Project Dijkstra (yes, we spelt it correctly)

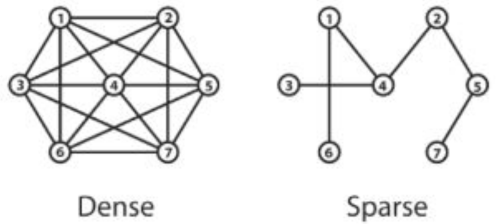
Yap Shen Hwei
Nicholas Png
Marcus Soh
Vignesh

Content Page

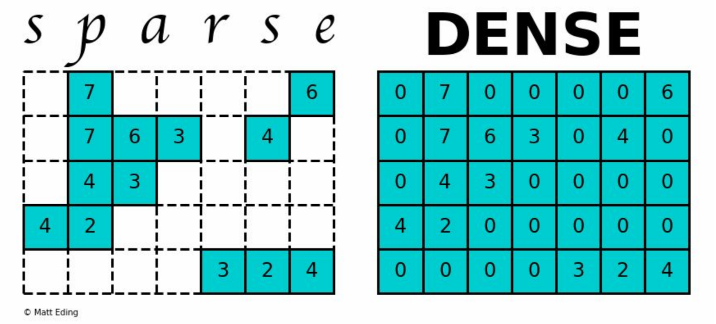
- Adjacency Matrix + Array (Time Complexity)
- Adjacency List + MinHeap (Time Complexity)
- Our Implementation
- Sparse vs Dense Graph (Theoretical)
- Empirical Trends
- Array + AdjMatrix **vs** MinHeap + AdjList
- Conclusion

Sparse & Dense Graph Definition

- Sparse graph is a graph in which the number of edges is close to the <u>minimal</u> number of edges - sparsely connected
  - |E| = **|V|**
  - Binary trees
  - Can exist as a disconnected graph

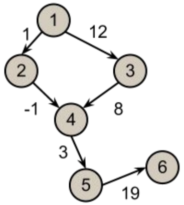


- Dense graph is a graph in which the number of edges is close to the <u>maximal</u> number of edges - densely connected
  - |E| = |V| * |V-1|
    = **|V|**²
  - Complete graph



Time Complexity: Adjacency Matrix + Array

	Time Complexity
Extract cheapest vertices for processing	Number of Vertices * (Extract next cheapest node + Relaxation of nodes) = O(|V|) * [O(|V|) + O(|V|)] **= O|V|^2**
Update Shortest Distance	Number of Edges * Update array = O(|E|) * O(1) = **O(|E|)**
Overall	O(|V|^2 + |E|) = **O(|V|^2)**



Weighted Directed Graph

	1	2	3	4	5	6
1	0	1	12	0	0	0
2	-1	0	0	-1	0	0
3	-12	0	0	8	0	0
4	0	1	-8	0	3	0
5	0	0	0	-3	0	19
6	0	0	0	0	-19	0

Adjacency Matrix

Time Complexity: Adjacency List + MinHeap

	Time Complexity
Heapify	**O(log|V|)**
Extract Vertex from Heap	Number of vertices * Cost of extracting root node = O(|V|) * [O(log|V|) * O(1)] = **O(|V| * log|V|)**
Updating Path Cost	Number of edges * Cost of updating = O(|E|) * O(log|V|) = **O(|E| * log|V|)**
Overall	O(<u>|E|log|V|</u> + <u>|V|log|V|</u> + <u>log|V|</u>) = **O([|E| + |V|] *log|V|)**

Theoretical Best Case

Sparse Graph → **|E| = |V| - 1**

	Adjacency Matrix + Array	Adjacency List + MinHeap
Time Complexity (Formulae)	$\|V\|^2$	$\|E\| \log \|V\|$
Vertices	1024	1024
Edges	1023	1023
Time Complexity	1,047,552	**3079**

Theoretical Worse Case

Dense Graph → **E = V x (V - 1)**

	Adjacency Matrix + Array	Adjacency List + MinHeap
Size of Priority Queue	$\|V\|^2$	$(\|E\|+\|V\|)*\log \|V\|$
Vertices	1024	1024
Edges	1,047,552	1,047,552
Time Complexity	**1,048,576**	3,156,528

Our Implementation: PriorityQueue() using Array

```python
class PriorityQueue(object):
    def __init__(self):
        self.queue = []

    def __str__(self):
        return ' '.join([str(i) for i in self.queue])

    # if empty return True
    def isEmpty(self):
        return len(self.queue) == 0

    # vertex is stored with distance
    # vertex 2 with distance 20: (2, 20)
    def insert(self, vertex):
        self.queue.append(vertex)

    def get_smallest(self):
        min_val = sys.maxsize
        min_index = 0
        for i in range(len(self.queue)):
            if self.queue[i][1]<min_val:
                min_val = self.queue[i][1]
                min_index = i
        smallest = self.queue[min_index]
        del(self.queue[min_index])
        return smallest

    def remove_vertex(self, vertex):
        for i in range(len(self.queue)):
            if self.queue[i][0] == vertex:
                item = self.queue[i]
                del(self.queue[i])
                return item
```

Our Implementation: Heap()

```python
def removeVertex(self, vertex):
    for i in range(self.size):
        # print(self.storage[i][0])
        if self.storage[i][0] == vertex:
            #print("vertex: ", vertex, " storage: ", self.storage)
            self.swap(i, self.size-1)
            del self.storage[self.size-1]
            self.size -= 1
            self.heapifyDown(i)
            #print("after heapifyDown: ", self.storage)
            break

def heapifyDown(self, index):
    while(self.hasLeftChild(index)):
        smallerChildIndex = self.getLeftChildIndex(index)
        self.comparisons += 1
        if(self.hasRightChild(index) and self.rightChild(index)[1]< self.leftChild(index)[1]):
            smallerChildIndex = self.getRightChildIndex(index)
        if (self.storage[index][1] < self.storage[smallerChildIndex][1]):
            break
        else:
            self.swap(index, smallerChildIndex)
        index = smallerChildIndex
```

```python
def swap(self, i1, i2):
    self.storage[i1], self.storage[i2] = self.storage[i2], self.storage[i1]
                    (parameter) vertex: Any
def insert(self, vertex):
    if(self.isFull()):
        raise("Heap is full")
    self.storage.append(vertex)
    self.size += 1
    self.heapifyUp()

def heapifyUp(self):
    # print("heapifyUp curr size", self.size)
    index = self.size-1
    while(self.hasParent(index) and self.parent(index)[1] > self.storage[index][1]):
        self.comparisons += 1
        self.swap(self.getParentIndex(index), index)
        index = self.getParentIndex(index)
    # print("after heapifyUp", self.storage)

def removeMin(self):
    if(self.size==0):
        raise("Empty Heap")
    node = self.storage[0]
    # print("removeMin Node", node)
    self.storage[0] = self.storage[self.size-1]
    del self.storage[self.size-1]
    self.size -=1
    self.heapifyDown(0)
    return node
```

Our Implementation: Graph()

```python
class Graph():

    def __init__(self, v):
        self.V = v #number of nodes
        self.graph = [[0 for column in range(v)]for row in range(v)]
        self.adjL = {}
        self.d = [sys.maxsize for x in range(v)]
        self.pi =[None for x in range(v)]
        self.S = [0 for x in range(v)]
        self.comparisonsMHTotal, self.comparisonsDijkstraMH = 0, 0
        self.comparisonsArrTotal, self.comparisonsDijkstraArr = 0, 0

    def getSize(self):
        return self.V
    def adjList(self):
        for i in range(self.V):
            adjNodes = []
            for j in range(self.V):
                if self.graph[i][j]:
                    # (adjNode, weight), adjNodes[0] == the adjNode, adjNodes[1] ==
                    # the distance between node current node i and node j
                    adjNodes.append((j,self.graph[i][j]))
            self.adjL[i] = adjNodes
```

For when using Heap

```python
def dijkstraHQ(self, source):
    # reinitializing
    self.d[source] = 0

    pq2 = MinHeap(self.V)

    for i in range(self.V):
        pq2.insert([i, self.d[i]])


    while(pq2.isEmpty() == False):
        u = pq2.removeMin()
        self.S[u[0]] = 1
        # for each v= (node, weight) adj to u[0]
        for v in self.adjL[u[0]]:
            if self.S[v[0]] != 1 and self.d[v[0]] > self.d[u[0]] + v[1]:
                pq2.removeVertex(v[0])
                self.d[v[0]] = self.d[u[0]] + v[1]
                self.pi[v[0]] = u[0]
                pq2.insert([v[0], self.d[v[0]]])
```

For when using Priority Queue (Array)

```python
def dijkstra(self,source):
    self.d[source] = 0
    pq = PriorityQueue()
    for i in range(self.V):
        pq.insert((i,self.d[i]))

    while(pq.isEmpty()==False):
        u = pq.get_smallest()
        # u = (node, weight)
        self.S[u[0]] = 1
        # for each v adjacent to u
        for i in range(self.V):
            if self.graph[u[0]][i]> 0 and self.S[i] != 1 and self.d[i] > self.d[u[0]] + self.graph[u[0]][i]:
                pq.remove_vertex(i)
                self.d[i] = self.d[u[0]] + self.graph[u[0]][i]
                self.pi[i] = u[0]
                pq.insert((i,self.d[i]))
```

Our Methods: Important Functions

```
params : n == the number of nodes, density == the number of edg
f generateConnectedGraph(n, density):
    mat = [[0 for i in range(n)] for j in range (n)]
    setOfConnectedNodes = []
    i = 0
    while (i < n):
        # weight = randint(1,10)
        setOfConnectedNodes.append(i)
        if (i <= 1):
            weight = randint(1,n*2)
            if (i == 1):
                mat[0][1] = weight
                mat[1][0] = weight
            i+=1
            continue

        for j in range(density+1):
            weight = randint(1,n*2)
            adjI = randint(0, i-1)
            adj = setOfConnectedNodes[adjI]
            mat[i][adj] = weight
            mat[adj][i] = weight

        i+=1

    return mat
✓  0.2s
```

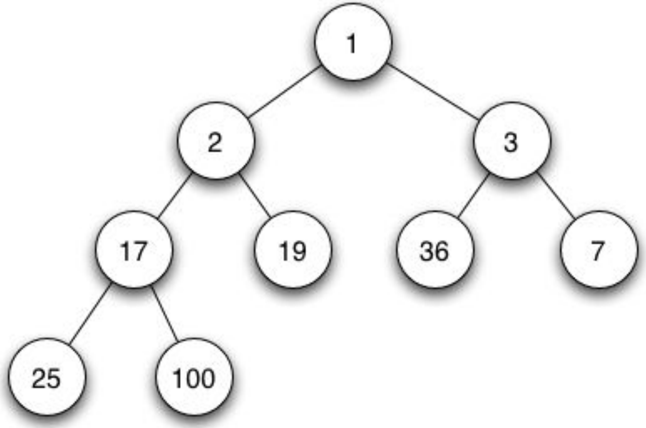
```
(n, howMany,useMinHeap, density):
 []
n range(howMany):
t = generateConnectedGraph(n, density)
 Graph(n)
raph = gMat

 (useMinHeap):
  time.append(runDijkstraHQ(g, 0))
e:
  time.append(runDijkstra(g, 0))

 sum(time)/len(time)
```

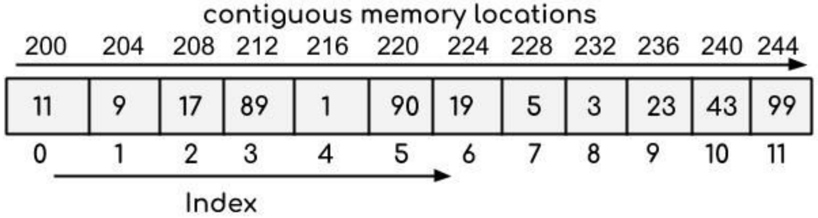
Generate Connected Graphs

1. This function allows you to change the
sity of graph, such that each node can be
nected to at most a certain number
sity) of other nodes.
e that the density < number of nodes,
ause one node can have at most V-1
es!

