Strand 2

Algorithms

MR GARVEY 2021

Learning Intentions

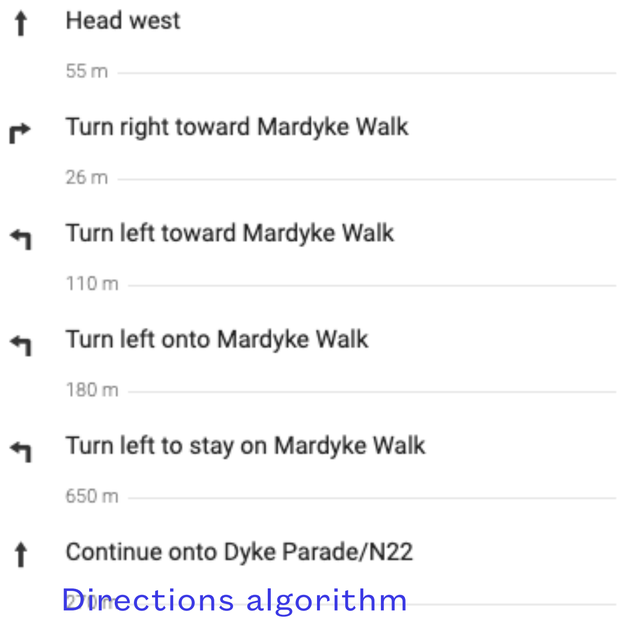
Strand 1

Explain the operation of a variety of algorithms
Develop algorithms to implement chosen solutions
Use pseudo code to outline the functionality of an algorithm
Construct algorithms using appropriate sequences, selections/conditionals, loops and operators to solve a range of problems, to fulfil a specific requirement
Implement algorithms using a programming language to solve a range of problems
Apply basic search and sorting algorithms and describe the limitations and advantages of each algorithm
Explain the common measures of algorithmic efficiency using any algorithms studied

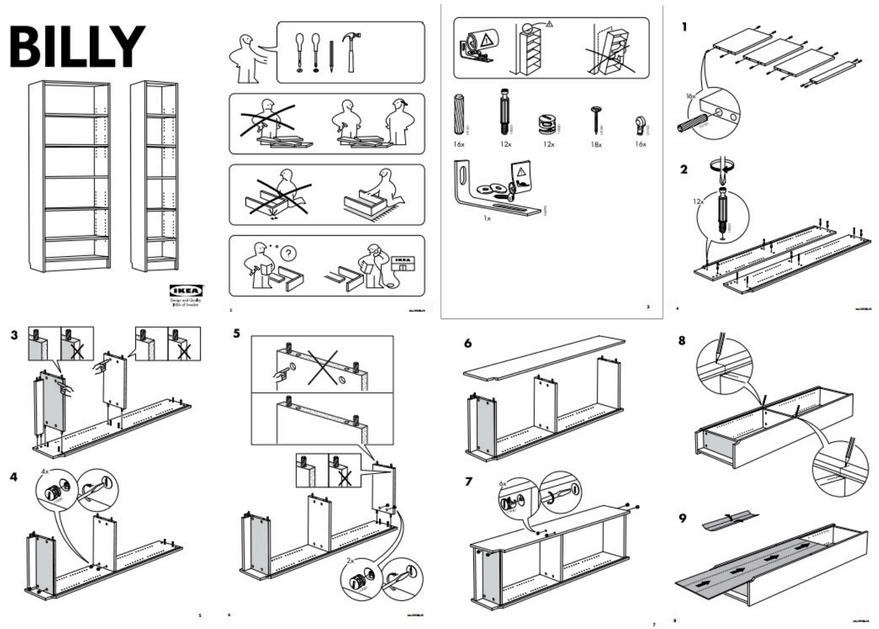
What is an Algorithm?

