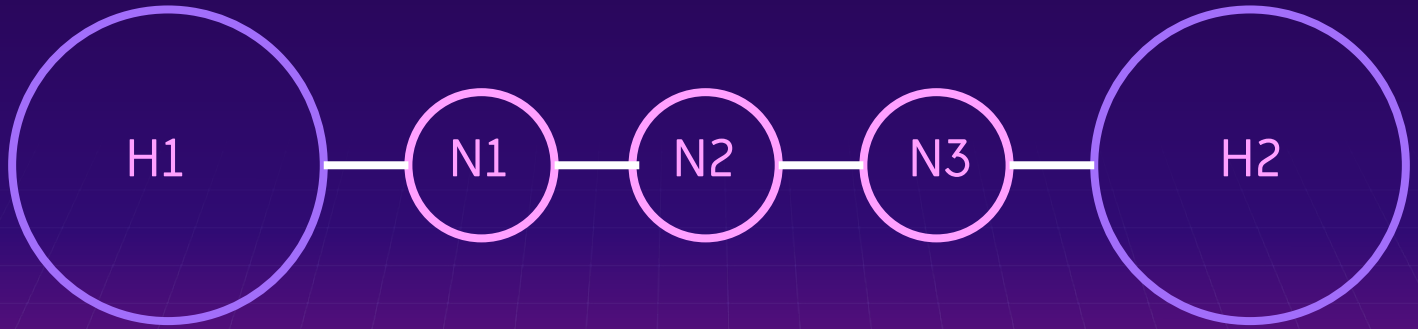
Breadth First Search
(BFS)

Breadth First Search

Summary of Breadth First Search (BFS)

- starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'[1])
- explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level

Distance & Shortest Path (. solve())



Shortest Path :

['H1']
['H2']
['N1']+['H1'] =['N1', 'H1']
['N3']+['H2'] =['N3', 'H2']
['N2']+['N1', 'H1']= ['N2','N1', 'H1']

Distance & Shortest Path (.solve())

```
def solve(self):                                # finds shortest path to the nearest hospital for every node

    Q=Queue()
    for H in self.hospitals:                    # shortest path of a hospital is simply to itself
        self.solutionPath[H][H]=[H]
        self.solutionDistance[H][H]=len(self.solutionPath[H][H])-1
        Q.put(H)
                                                # apply BFS to each hospital simultaneously!
    while not Q.empty():
        currentNode=Q.get()
        H=list(self.solutionPath[currentNode].keys())[-1]     # H is the nearest hospital of the current Node
        for neighbourNode in self.adj_list[currentNode]:      # for the neighbour nodes, update their shortest path using the
            if self.solutionPath[neighbourNode]=={}:        # e.g. ['567']+['678','345','0'], so ['567','678','345','0'] is t
                self.solutionPath[neighbourNode][H]=[neighbourNode]+self.solutionPath[currentNode][H]
                self.solutionDistance[neighbourNode][H]=len(self.solutionPath[neighbourNode][H])-1
                Q.put(neighbourNode)
```

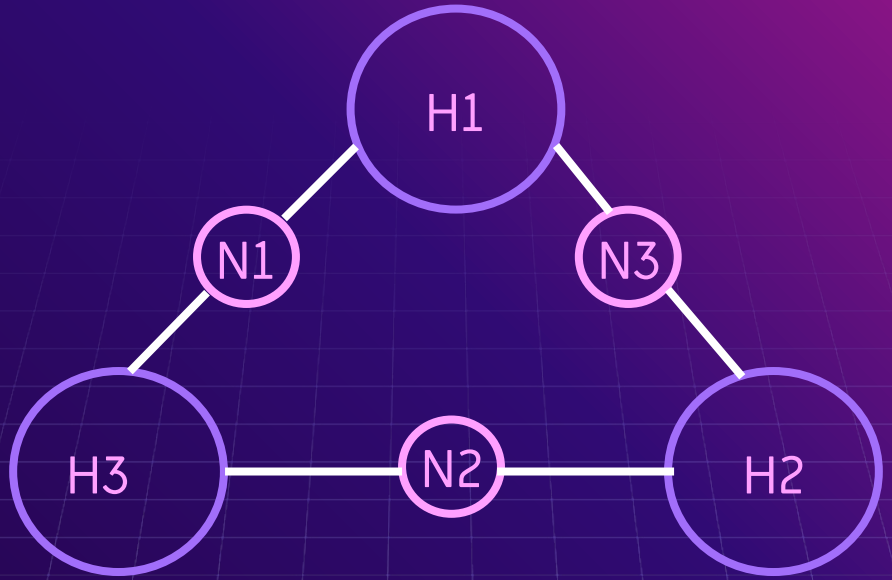
Time Complexity (.solve())

$$C_1|E| + C_0|V| = O(|V|+|E|)$$

C1 = time cost of comparisons done in the nested for loop

C0 = time spent on each node

Top-k Nearest Hospitals (.topK())



Top-k Nearest Hospitals (.topK())

C_2

C_3

```
topK(self,K):

for H in self.hospitals:                    # apply BFS each hospital consecutively

    if len(self.topKDistances[H])>=K:        # this ensures that hospitals will have only top k distances including
        del self.topKDistances[H][max(self.topKDistances[H],key=lambda x: self.topKDistances[H][x])]
    self.topKDistances[H][H]=0               # distance of hospital to itself is 0

    self.explored=defaultdict(lambda: False)
    self.explored[H]=True                                # consider hospital as explored already

    Q=Queue()                                            # put hospital in Q and begin BFS
    Q.put(H)

    while not Q.empty():
        currentNode=Q.get()

        for neighbourNode in self.adj_list[currentNode]:
            if self.explored[neighbourNode]==False:                    # if the node is not yet explored

                distance=self.topKDistances[currentNode][H]+1          # its shortest distance to H is 1+currentNod

                if len(self.topKDistances[neighbourNode])>=K:          # if there is already K solutions
                    furthestHospital=max(self.topKDistances[neighbourNode],key=lambda x: self.topKDistances[neighbo
                    if distance<self.topKDistances[neighbourNode][furthestHospital]:   # delete the longest existin
                        del self.topKDistances[neighbourNode][furthestHospital]
                    else:                                                   # if there is already k distances and exist
                        self.explored[neighbourNode]=True
                        continue

                self.topKDistances[neighbourNode][H]=distance  # all else, update topKDistances
                self.explored[neighbourNode]=True
                Q.put(neighbourNode)
```