Average Time Calculation

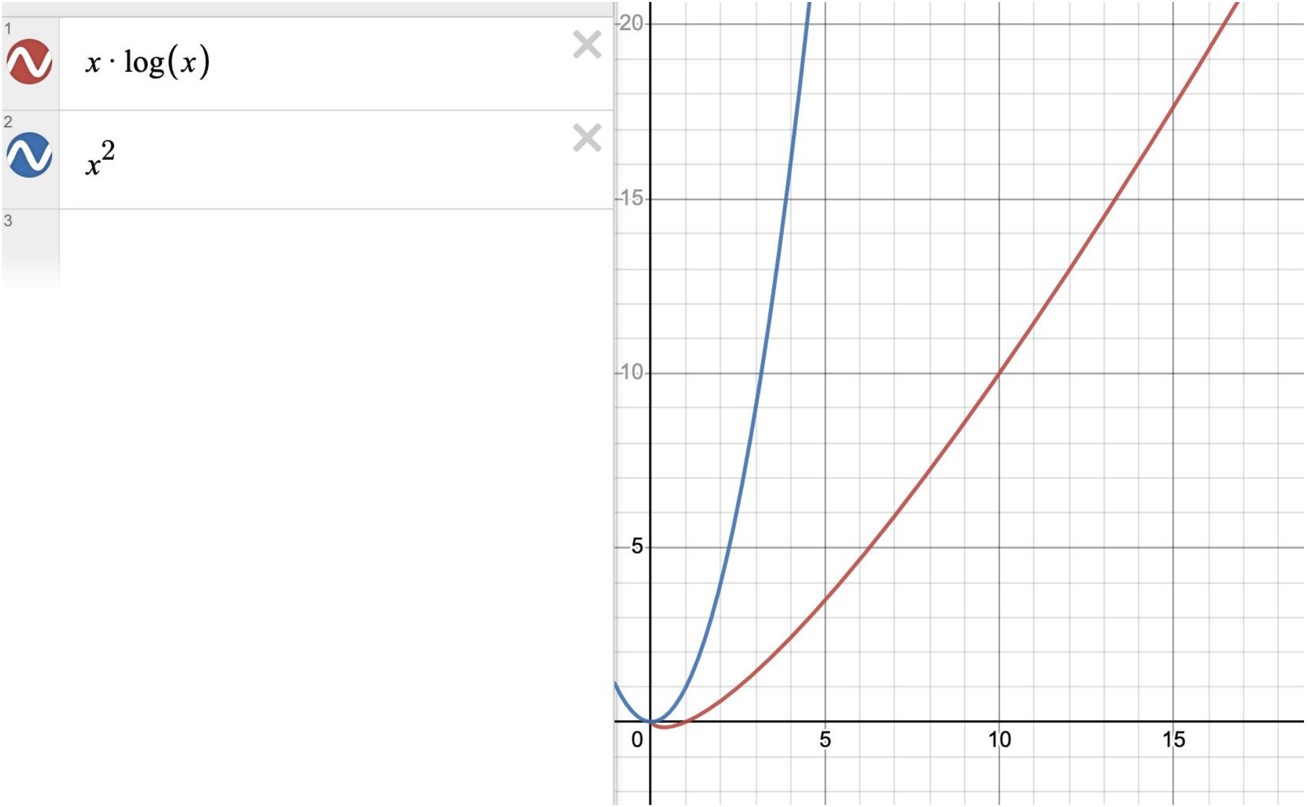
1. Important to obtain averages of the same case, but different graphs.