Explain the operation of a variety of algorithms

- An algorithm is a list of steps that you follow to complete a task.
- An algorithm will always return an answer OR perform a task.
- Forms of algorithms you are already familiar with include: recipes, giving directions, instructions for flat pack furniture.
- An algorithm does NOT have to be written in code. E.g IKEA instructions or Google Map instructions



Directions algorithm



IKEA furniture algorithm

Problems solved by algorithms

Explain the operation of a variety of algorithms

Internet-related algorithms. Algorithms are used to manage and manipulate the huge amount of data stored on the Internet. How does a search engine find all the pages on which particular information resides in a fraction of a second?

Route-finding algorithms. Given two locations, how does a route-finder determine the shortest or best route between the two points? There may be thousands of possible routes. This type of algorithm is used not only for driving a vehicle from A to B, but also for many other applications, for example, finding the best route to transmit packets cf data from A to B ever a network.

Compression algorithms. These are used to compress data files so that they can be transmitted faster or held in a smaller amount of storage space. For example. Image files are compressed so that they can be stored on social media sites.

Encryption algorithms. When someone purchases something over the Internet and sends their credit card number and other personal details to the store, the data needs to be encrypted so that even if it is intercepted, it cannot be read.

Algorithms for computers

Explain the operation of a variety of algorithms

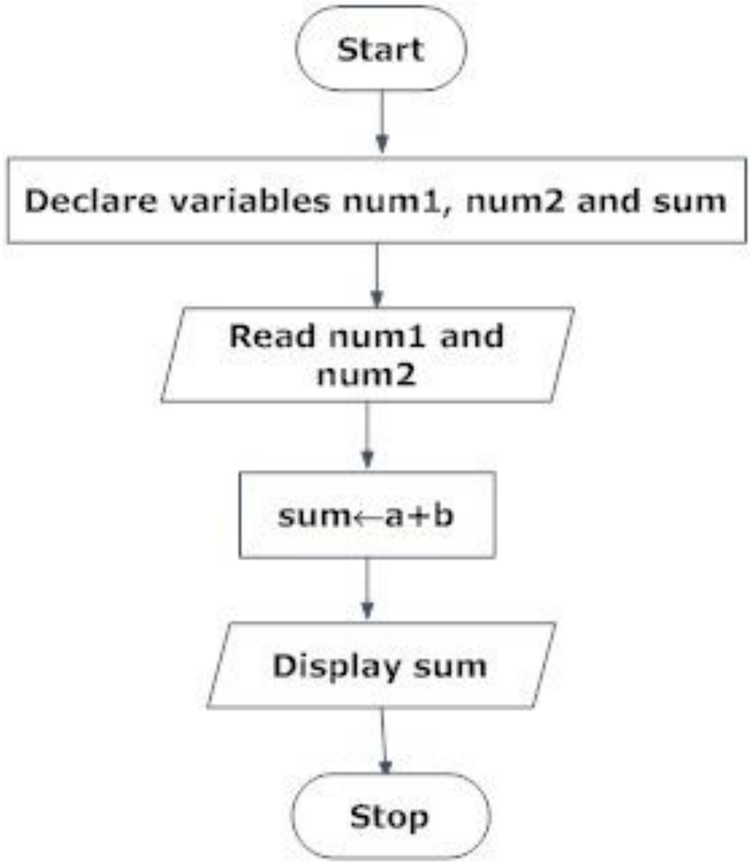
- Algorithms must be unambiguous, this means that every detail and every possibility must be accounted for. Computers cannot guess or rely on previous experience like humans can.
- Algorithms must consist of a finite number of steps.
- Have a clear purpose/flow from the beginning to the end that produce the correct output for any valid inputs.
- Should execute in as few steps as possible, ie. be efficient.
- Should allow for invalid inputs and get an appropriate message.
- Should be easy for people to understand.



Creating an algorithm?

Use pseudocode to outline the functionality of an algorithm

- When designing a solution to a problem we express the solution (algorithm) using steps written in pseudocode or flowcharts.



Flowchart to sum two numbers

```
Num1 =0

num2=0

sum=0

Input ( Value for Num1)

Input (Value for Num 2)

Sum = Num1 + Num2

Output (Sum)
```

Pseudocode to sum two values

Pseudocode notation

Use pseudocode to outline the functionality of an algorithm

- There is no *incorrect* pseudocode, but there is some widely recognised notation.