Time Complexity (.topK())

$$C2|V|k + (C2+C3)(h-k)|v'|+ C4k|E| + C4(h-k)|e'| \leq (C2+C3)(h)|V| +C4h|E|$$

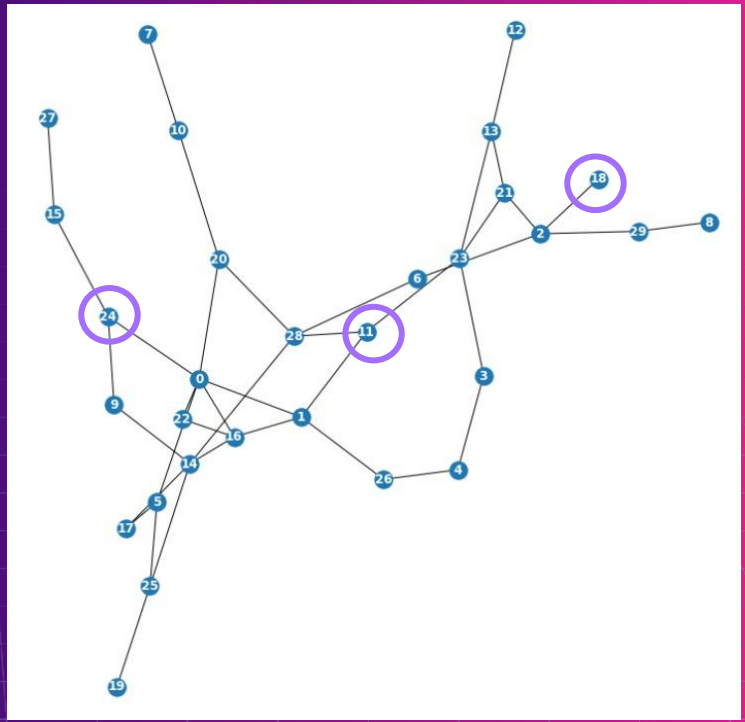
$$= O(h(|V|+|E|))$$

- C2 = total time cost of assigning values to topKDistances, labelling True for explored nodes, assigning values to existingDistances and putting and getting nodes from Q
- C3 = total time cost of deleting existing solutions from topKDistances
- C4 = time cost of comparing the Boolean value of a node with False to check whether it has been explored

Random Graph Generator

- Small scale graph generated from Random Graph Generator
- No. of nodes: 30
- No. of edges: 45
- Declaration of hospitals:

Node 11, 18, 24




```
In [136]: # declare hospitals
          randomHospitals2=list(map(int,random.sample(range(0,30),3)))

In [137]: randomHospitals2

Out[137]: [18, 11, 24]
```

Random Graph Generator

- Convert graph to adjacency list

```
In [135]: adj_list2

Out[135]: defaultdict(list,
                      {0: [1, 5, 16, 20, 22, 24],
                       1: [0, 11, 16, 26],
                       2: [6, 18, 21, 29],
                       3: [4, 23],
                       4: [3, 26],
                       5: [0, 17, 25],
                       6: [2, 28],
                       7: [10],
                       8: [29],
                       9: [14, 24],
                       10: [7, 20],
                       11: [1, 23, 28],
                       12: [13],
                       13: [12, 21, 23],
                       14: [9, 16, 17, 25, 28],
                       15: [24, 27],
                       16: [0, 1, 14, 22],
                       17: [5, 14],
                       18: [2],
                       19: [25],
                       20: [0, 10, 28],
                       21: [2, 13, 23],
                       22: [0, 16],
                       23: [3, 11, 13, 21],
                       24: [0, 9, 15],
                       25: [5, 14, 19],
                       26: [1, 4],
                       27: [15],
                       28: [6, 11, 14, 20],
                       29: [2, 8]})
```

Random Graph Generator

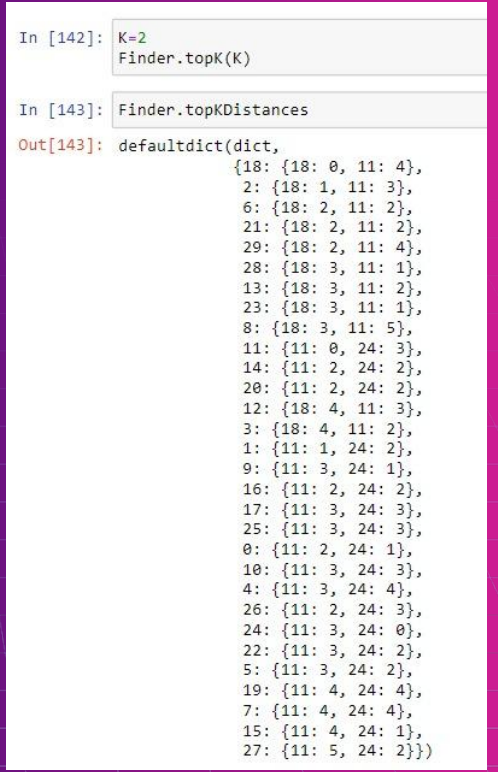
- Applying .solve()
- Nearest hospital to each node is shown
- The shortest path from each node to its nearest hospital
- The shortest distance from each node to its nearest hospital

```
In [140]: Finder.solutionPath
Out[140]: defaultdict(dict,
                      {18: {18: [18]},
                       11: {11: [11]},
                       24: {24: [24]},
                       2: {18: [2, 18]},
                       1: {11: [1, 11]},
                       23: {11: [23, 11]},
                       28: {11: [28, 11]},
                       0: {24: [0, 24]},
                       9: {24: [9, 24]},
                       15: {24: [15, 24]},
                       6: {18: [6, 2, 18]},
                       21: {18: [21, 2, 18]},
                       29: {18: [29, 2, 18]},
                       16: {11: [16, 1, 11]},
                       26: {11: [26, 1, 11]},
                       3: {11: [3, 23, 11]},
                       13: {11: [13, 23, 11]},
                       14: {11: [14, 28, 11]},
                       20: {11: [20, 28, 11]},
                       5: {24: [5, 0, 24]},
                       22: {24: [22, 0, 24]},
                       27: {24: [27, 15, 24]},
                       8: {18: [8, 29, 2, 18]},
                       4: {11: [4, 26, 1, 11]},
                       12: {11: [12, 13, 23, 11]},
                       17: {11: [17, 14, 28, 11]},
                       25: {11: [25, 14, 28, 11]},
                       10: {11: [10, 20, 28, 11]},
                       19: {11: [19, 25, 14, 28, 11]},
                       7: {11: [7, 10, 20, 28, 11]}})
```

```
In [141]: Finder.solutionDistance
Out[141]: defaultdict(dict,
                      {18: {18: 0},
                       11: {11: 0},
                       24: {24: 0},
                       2: {18: 1},
                       1: {11: 1},
                       23: {11: 1},
                       28: {11: 1},
                       0: {24: 1},
                       9: {24: 1},
                       15: {24: 1},
                       6: {18: 2},
                       21: {18: 2},
                       29: {18: 2},
                       16: {11: 2},
                       26: {11: 2},
                       3: {11: 2},
                       13: {11: 2},
                       14: {11: 2},
                       20: {11: 2},
                       5: {24: 2},
                       22: {24: 2},
                       27: {24: 2},
                       8: {18: 3},
                       4: {11: 3},
                       12: {11: 3},
                       17: {11: 3},
                       25: {11: 3},
                       10: {11: 3},
                       19: {11: 4},
                       7: {11: 4}})
```