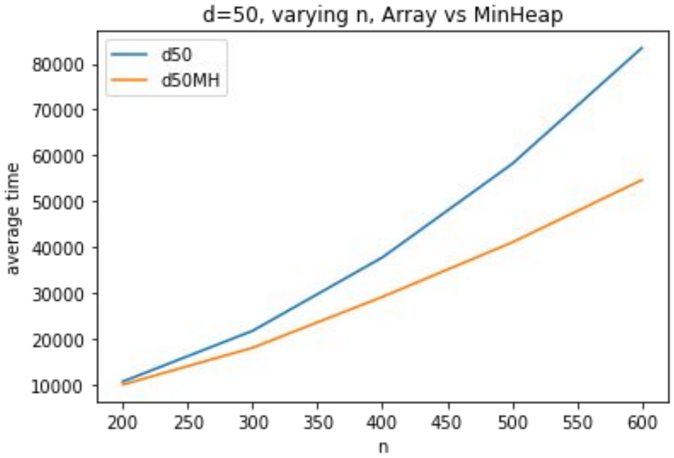


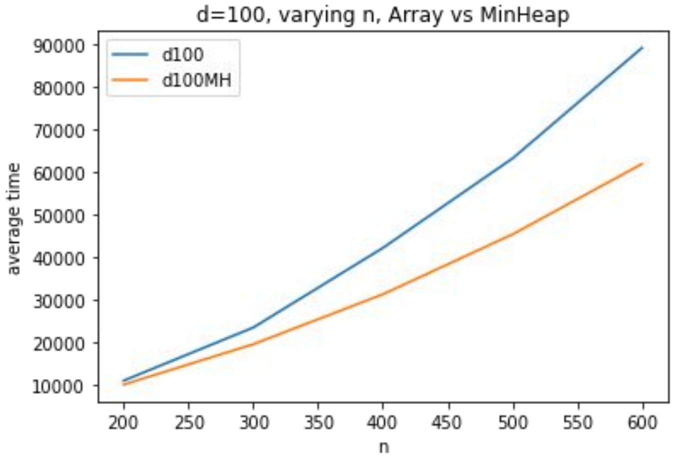
Array vs MinHeap

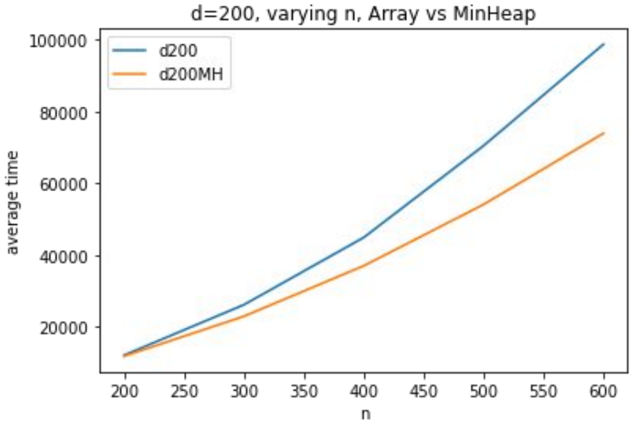
It follows the general trend

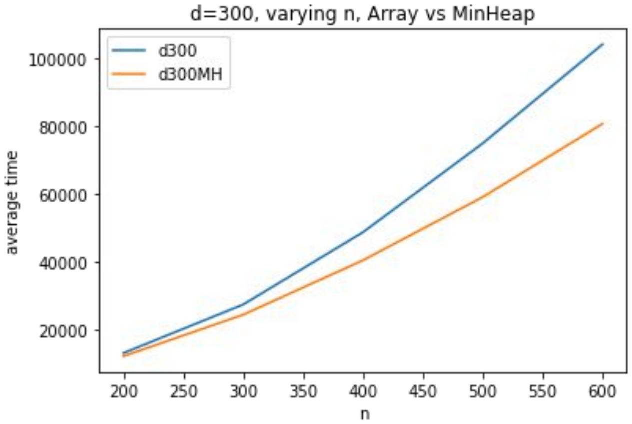


Array vs MinHeap, varying n



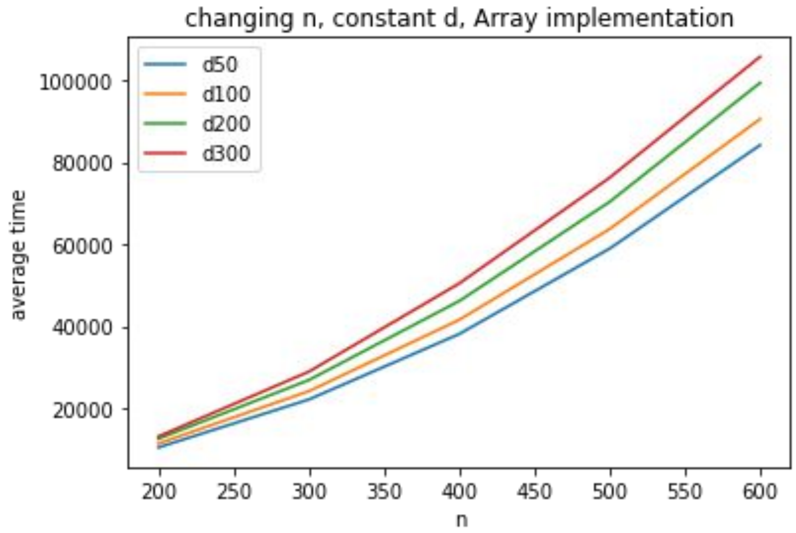




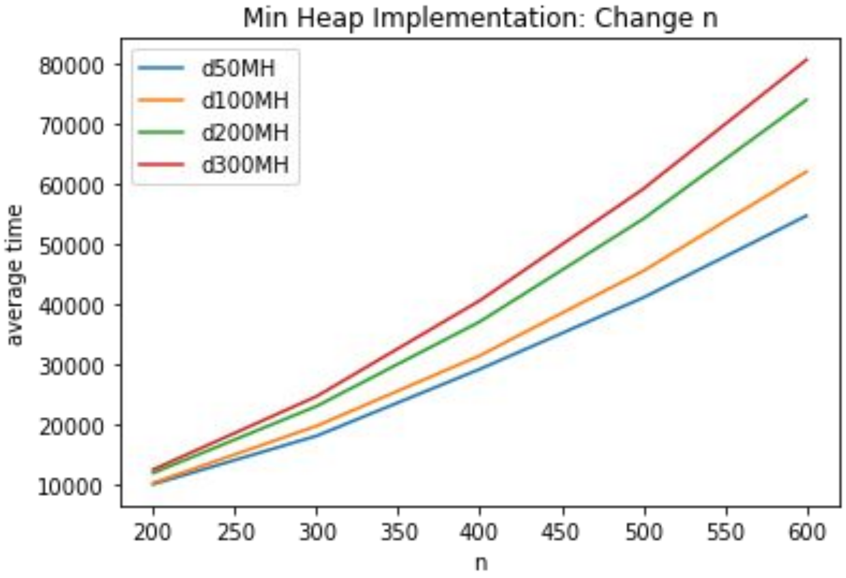


Empirical Trends: Changing n, constant densities

d = 50, 100, 200, 300

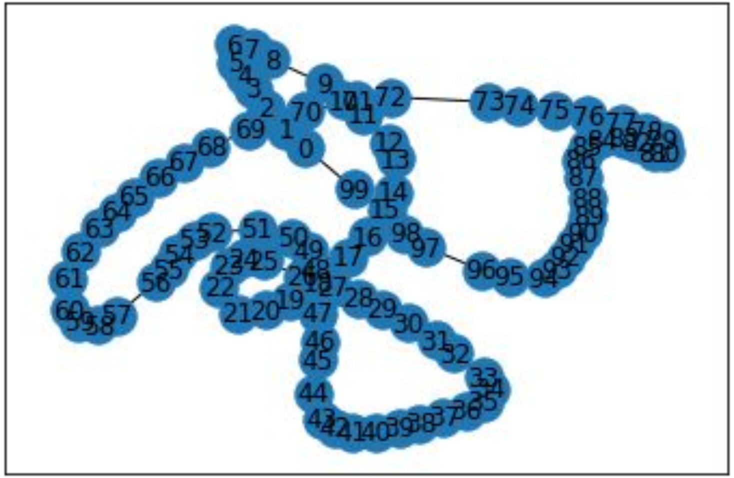


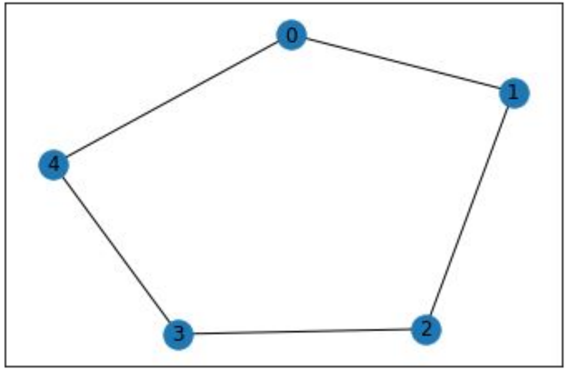
Array,
adjMat

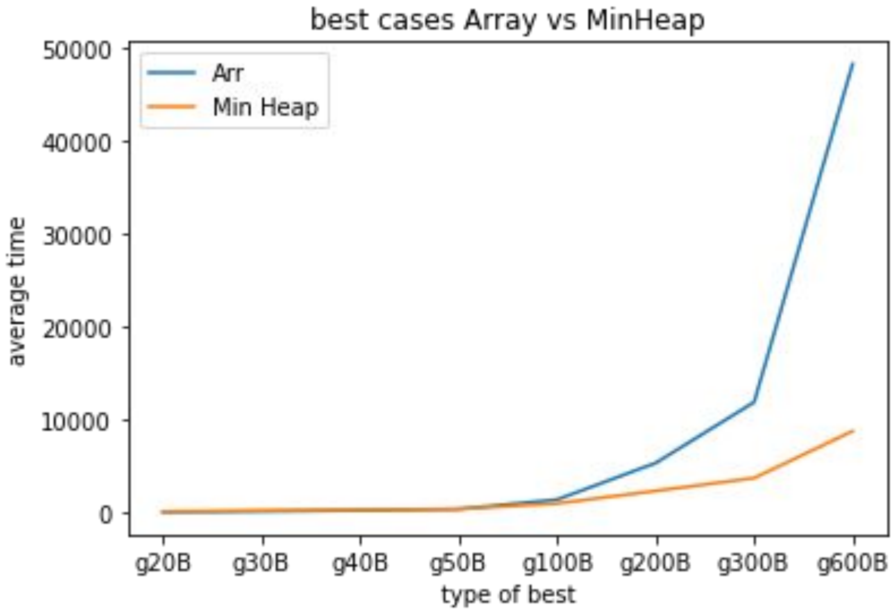


Min Heap,
adjList

The best case for MinHeap?





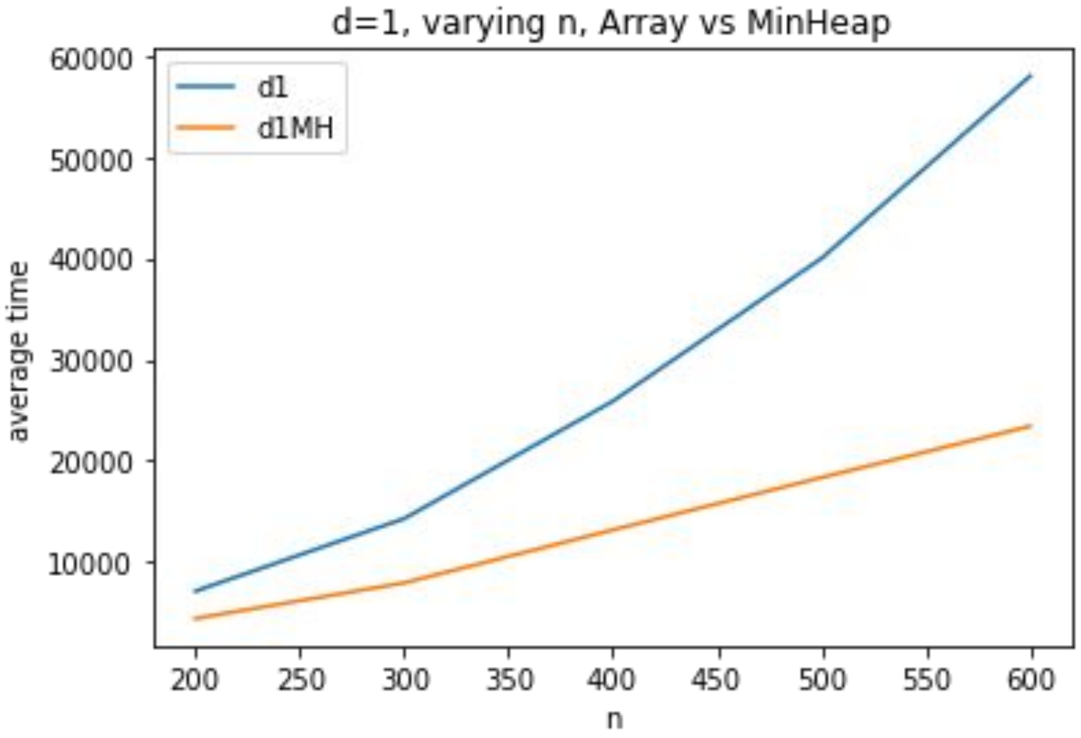


Rate of growth in average time much much **lower** using Minimum Heap for sparse graph

Reasoning: Using adjacency list, we only need to loop through the adjacent nodes, not all the nodes.

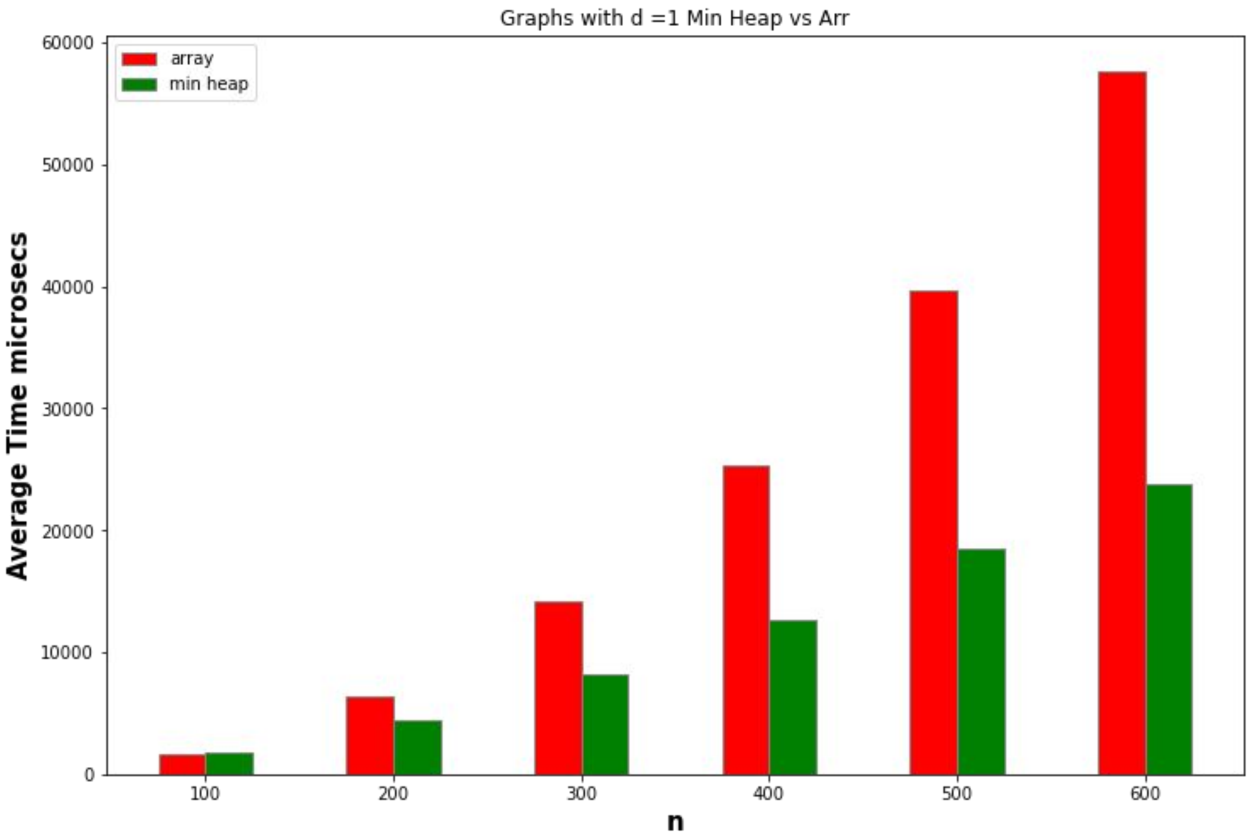
The case of d = 1:

- Sparse connected graph

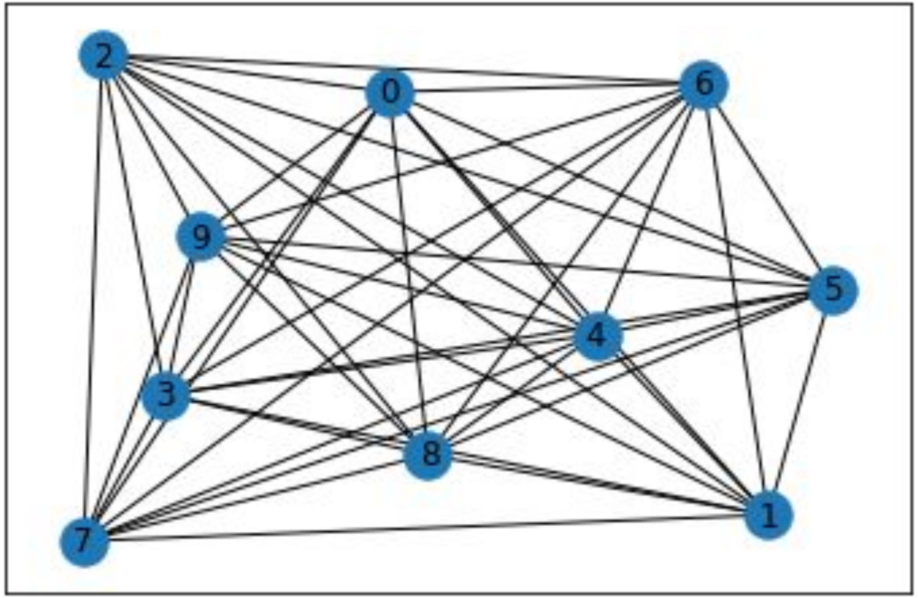


We can observe that MinHeap performs better when the graph is sparsely connected

What if it was sparse in general?

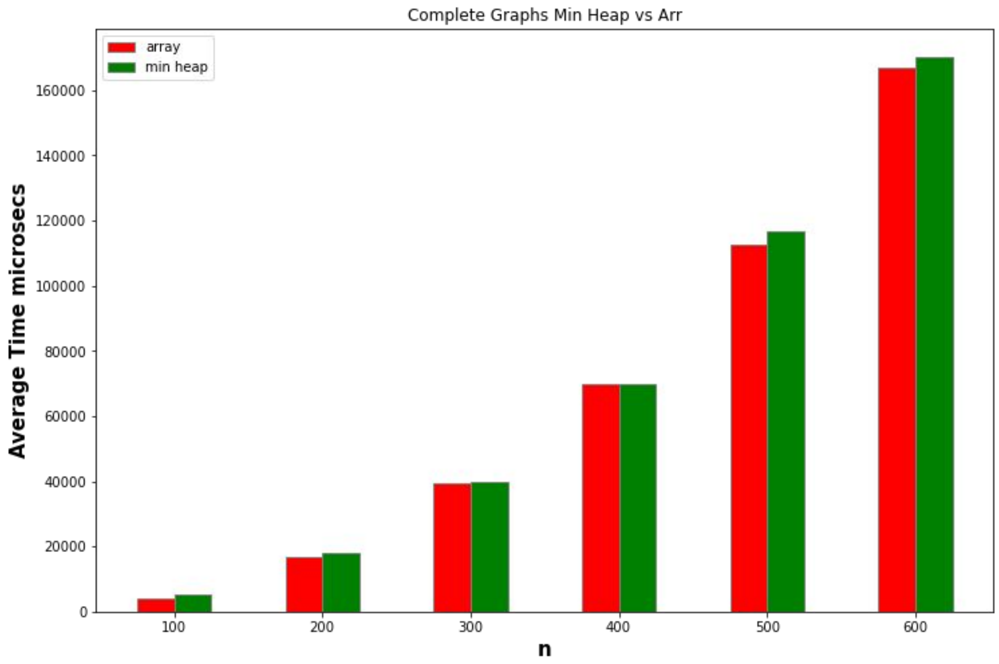


The best case for Array?



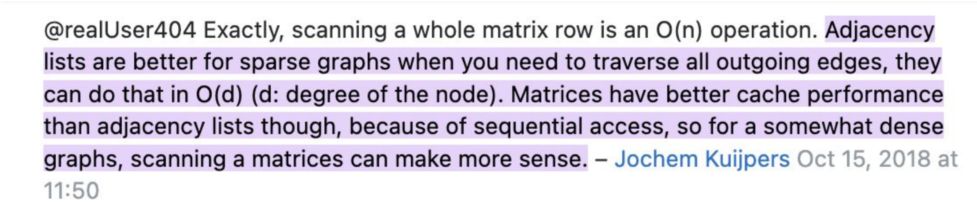
We would expect that complete graphs will be the best cases for adjacency matrix (& using Array)

Using Array Performs Better: Complete Graphs

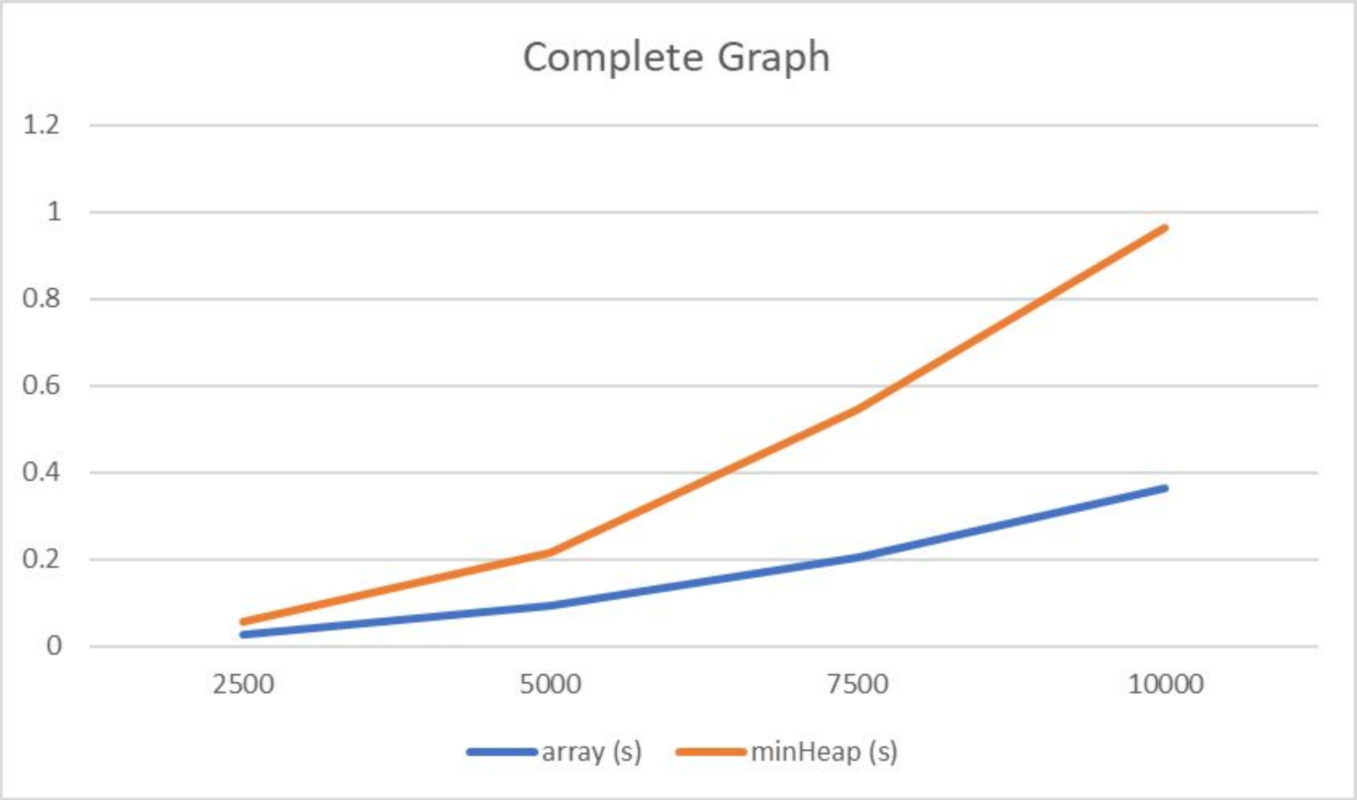


Because array implementation uses AdjMatrix, and Matrices have better cache performance.

(according to Stack Overflow)


@realUser404 Exactly, scanning a whole matrix row is an O(n) operation. Adjacency lists are better for sparse graphs when you need to traverse all outgoing edges, they can do that in O(d) (d: degree of the node). Matrices have better cache performance than adjacency lists though, because of sequential access, so for a somewhat dense graphs, scanning a matrices can make more sense. – Jochem Kuijpers Oct 15, 2018 at 11:50

Using Array Performs Better: Complete Graphs (in C)



C implementation

Conclusion

If the graph is dense and compact → Complete graph

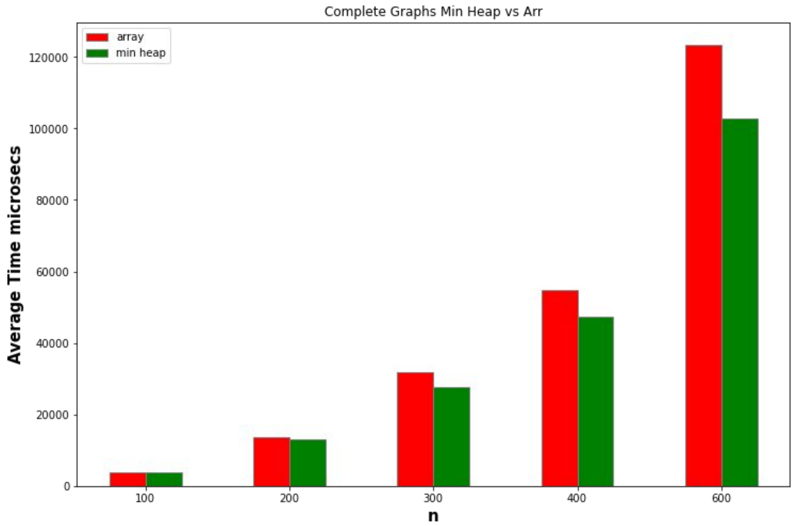
- Use Adjacency Matrix with Array as Priority Queue