- **INPUT** – indicates a user will be inputting something
- **OUTPUT** – indicates that an output will appear on the screen
- **WHILE** – a **loop** (**iteration** that has a **condition** at the beginning)
- **FOR** – a counting loop (iteration)
- **IF – THEN – ELSE** – a decision (**selection**) in which a choice is made any instructions that occur inside a selection or iteration are usually indented

IF – single choice with alternative

```
IF MyValue > YourValue
  THEN
    OUTPUT "I win"
  ELSE
    OUTPUT "You win"
ENDIF
```

```
Number ← 0
WHILE Number >= 0 DO
    OUTPUT "Please enter a negative number "
    INPUT Number
ENDWHILE
```

Pseudocode examples

Use pseudocode to outline the functionality of an algorithm

- What processes do you think the following pseudocode examples represent.

A

```
BuyBook(bookname) :
1. Open Amazon
 2. Search for the bookname
 3. If in stock, add to the cart
Else tell user book out of stock
 4. Proceed with checkout process
 5. Wait for your book to be shipped


BuyBook('Into The Water')
```

B

```
x=0
Repeat 10 times
    Ask user to enter a mark
    Accept the mark
    If mark> x then x = mark
Endrepeat
Print x
```

What does algorithm B output if the numbers 14, 7, 16, 12, 10, 8, 12, 9, 11, 8 are entered.

Algorithms in action

Rescue dog suitability

55	67	89	84	34	23

Likelihood of being a successful rescue dog

Use pseudocode to outline the functionality of an algorithm

- Finding the largest value in an array is a common use of algorithms.

INPUT List of values

Max = List[0]

FOR I in range[length of list]

IF LIST[i] > Max

Max==List[i]

OUTPUT ‘Max value is’ Max

Using Python and this pseudocode try and recreate the Python program to find the max value in the list and its position

```
The max value is 89
The dog most likely to be a good rescue dog is at: 2
>>>
```

Pseudocode practice

Use pseudocode to outline the functionality of an algorithm

Write algorithms using pseudocode for the following three tasks.

- Write an algorithm using pseudocode to display the sum of three user inputted values.
- Write an algorithm using pseudocode to add up the numbers from 1 to 100.

Real world practice

- Write pseudocode **and** code for a program which asks the user to enter the total bill for a restaurant meal, and the total number of people who had a meal. The program should add 10% to the bill as a tip and then calculate and display to the nearest penny what each person owes, assuming the bill is evenly split.
- Write the pseudocode **and** code that asks the user for a distance value in miles. You then must covert this distance to kilometres, rounding the answer to the first decimal place.
    - Create your pseudocode before starting any coding.
    - Identify any variables you might use in your program In your pseudocode.

Algorithms in the real world

Have you ever used a search algorithm?

- Any time you have used Google or tried to find someone on Instagram you have used a search algorithm.
- Without search algorithms if you wanted to find data you would have to search an entire datasets one element at a time.
- Google uses a number of algorithms in its search engine. These algorithms look at:
    - Words in the query
    - Relevance
    - Page rank
    - Location

Lists and Arrays

Recall: How do we store multiple values using just one variable in Python?

list = ["abc", 34, True, 40, “male"]

Studentname = [“Martin”, “Joe”, “Shane”]