Random Graph Generator

- Applying .topK().
- K = 2
- The top 2 nearest hospitals to each node
- The distance from each of the hospitals to each of the nodes are printed

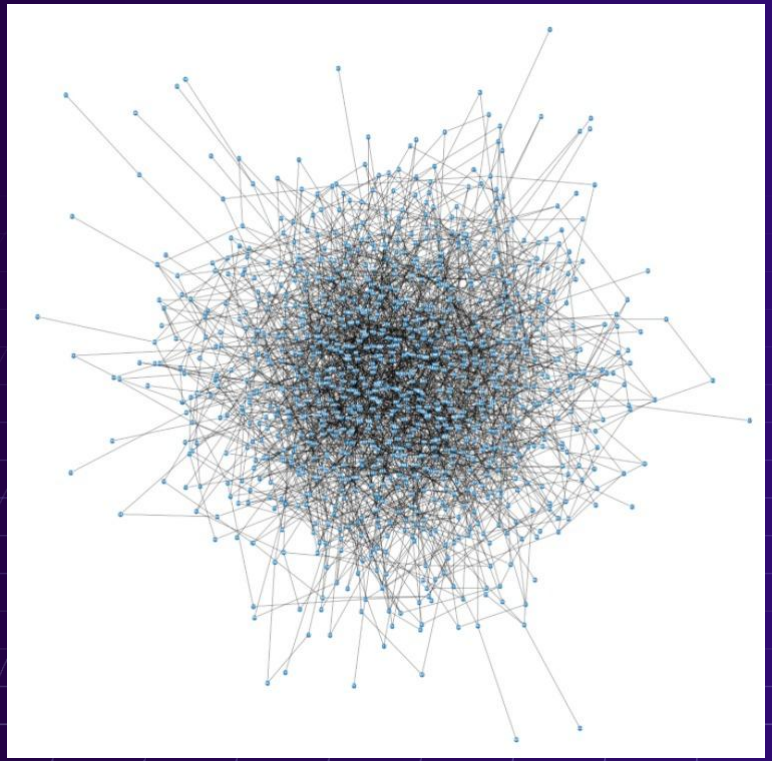


```
In [142]: K=2
          Finder.topK(K)

In [143]: Finder.topKDistances

Out[143]: defaultdict(dict,
                      {18: {18: 0, 11: 4},
                       2: {18: 1, 11: 3},
                       6: {18: 2, 11: 2},
                       21: {18: 2, 11: 2},
                       29: {18: 2, 11: 4},
                       28: {18: 3, 11: 1},
                       13: {18: 3, 11: 2},
                       23: {18: 3, 11: 1},
                       8: {18: 3, 11: 5},
                       11: {11: 0, 24: 3},
                       14: {11: 2, 24: 2},
                       20: {11: 2, 24: 2},
                       12: {18: 4, 11: 3},
                       3: {18: 4, 11: 2},
                       1: {11: 1, 24: 2},
                       9: {11: 3, 24: 1},
                       16: {11: 2, 24: 2},
                       17: {11: 3, 24: 3},
                       25: {11: 3, 24: 3},
                       0: {11: 2, 24: 1},
                       10: {11: 3, 24: 3},
                       4: {11: 3, 24: 4},
                       26: {11: 2, 24: 3},
                       24: {11: 3, 24: 0},
                       22: {11: 3, 24: 2},
                       5: {11: 3, 24: 2},
                       19: {11: 4, 24: 4},
                       7: {11: 4, 24: 4},
                       15: {11: 4, 24: 1},
                       27: {11: 5, 24: 2}})
```

Random Graph Generator

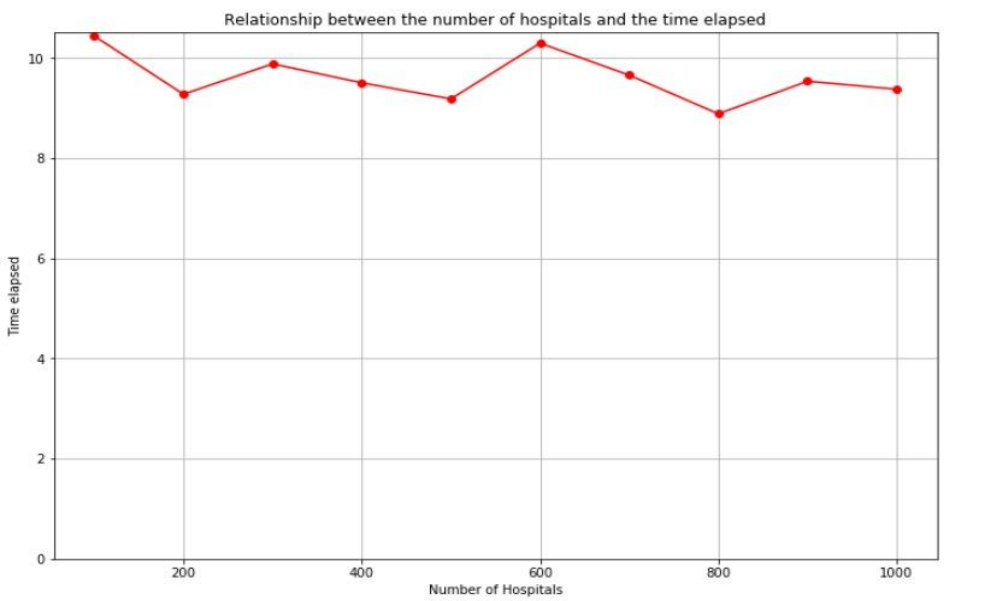


In [18]:
```
n_nodes = 1000
m_edges = 2500
generator = GraphGenerator(max_nodes=n_nodes,max_edges=m_edges)
G = generator.generate_unweighted_graph()
graphRandom = G
```

Empirical Analysis

For .solve(), with parameter h:

	number of hospitals	time elapsed (seconds)
0	100	10.438270
1	200	9.276233
2	300	9.881493
3	400	9.503964
4	500	9.182226
5	600	10.295055
6	700	9.656386
7	800	8.884372
8	900	9.535182
9	1000	9.375122