If the graph is sparse and less compact → Incomplete graph

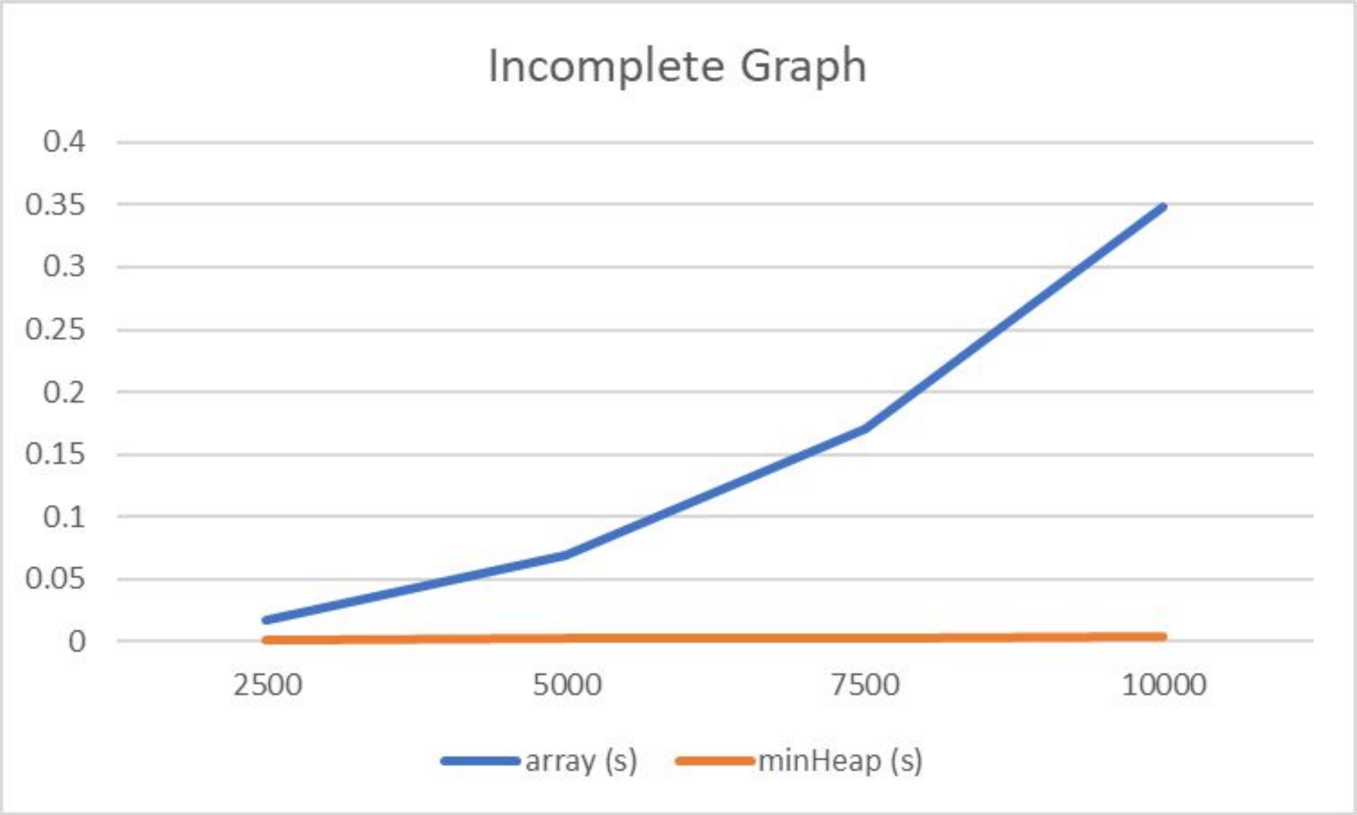
- Use Adjacency List with MinHeap as Priority Queue

Some other findings and thoughts

- Using **numpy matrix to represent our AdjMatrix may make operations faster** (in the context of Python)
- The variation in weights also affects performance.
    - This is because by mistake, we randomly generated **weights only from [1,10]**
    - **This may suggest that the dijkstra takes longer to find the shortest path because the difference between values may be very small and therefore**, when you pick from the queue, **there are high chances that you have to explore more nodes.**
    - Our **array implementation showed worse performance (weights [1,10])** highly because the advantage in better cache performance was **masked by the better O(ElogV) of the MinHeap Implementation.**
- CPython!



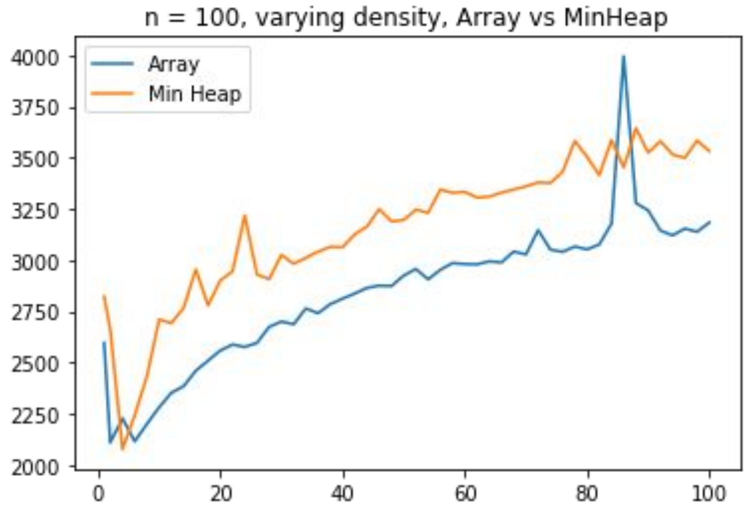
Incomplete Graphs (in C)

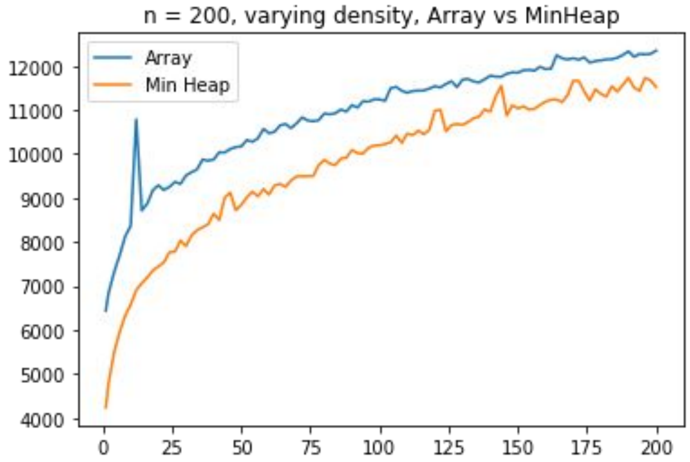


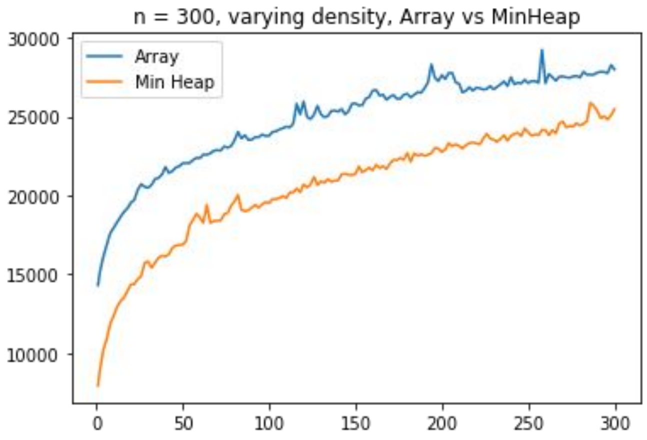
C implementation

Thank you :)