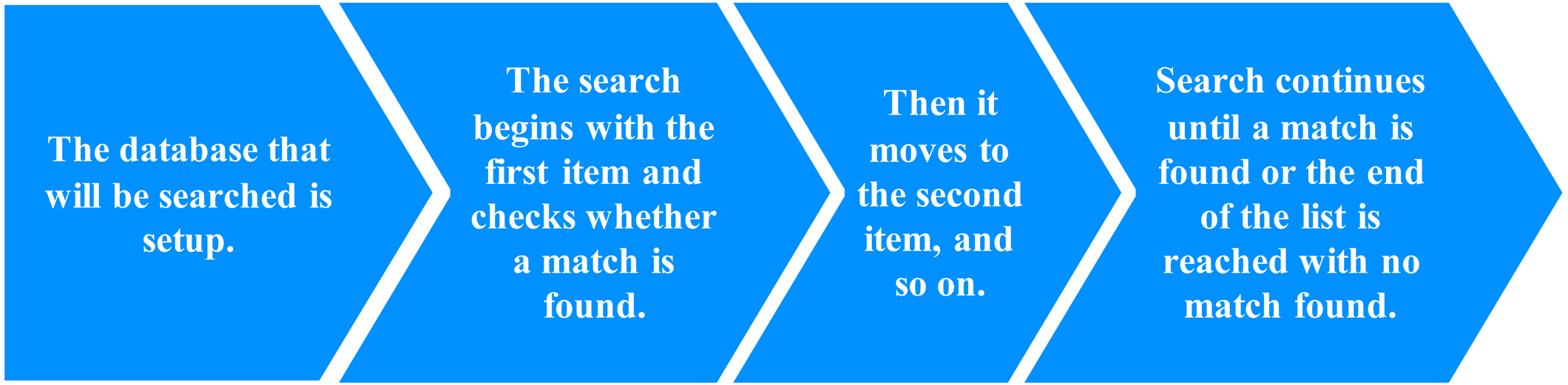
- A list can contain different types of data, (integers, floats, strings), in the same list.
- Each item on the list has an index position that could be referenced.
- The first item in the list is assigned the index position of 0.
- Data is often stored in Arrays, these are similar to lists but only contain elements of the same data type.

Linear search algorithms

Apply basic search algorithms and describe the limitations and advantages of each algorithm

- Search algorithms are used to search for data in lists and arrays.
- We will study two types of search algorithms, Linear Search and Binary Search.

When searching through lists, what effect would having the data in order or not in order have on the length of time to find the required data?

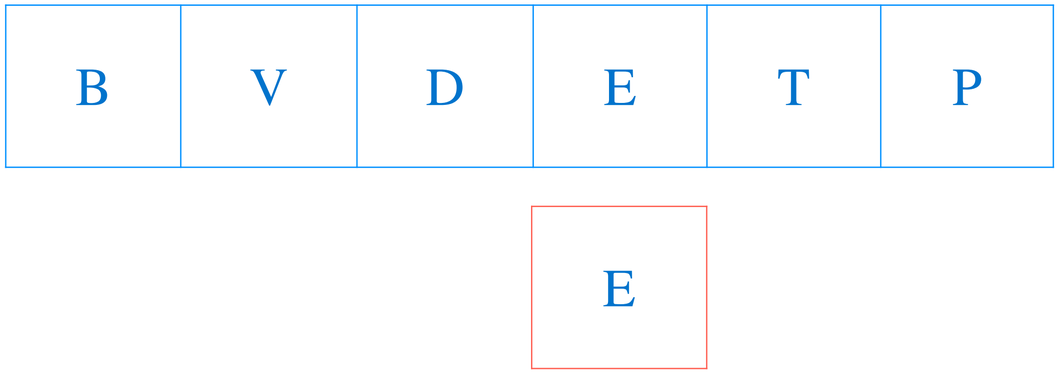


Linear search pseudocode

Linear search in action

Apply basic search algorithms and describe the limitations and advantages of each algorithm

Implement algorithms using a programming language to solve a range of problems



- Let's consider a list with 6 elements.
- We want to find the letter E using our search algorithm.
- In a linear search each item in the list is compared to our search value, one item at a time.
- Linear search works on all data types.
- The list does NOT have to be ordered for a linear search to work.

Linear search pseudocode

Explain the operation of a variety of algorithms

Use pseudocode to outline the functionality of an algorithm

```
Input List
Input( Name to be searched for)

FOR  index in range (Length of list)
    If list[index] == Input
        Item position = index value
        Output position
    Else
        Output Item not in list
```

B	V	D	E	T	P
			position		

E

Index = 3
List[Index] = List3] = E

List[Index] is not equal to the the searched item ‘E’
Position variable is incremented.