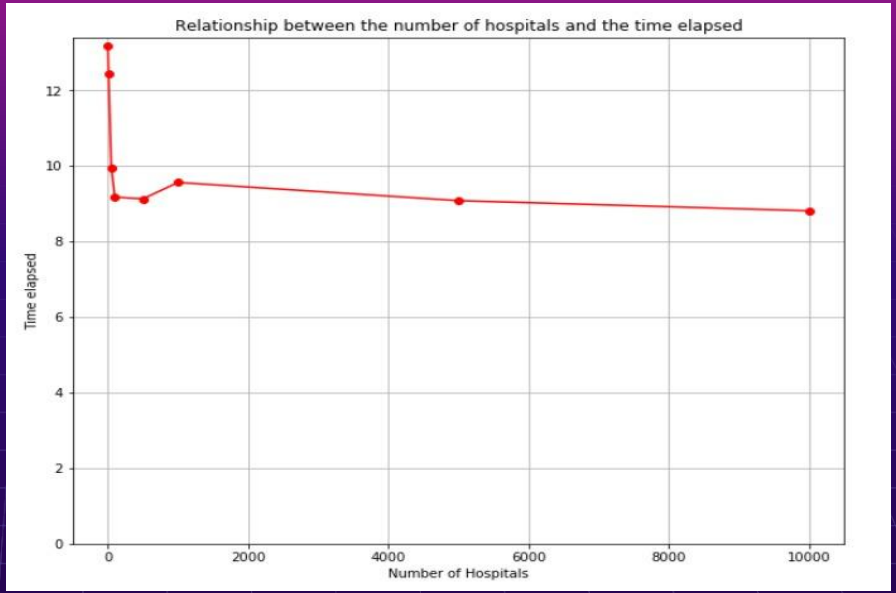


No clear relationship

Empirical Analysis

For .solve(), with parameter h:

	number of hospitals	time elapsed (seconds)
0	1	13.177167
1	10	12.431195
2	50	9.932098
3	100	9.175183
4	500	9.125928
5	1000	9.560966
6	5000	9.080527
7	10000	8.810948



(Almost) Negative Exponential

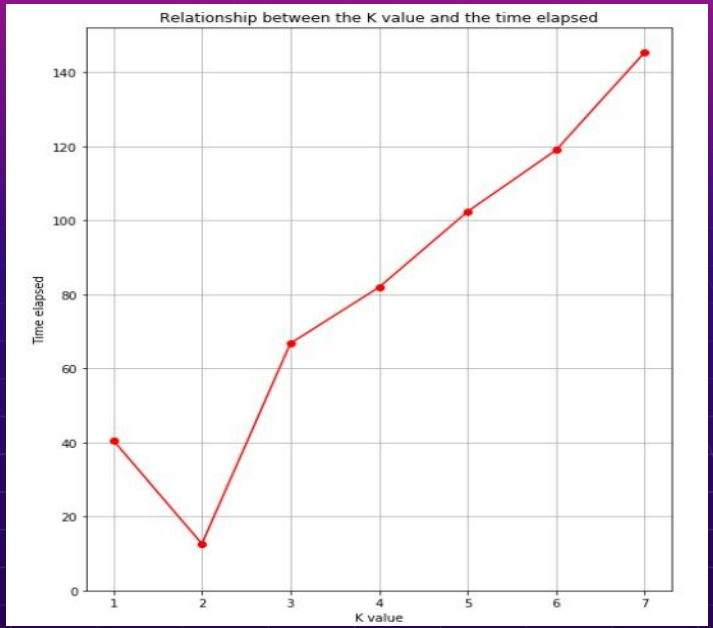
Empirical Analysis

For .topK(), with parameter k :

```
In [68]: data_k
```

Out[68]:

	time elapsed (seconds)
1	40.516993
2	12.597780
3	66.745062
4	81.864634
5	102.318465
6	118.932924
7	145.327438

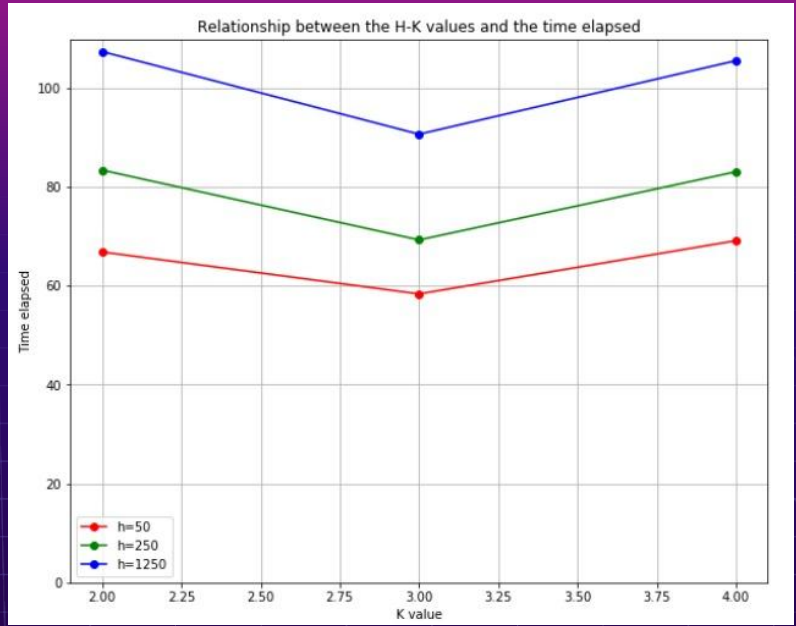


Increasing linearly (abnormality at k=2)

Empirical Analysis

For .topK(), with parameter:
h (50, 250, 1250)
K (2, 3, 4)

	timetaken(seconds, h=50)	timetaken(seconds, h=250)	timetaken(seconds, h=1250)
2	66.824084	83.371474	107.340773
3	58.342160	69.258589	90.634039
4	69.098429	83.041214	105.504102