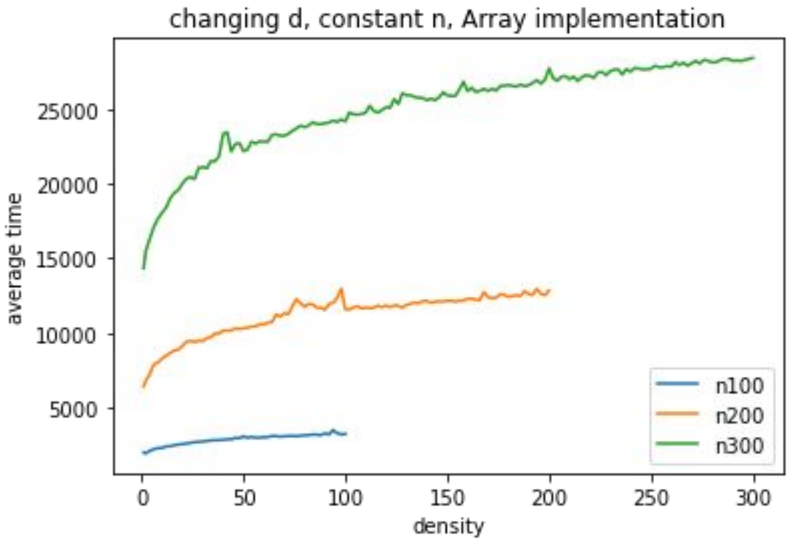
Array vs MinHeap, varying density



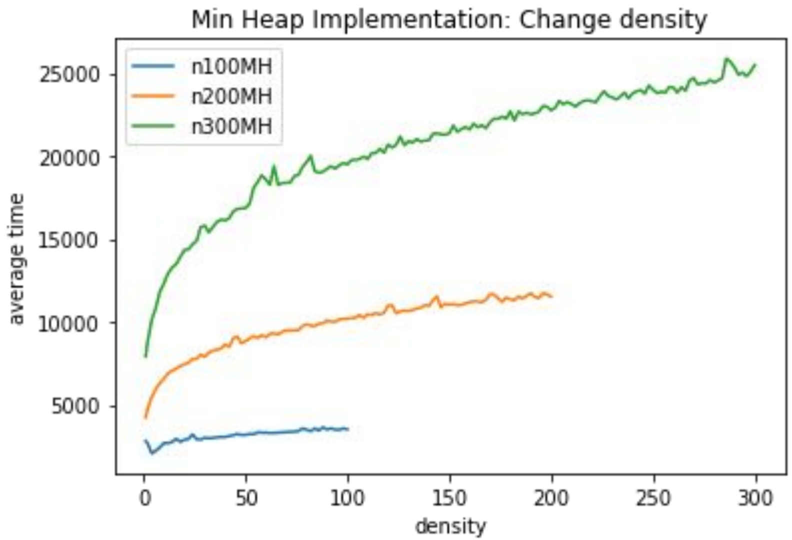




Changing Density, Constant n for n = 100,200,300



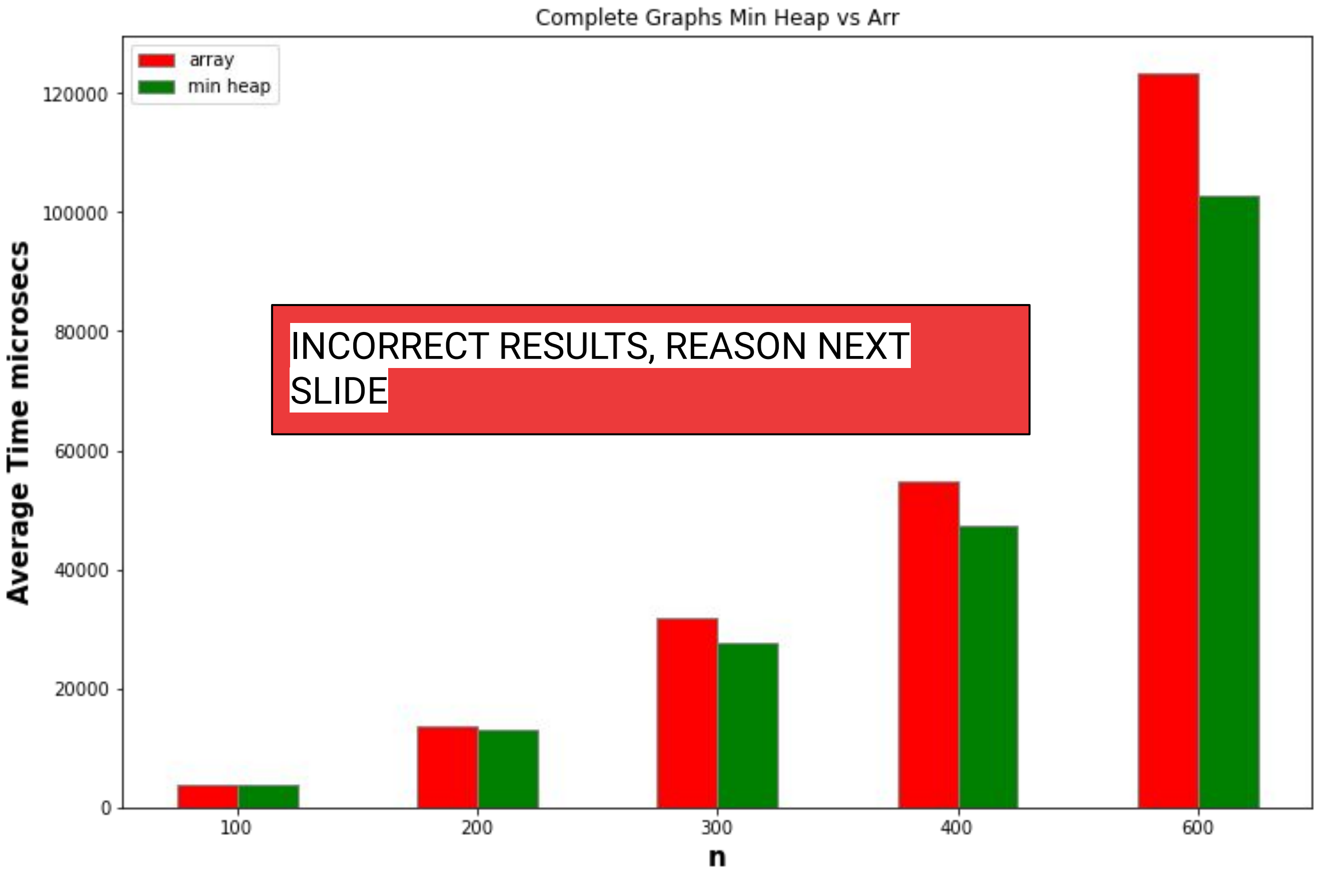
Array, using adjMatrix



MinHeap, using adjList

Initially, we got our results as this:

- This seems incorrect, because
- Complete graphs theoretically favours Array Implementation



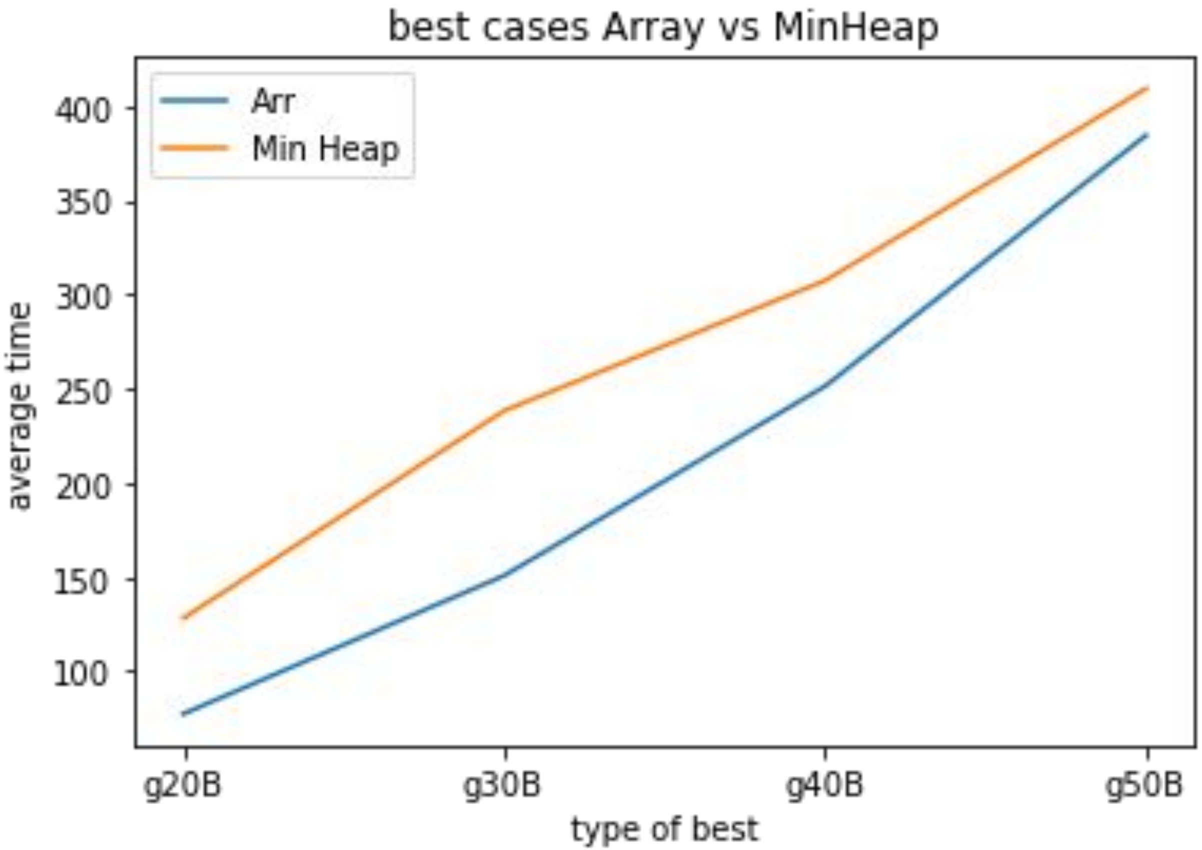
Found the problem! Range of Weights too small!

```python
# params : n == the number of nodes, density == the number of edges that o
def generateConnectedGraph(n, density):
    mat = [[0 for i in range(n)] for j in range (n)]
    setOfConnectedNodes = []
    i = 0
    while (i < n):
        # weight = randint(1,10)
        setOfConnectedNodes.append(i)
        if (i <= 1):
            weight = randint(1,10)
            if (i == 1):
                mat[0][1] = weight
                mat[1][0] = weight
            i+=1
            continue

        for j in range(density+1):
            weight = randint(1,10)
            adjI = randint(0, i-1)
            adj = setOfConnectedNodes[adjI]
            mat[i][adj] = weight
            mat[adj][i] = weight

        i+=1

    return mat
```

The range of random integers too small.

- Therefore, there are many repeats of the same weight
- This case may favour min heap implementation more, because it's likely to make the search

BUT if we zoom in



We can observe that the Array performs slightly better.

This may suggest that the

Project Management Infographics

Mars

Despite being red, Mars is a cold place

Jupiter

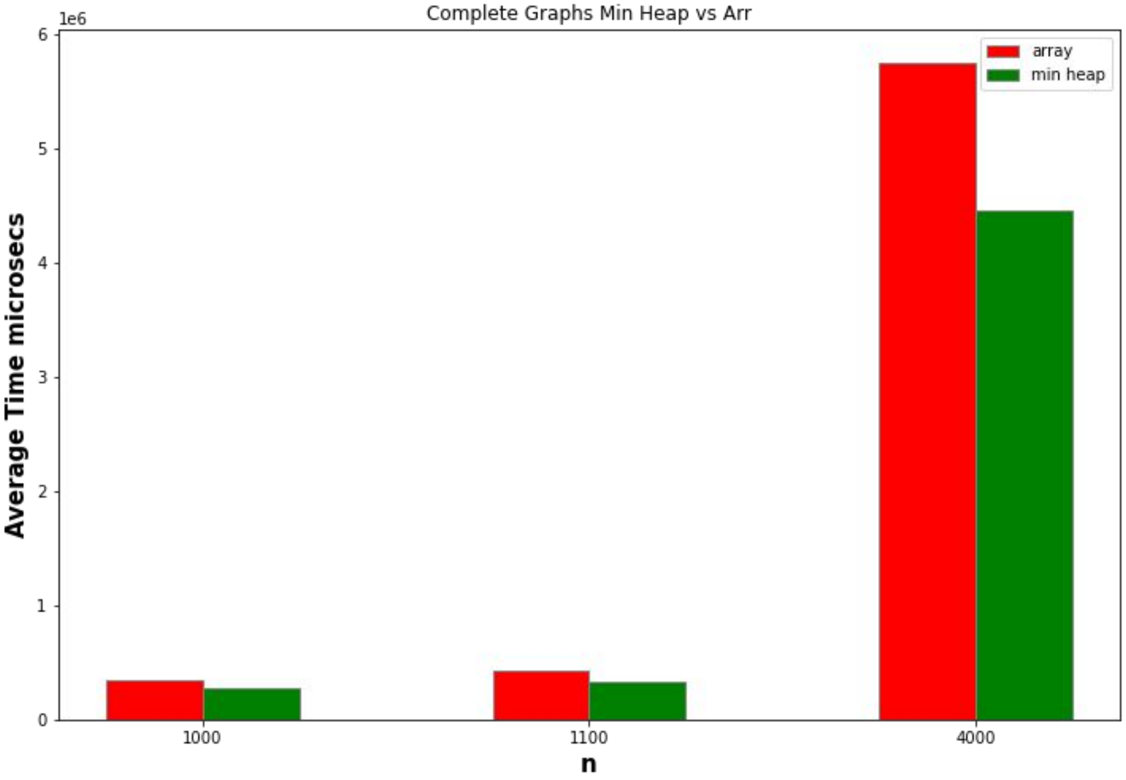
Jupiter is a gas giant and the biggest planet