Alternative linear search algorithm

Construct algorithms using appropriate sequence to solve a range of problems.

```
Found = False
Counter = 0
Length= Length(array)
Input( Name to be searched for)
While found = False AND Counter <Length DO
        IF Array[Counter]=Input
            Set Found to True
            Output ( Found at position Counter)
        Else
            Set counter TO Counter +1
        END IF
END While
IF found = False
    Output (Item not found in the Array)
END If
```

```
name=["Martin","Steve","Kathy","Paul","Michelle","Linda"]

Found=False
Counter = 0
Lenght=len(name)

User_Input=input("Please enter the name you want to search for: ")

while (Found==False) and Counter<Lenght:
    if name[Counter]==User_Input:
        Found=True
        print(Counter)
    else:
        Counter=Counter+1
if Found==False:
    print("Item not in Array")
```

Homework due 27th Jan

1. In computer science, what term is used to refer to the item that we are searching for?
2. How does a linear search algorithm work?
3. State one way in which a binary search differs from a linear search.
4. What is another name for a binary search?
5. How does a binary search work?
6. When writing a program for a binary search, state two variables whose values we must keep track of.

1. Using the list below, write the pseudocode for a linear search with a key of -1. Then repeat this process with a key of 33.

   [19, 87, 1, -1, 11, 0, 4, 33, 19]

2. Write the pseudocode for a binary search using the list above. First search for -1 and then repeat this process searching for 33.

3. Explain the differences between the linear search and binary search algorithms.

4. Given the following list:

   [19, 87, 1, -1, 11, 0, -1, 33, 19]

   a) Create a linear search in Python and test it to see the outcome when searching for a key of -1, then 33, then 117. What are the outcomes?

   b)

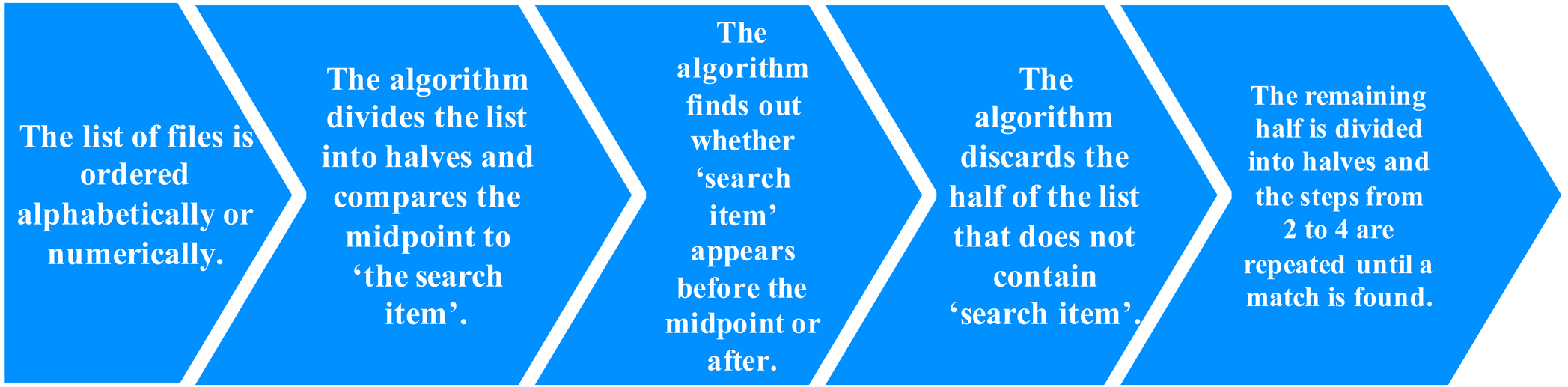
   c) Modify the linear search again to display -1 as the index, if the key is not found, i.e., 117.