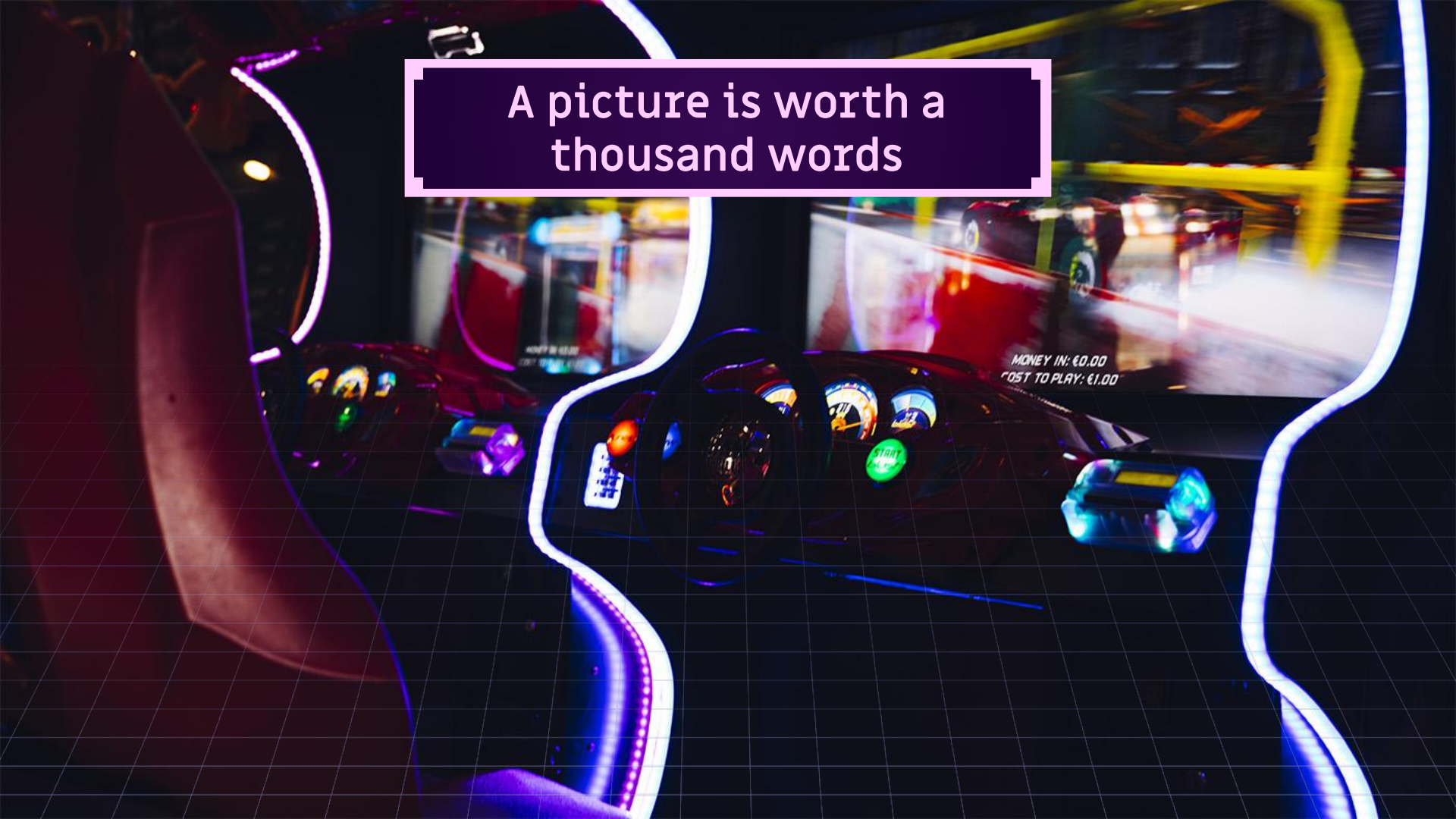
The graph shifts up corresponding to value of h and k

(h,k)↑ time↑

Conclusion

- Initial BFS is lengthy and takes multiple runs, our originally improved algo (.solve()) takes only one run
- Second algo (.topK()) is nearly a brute force method which consumes time.
- The time taken for both algorithms increases linearly with h and k.

Thank You!


A picture is worth a thousand words
MONEY IN: €0.00

This Is the Slide Title

Jupiter is a gas giant, the biggest planet in the Solar System and the fourth-brightest object in the sky. Jupiter, the fifth planet from the Sun, was named after a Roman god.

Major Requirements

Despite being red, Mars is actually cold

Mercury is the closest planet to the Sun

Saturn is composed of hydrogen and helium

Jupiter is the biggest planet of them all

Venus has a beautiful name, but it’s hot

Neptune is the outermost planet

Contents of This Template

Here's what you'll find in this **Slidesgo** template:

1. A slide structure based on a project proposal, which you can easily adapt to your needs. For more info on how to edit the template, please visit **Slidesgo School** or read our **FAQs**.
2. An assortment of illustrations that are suitable for use in the presentation can be found in the **alternative resources** slide.
3. A **thanks** slide, which you must keep so that proper credits for our design are given.
4. A **resources** slide, where you'll find links to all the elements used in the template.
5. **Instructions for use**.
6. Final slides with:

    1. The **fonts and colors** used in the template.
    2. A selection of **illustrations**. You can also customize and animate them as you wish with the online editor. Visit **Stories by Freepik** to find more.
    3. More **infographic resources**, whose size and color can be edited.
    4. Sets of **customizable icons** of the following themes: general, business, avatar, creative process, education, help & support, medical, nature, performing arts, SEO & marketing, and teamwork.

You can delete this slide when you're done editing the presentation.

“This is a quote, words full of wisdom that someone important said and can make the reader get inspired.”

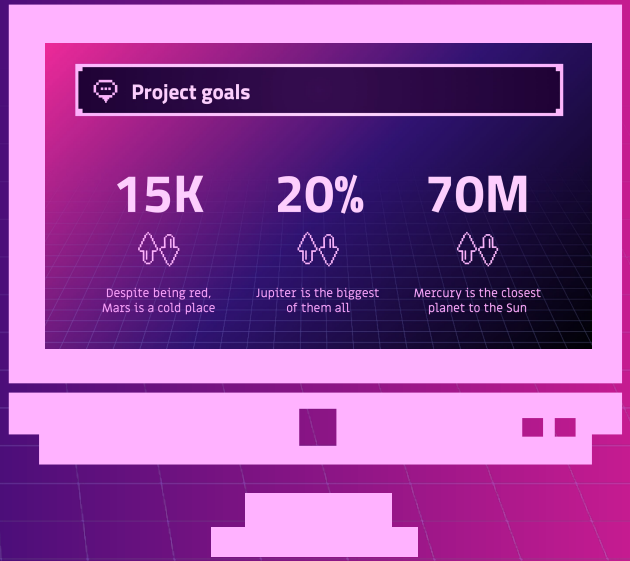
—Someone Famous

Predicted Results

	2020	2021	2021
Area 1	x	x	x
Area 2		x	x
Area 3			x

Sneak Peek

You can replace the image on the screen with your own work.
Right-click on it and then choose “Replace image” so you can add yours



Timeline

02

Venus is the second planet from the Sun

01

Despite being red, Mars is actually cold

03

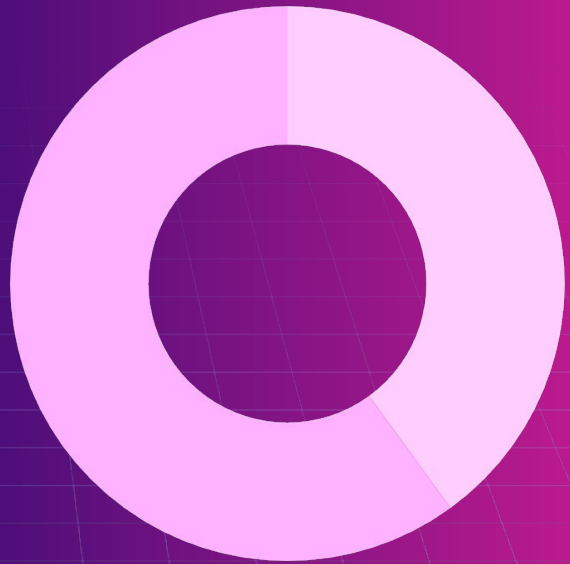
Saturn is a gas giant and has several rings