Saturn

It’s composed of hydrogen and helium

Neptune

Neptune is the farthest planet from the Sun


Complete Graphs Min Heap vs Arr
1e6
6
5
4
3
2
1
0
Average Time microsecs
array
min heap
1000
1100
4000
n

Project Management Infographics

Mercury

Mercury is the closest planet to the Sun and the smallest one

Neptune

Neptune is the farthest planet from the Sun and the fourth-largest one

Mars

Despite being red, Mars is a cold place full of iron oxide dust

Earth

Earth is the third planet from the Sun and the only one that harbors life

Project Management Infographics

Neptune

Neptune is the farthest planet from the Sun and a gas giant

Mars

Despite being red, Mars is actually a cold place

Venus has a nice name and is the second

Earth is the third planet from the Sun

Earth

Jupiter is a gas giant and the biggest planet of them all

Jupiter

Project Management Infographics

Venus

Mercury

Mercury is the smallest planet in the Solar System

Jupiter

Jupiter is a gas giant and also the biggest planet

Saturn

Saturn is a gas giant made of hydrogen and helium

Project Management Infographics

Concept 1

Concept 2

Concept 3

Concept 4

Saturn

Saturn is a gas giant made of hydrogen and helium

Jupiter

Jupiter is a gas giant and the biggest planet of them all

Venus

Venus has a beautiful name, but also high temperatures

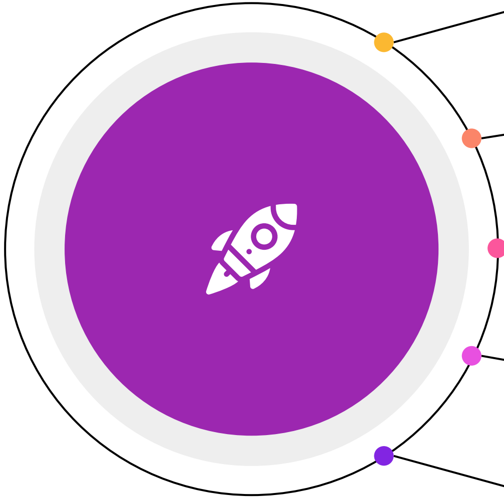
Mercury

Mercury is the smallest planet in the Solar System

Project Management Infographics

STRENGTHS	WEAKNESSES	OPPORTUNITIES	THREATS
Venus	**Mars**	**Jupiter**	**Saturn**
Venus has a beautiful name and high temperatures	Despite being red, Mars is a cold place full of iron oxide dust	Jupiter is the fourth and the biggest planet	Saturn is composed of hydrogen and also helium

Project Management Infographics



Mars — Despite being red, Mars is actually a cold place

Jupiter — Jupiter is a giant and the fourth planet from the Sun

Saturn — Saturn is a gas giant made of hydrogen and helium

Neptune — Neptune is the farthest planet from the Sun

Venus — Venus is the second planet from the Sun

Project Management Infographics

Step 01

Mercury is the closest planet to the Sun

Step 02

Despite being red, Mars is a cold place

Step 03

Jupiter is a gas giant and the biggest planet

Step 04

Neptune is the farthest planet from the Sun

Project management infographics

Saturn

Saturn is a gas giant made of hydrogen and helium

Jupiter

Jupiter is a gas giant and the biggest planet

Venus

Venus is the second planet from the Sun

Mercury

Mercury is the smallest planet in the Solar System

Mars

Despite being red, Mars is actually a cold place

Project management infographics

Saturn

Saturn is a gas giant composed mostly of hydrogen and helium

Mars

Despite being red, Mars is a cold place full of iron oxide dust

Jupiter

Jupiter is a gas giant and the biggest planet in the Solar System

Venus

Venus has a beautiful name, but it is terribly hot there

Mercury

Mercury is the smallest planet in the Solar System

Project Management Infographics

Saturn

Saturn is a gas giant made of hydrogen and helium

Mercury

Mercury is the smallest planet in the Solar System

Jupiter

Jupiter is a gas giant and the biggest planet

Neptune

Neptune is the farthest planet from the Sun

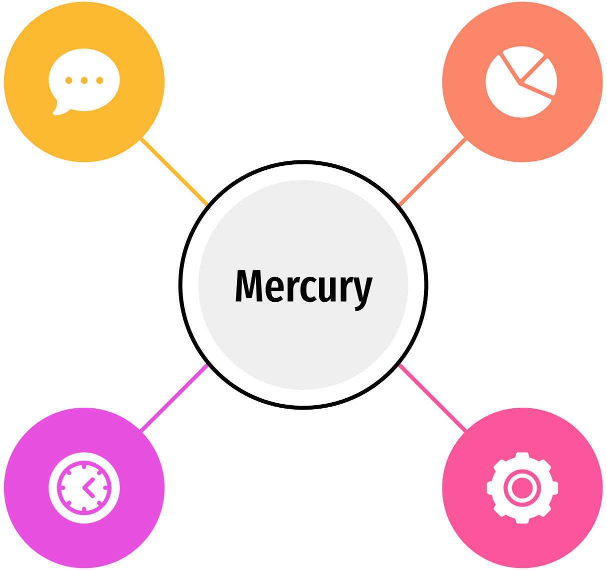
Project Management Infographics

Mercury

Mercury is the closest planet to the Sun and the smallest one

Neptune

Neptune is the farthest planet from the Sun and the fourth-largest



Saturn

Despite being red, Mars is a cold place full of iron oxide dust

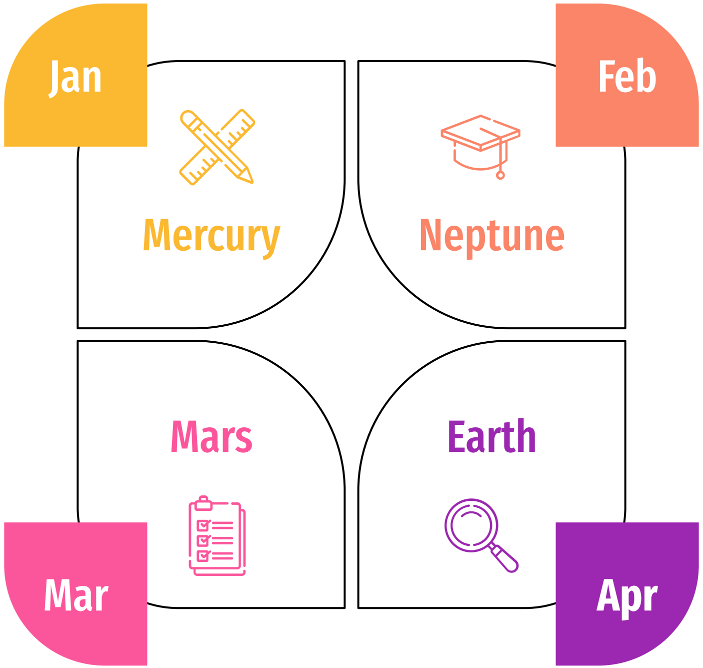
Earth

Earth is the third planet from the Sun and the only one that harbors life

Project Management Infographics

Mercury is the closest planet to the Sun and the smallest one

Neptune is the farthest planet from the Sun and the fourth-largest one



Despite being red, Mars is a cold place full of iron oxide dust

Earth is the third planet from the Sun and the only one that harbors life

Project Management Infographics

Neptune

Neptune is the farthest planet from the Sun and a gas giant

Mercury

Mercury is the closest planet to the Sun and the smallest one

Earth

Earth is the third planet from the Sun and the one that harbors life

Mars

Despite being red, Mars is a cold place full of iron oxide dust

Jupiter

Jupiter is a gas giant and the biggest planet in the Solar System

Project Management Infographics

01

Mercury is the closest planet to the Sun

02

Despite being red, Mars is a cold place

03

Jupiter is a gas giant and the biggest planet

04

Neptune is the farthest planet from the Sun

Project Management Infographics

Neptune

Neptune is the farthest planet from the Sun and a gas giant

Mercury

Mercury is the closest planet to the Sun and the smallest

Earth

Earth is the third planet from the Sun

Mars

Despite being red, Mars is actually a cold place

Project Management Infographics

Why 1	Mercury is the closest to the Sun and the smallest
Why 2	Venus is the second planet from the Sun
Why 3	Despite being red, Mars is a cold place
Why 4	Jupiter is the biggest planet in the Solar System
Why 5	Saturn is composed of hydrogen and helium

Project Management Infographics

Neptune is the farthest planet from the Sun and a gas giant

Despite being red, Mars is a very cold place full of iron oxide dust

Earth is the third planet from the Sun and the only one that harbors life

Jupiter is a gas giant and the biggest planet in the Solar System

Project Management Infographics

Neptune

Neptune is the farthest planet from the Sun and a gas giant

Mercury

Mercury is the closest planet to the Sun

Earth

Earth is the third planet from the Sun

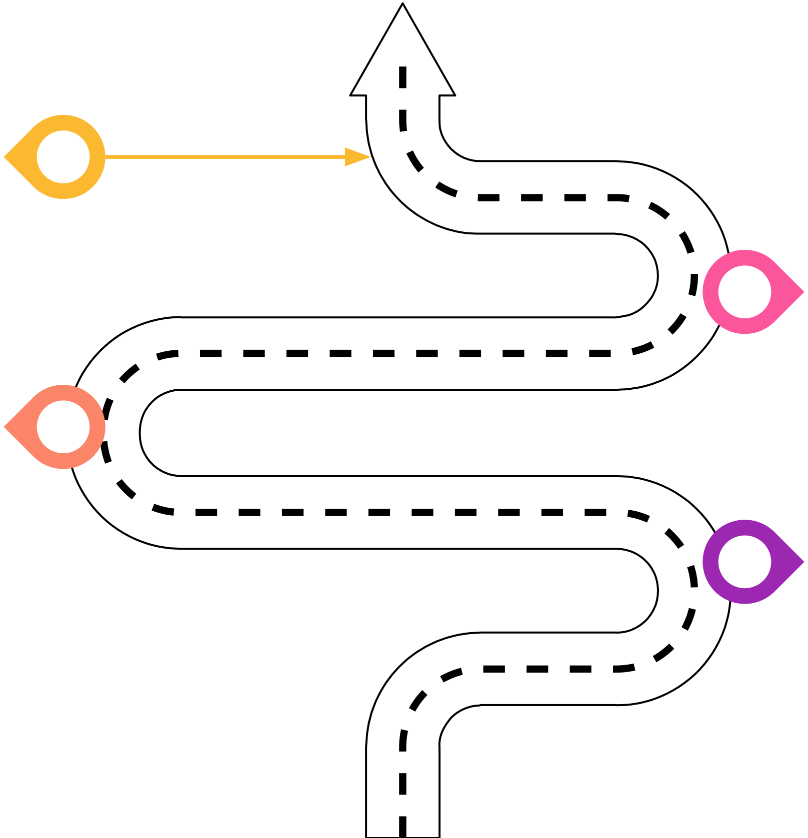
Mars

Despite being red, Mars is actually a cold place

Jupiter

Jupiter is a gas giant and the biggest planet of them all

Project Management Infographics



Neptune

Neptune is the farthest planet from the Sun

Saturn

Saturn is composed of hydrogen and helium

Mars

Despite being red, Mars is a very cold place

Mercury

Mercury is the closest planet to the Sun

Project management infographics

Saturn

Saturn is a gas giant made of hydrogen and helium

Jupiter

Jupiter is a gas giant and the biggest planet

Venus

Venus is the second planet from the Sun

Mercury

Mercury is the smallest planet in the Solar System

Neptune

Neptune is the farthest planet from the Sun

Mars

Despite being red, Mars is actually a cold place

Project Management Infographics

Venus has a beautiful name and is the second planet from the Sun. It’s terribly hot—even hotter than Mercury—and its atmosphere is extremely poisonous. It’s the second-brightest natural object in the night sky after the Moon