Budget

Total: €55,432,102

Despite being red, Mars is actually cold

Mercury is the closest planet to the Sun



To modify this graph, click on it, follow the link, change the data and replace it

Project Stages

Mars

Despite being red, Mars is actually a cold place

Jupiter

Jupiter is the biggest planet of them all

Mercury

Mercury is the closest planet to the Sun

Neptune

Neptune is the farthest planet from the Sun

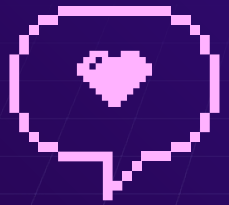
Our Locations

Despite being red, Mars is actually a cold place

Mercury is the closest planet to the Sun

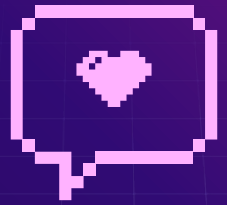
Jupiter is the biggest planet of them all

Our Partners



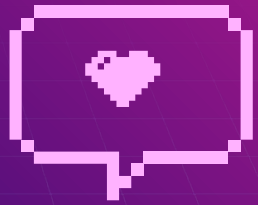
Mars

Despite being red, Mars is actually a cold place



Mercury

Mercury is the closest planet to the Sun



Jupiter

Jupiter is the biggest planet of them all

Our Team

Jane Doe

Here you could describe this team member

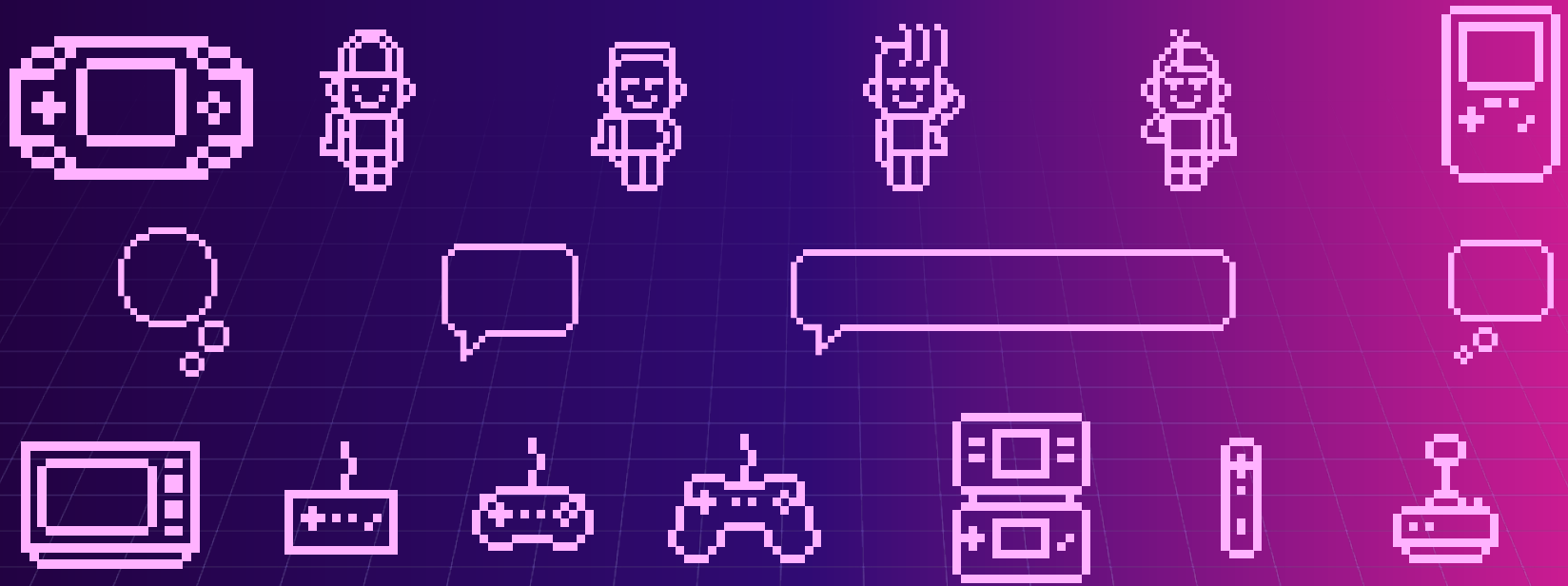
Sara Smith

Here you could describe this team member

Thanks!

Do you have any questions?
email@freepik.com
yourwebsite.com
+34 124 765487

Alternative Resources



Resources

Vectors

- Pack of social networking elements in pixel art style
- Consoles set in pixel art style
- Speech bubble collection and pixelated smiley
- White speech bubbles set in pixel art style
- Set of speech bubbles with messages and icons
- Collection of pixelated students
- Pack of social networking elements in pixel art style

Photos

- Racing arcade game with neon lights
- Close up hacker

Instructions for use

In order to use this template, you must credit **Slidesgo** by keeping the **Thanks** slide.

You are allowed to:

- Modify this template.
- Use it for both personal and commercial projects.

You are not allowed to:

- Sublicense, sell or rent any of Slidesgo Content (or a modified version of Slidesgo Content).
- Distribute Slidesgo Content unless it has been expressly authorized by Slidesgo.
- Include Slidesgo Content in an online or offline database or file.
- Offer Slidesgo templates (or modified versions of Slidesgo templates) for download.
- Acquire the copyright of Slidesgo Content.

For more information about editing slides, please read our FAQs or visit Slidesgo School:

https://slidesgo.com/faqs and https://slidesgo.com/slidesgo-school

Fonts & colors used

This presentation has been made using the following fonts:

Titillium Web
(https://fonts.google.com/specimen/Titillium+Web)

Miriam Libre
(https://fonts.google.com/specimen/Miriam+Libre)

#a26efa

#ffa0ff

#ffb2ff

#ffceff

#f22099

#300b74

#000003

Stories by Freepik

Create your Story with our illustrated concepts. Choose the style you like the most, edit its colors, pick the background and layers you want to show and bring them to life with the animator panel! It will boost your presentation. Check out How it Works.

Pana

Amico

Bro

Rafiki

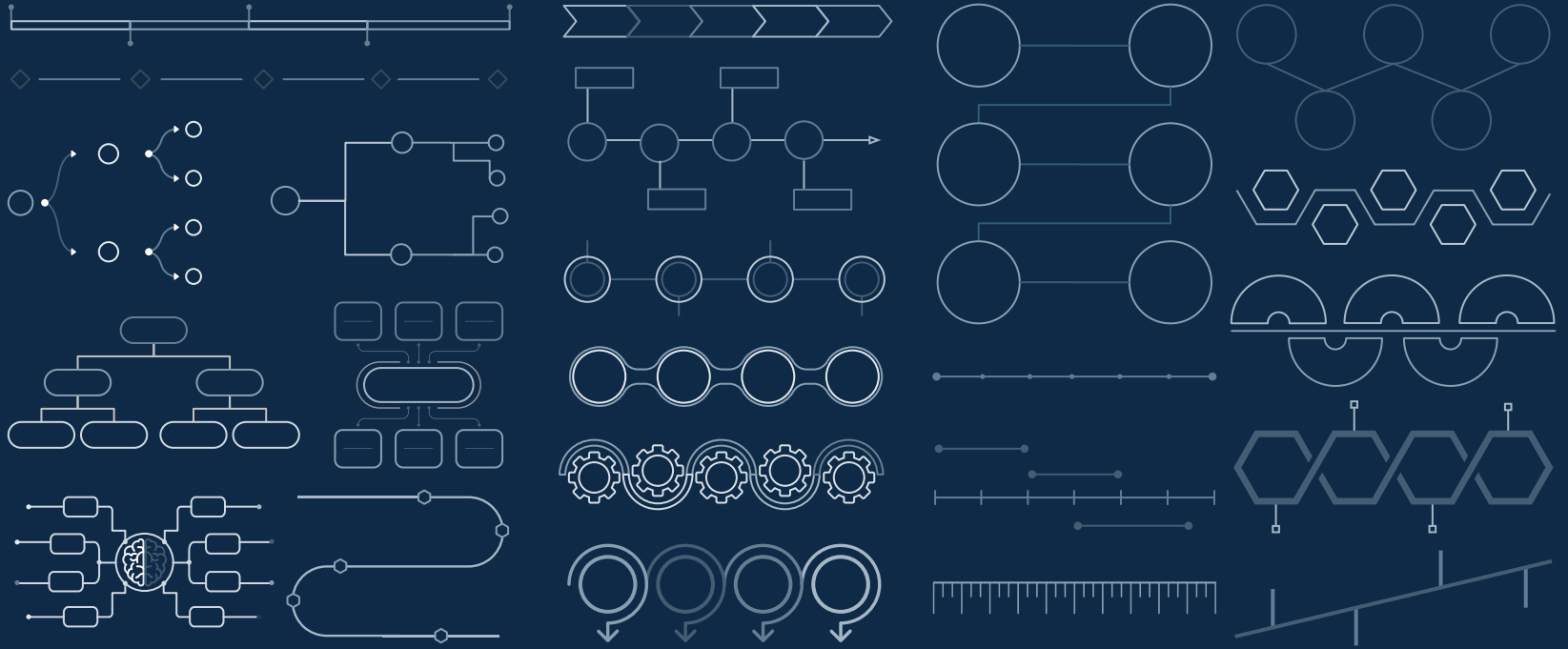
Cuate

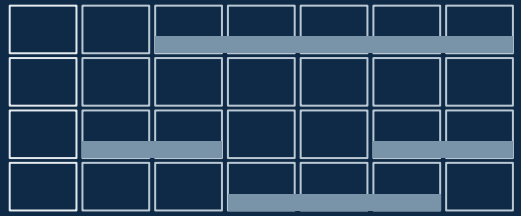
Use our editable graphic resources...

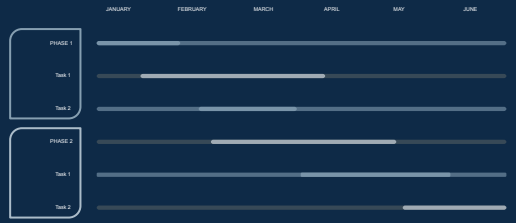
You can easily resize these resources without losing quality. To change the color, just ungroup the resource and click on the object you want to change. Then, click on the paint bucket and select the color you want. Group the resource again when you're done. You can also look for more infographics on Slidesgo.

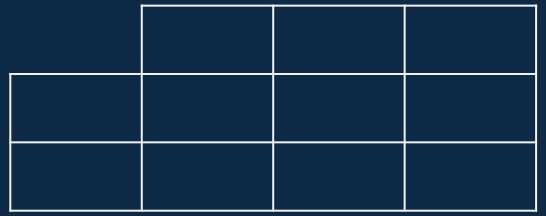


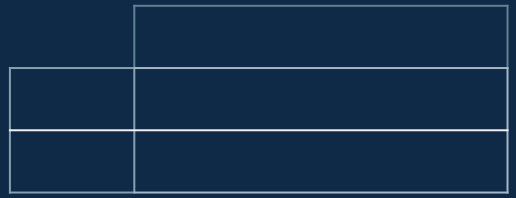


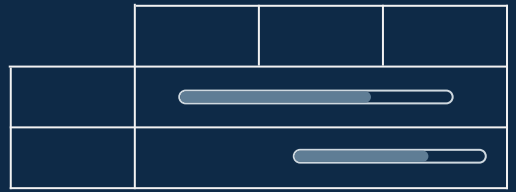



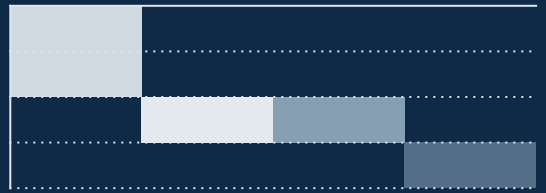


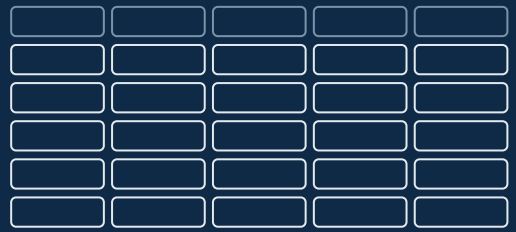

JANUARY
FEBRUARY
MARCH
APRIL
MAY
JUNE
PHASE 1
Task 1
Task 2
PHASE 2
Task 1
Task 2