Planet Jupiter is a gas giant and also the biggest one in the Solar System

Task 1

Mercury is the closest to the Sun and the smallest one in the Solar System

Task 2

Despite being red, Mars is a very cold place full of iron oxide dust

Task 3

Earth is the third planet from the Sun and the only one that harbors life

Project Management Infographics

Option 1

Mars

Despite being red, Mars is actually a cold place

Option 2

Jupiter

Jupiter is a gas giant and the biggest planet

Option 3

Saturn

Saturn is a gas giant and has several rings

Option 4

Neptune

Neptune is the farthest planet from the Sun

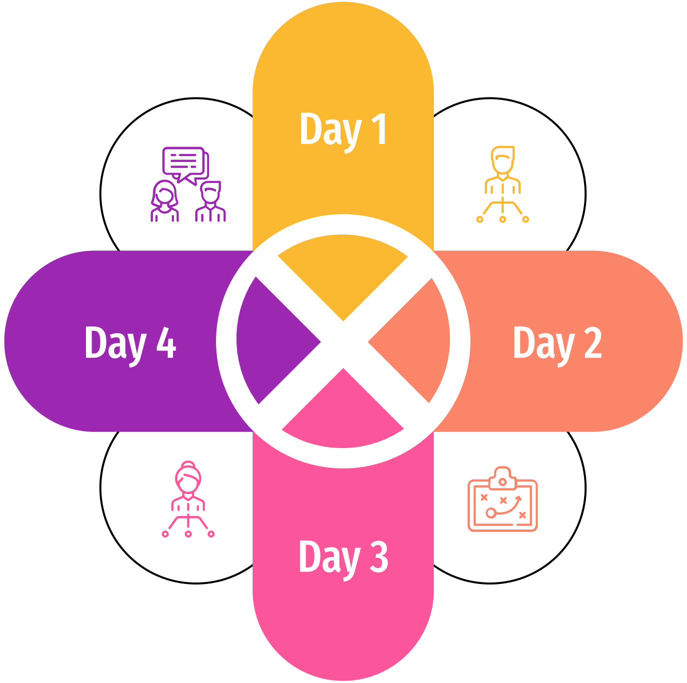
Project Management Infographics

Neptune

Neptune is the farthest planet from the Sun and a gas giant

Earth

Earth is the third planet from the Sun



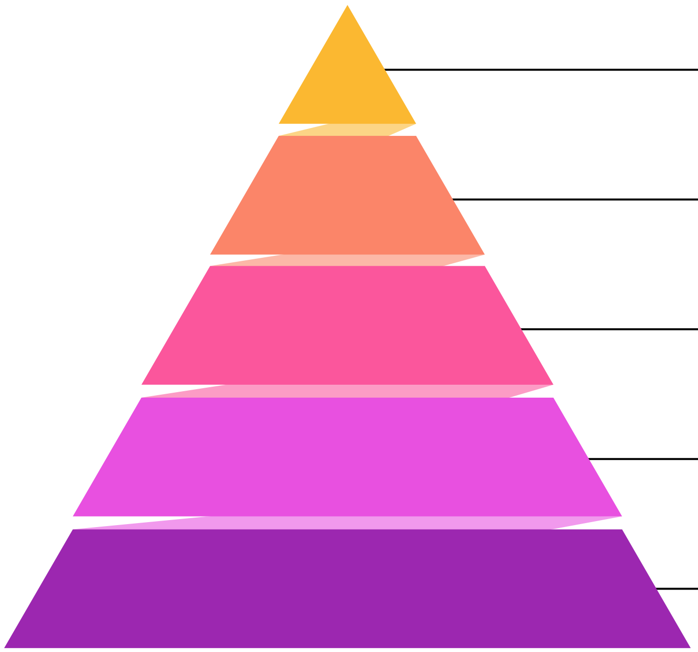
Mercury

Mercury is the closest planet to the Sun

Mars

Despite being red, Mars is actually a cold place

Project Management Infographics



Mercury

Mercury is the smallest planet in the Solar System

Mars

Despite being red, Mars is actually a cold place

Jupiter

Jupiter is a gas giant and the biggest planet of them all

Neptune

Neptune is the farthest planet from the Sun and a gas giant

Earth

Earth is the third planet from the Sun and the one with life

Project Management Infographics

Mars

Despite being red, Mars is actually a cold place

Mercury

Mercury is the smallest planet in the Solar System

2023

Saturn

Saturn is composed mostly of hydrogen and helium

Venus

Venus is the second planet from the Sun

Jupiter

Planet Jupiter is a gas giant and the biggest one

Neptune

Neptune is the farthest planet from the Sun

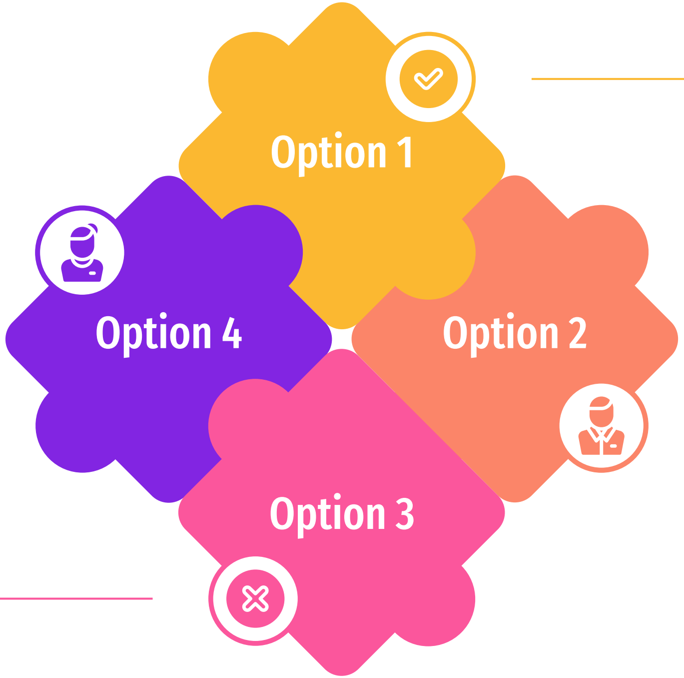
Project Management Infographics

Saturn

Saturn is composed of hydrogen and helium

Mercury

Mercury is the smallest planet in the Solar System



Jupiter

Jupiter is a gas giant and the biggest planet

Neptune

Neptune is the farthest planet from the Sun

Project Management Infographics

Mars

Despite being red, Mars is actually a cold place

Jupiter

Jupiter is a gas giant and also the biggest planet

Saturn

Saturn is a giant made of hydrogen and helium

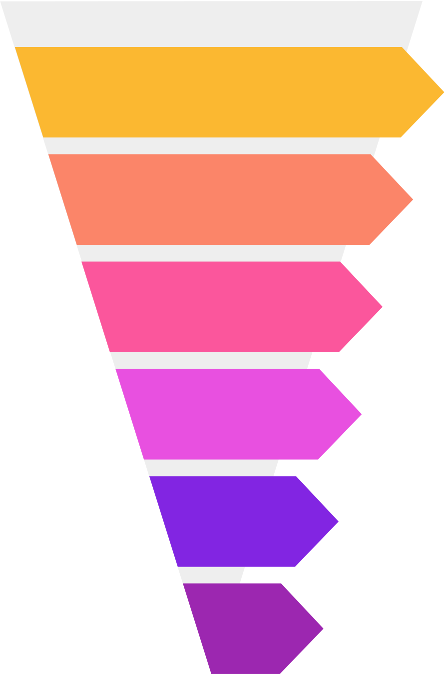
Neptune

Neptune is the farthest planet from the Sun

Venus

Venus is the second planet from the Sun

Project Management Infographics



Mercury	Mercury is the smallest planet of them all
Neptune	Neptune is the farthest planet from the Sun
Saturn	Saturn is composed of hydrogen and helium
Jupiter	Jupiter is a gas giant and the biggest planet
Mars	Despite being red, Mars is a very cold place
Venus	Venus is the second planet from the Sun

Project Management Infographics

01 Saturn

Planet Saturn is a gas giant composed mostly of hydrogen and helium

02 Mercury

Mercury is the smallest planet in the Solar System and the closest to the Sun

03 Mars

Despite being red, Mars is a very cold place full of iron oxide dust

04 Venus

Venus has a beautiful name and is the second planet from the Sun

Instructions for use

In order to use this template, you must credit **Slidesgo** and **Freepik** in your final presentation and include links to both websites.

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

Infographics

You can add and edit some **infographics** to your presentation to show your data in a visual way.

- Choose your favourite infographic and insert it in your presentation using Ctrl C + Ctrl V or Cmd C + Cmd V in Mac.
- Select one of the parts and **ungroup** it by right-clicking and choosing "Ungroup".
- **Change the color** by clicking on the paint bucket.
- Then **resize** the element by clicking and dragging one of the square-shaped points of its bounding box (the cursor should look like a double-headed arrow). Remember to hold Shift while dragging to keep the proportions.
- **Group** the elements again by selecting them, right-clicking and choosing "Group".
- Repeat the steps above with the other parts and when you're done editing, copy the end result and paste it into your presentation.
- Remember to choose the **"Keep source formatting"** option so that it keeps the design. For more info, please visit **Slidesgo School**.