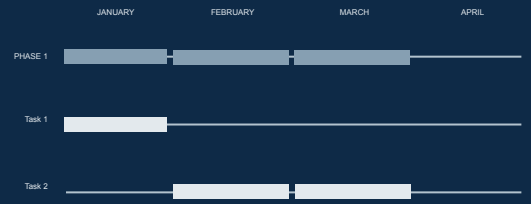


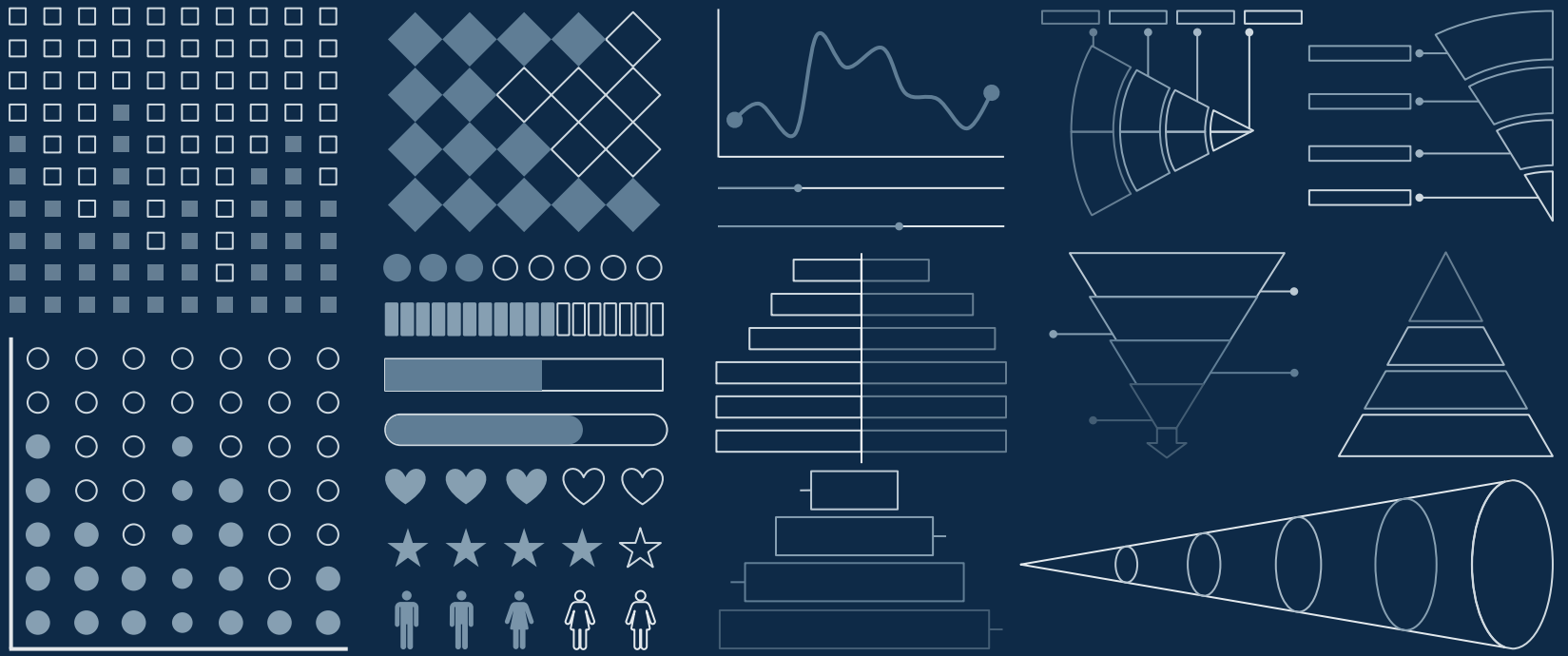









JANUARY
FEBRUARY
MARCH
APRIL
PHASE 1
Task 1
Task 2



...and our sets of editable icons

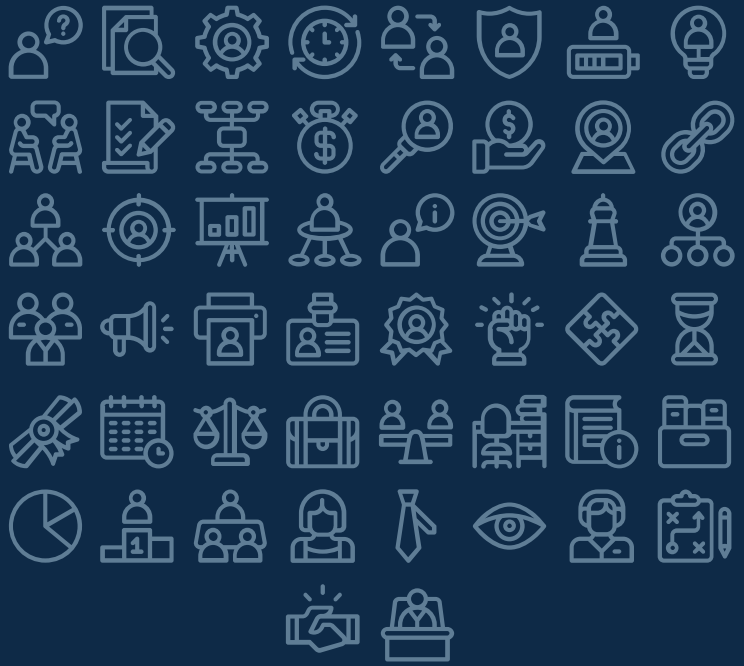
You can resize these icons without losing quality.
You can change the stroke and fill color; just select the icon and click on the paint bucket/pen.
In Google Slides, you can also use Flaticon's extension, allowing you to customize and add even more icons.

Educational Icons

Medical Icons

Business Icons

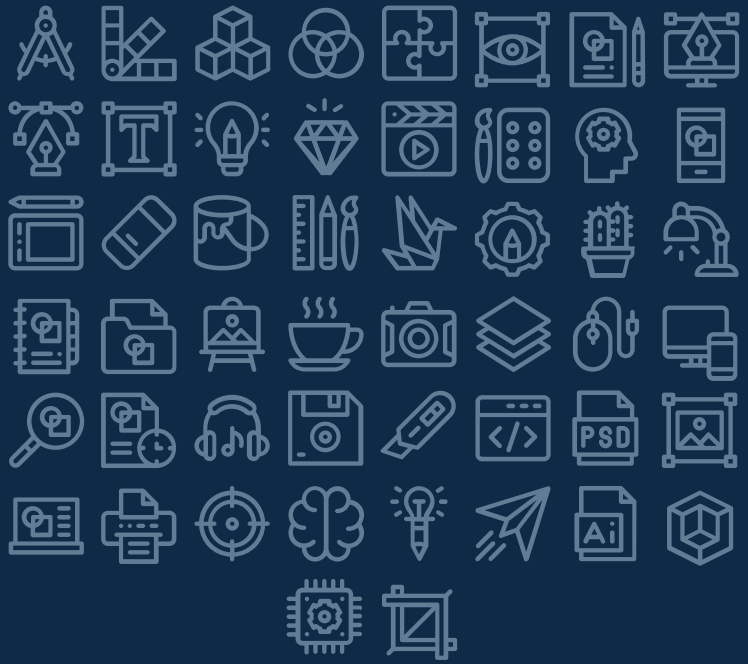
Teamwork Icons



Help & Support Icons

Avatar Icons

Creative Process Icons



Performing Arts Icons

Nature Icons

SEO & Marketing Icons


slidesgo