Binary search algorithms

Apply basic search algorithms and describe the limitations and advantages of each algorithm

- Binary search differ from Linear search algorithms as they require the data to be already sorted.
- If the data is not sorted, a linear search is the only option.
- It works by repeatedly dividing in half the portion of the data list that may contain the required data item

Do you think a linear or binary search would be faster to search a large sorted database?



Binary search pseudocode

Binary search in action

Apply basic search algorithms and describe the limitations and advantages of each algorithm

G

Position	0	1	2	3	4	5	6	7	8
Element	A	B	C	D	E	F	G	H	I

- Let us consider an ordered list with a certain length.
- first_index is the position of first element and last_index is the position of last element.
- In the table below, a list of 10 elements is shown.
- Here the first_index is 0, last_index is 8 and length_of_list is 9.

Binary search in action

Apply basic search algorithms and describe the limitations and advantages of each algorithm

G

Position	0	1	2	3	4	5	6	7	8
Element	A	B	C	D	E	F	G	H	I

1. Identify the first and last index positions.
2. **midpoint** = (first_index+last_index) /2 = (0+8)/2 = **4**
3. Compare list [midpoint] value with search key = list [4]= E
4. Because list[midpoint] < item, (Alphabetically E is before G), the first_index position changes to become midpoint +1
5. All items before the midpoint are now ignored.
6. Because E < G, the first_index value becomes (midpoint + 1) = 4+1 = 5

Binary search in action

Apply basic search algorithms and describe the limitations and advantages of each algorithm

G

5	6	7	8
F	G	H	I

6. midpoint = round ((5+8) //2) = 7
7. list [midpoint]= list [7]= H
8. Because list[midpoint] > item, the last_index position changes to become midpoint - 1
9. Because H > G, the last_index value becomes midpoint -1 = 7 - 1 = 6
10. Midpoint = our search value
11. Output position ‘6’

Binary search pseudocode

Apply basic search algorithms and describe the limitations and advantages of each algorithm

```
# Pseudocode
# Declare List
# Declare variable 'found'
# Sort the list if it is not sorted
# Input key value
# While loop
#     Declare first and last index position of list
#     While loop
#         Calculate the midpoint (first_index+last_index)/2
#         Compare midpoint element with key
#         If midpoint < key
#             New first index position becomes Midpoint+1
#         Elif midpoint > key
#             New first index position becomes Midpoint-1
#         Elif midpoint == key
#             Tell user index position
#             change found variable to True
#     End WHile
# End While
```

```
myList=['A','B','C','D','E','F','G','H','I']
myList.sort()

print(myList)
key=input("Enter key value to search for: ")
FOUND=False

while FOUND==False:
    first_index_value=0
    last_index_value=(len(myList)-1)

# **** Start of binary search algorithm*****

    while first_index_value<=last_index_value:
        mid_point=(last_index_value+first_index_value)//2
        print("Mid_point value is:",mid_point)
        if myList[mid_point]<key:
            first_index_value=mid_point+1
        elif myList[mid_point]>key:
            last_index_value=mid_point-1
        elif myList[mid_point]==key:
            print("Value is at position: ",mid_point)
            FOUND==True
            break

# ***** End of binary search algorithm ******
```

Search algorithms

Apply basic search algorithms and describe the limitations and advantages of each algorithm

We have the following sorted list of 10 items:

3	5	6	8	11	12	14	15	17	18

What kind of search algorithm will we use to search this list?

What is the correct sequence of comparisons when the algorithm is used to find the data item ‘8’?

(i) 12,6,8

(ii) 11,5,6,8

(iii) 3,5,6,8

(iv) 11,6,5,8

Ask your partner to think of a number between 1 and 1000. Use a binary search algorithm to guess the number. How many different guesses will you need, at most?

Search algorithms

Apply basic search algorithms and describe the limitations and advantages of each algorithm

Look at the following data list.

9	11	19	22	27	30	32	33	40	42	50	54	57	61	70	78	85

Would you use a (a) linear search or (b) a binary search to find the following data in the list above:

(i) 27

(ii) 11

(iii) 85

Why could you NOT use a binary search algorithm on the following data set

100	10	1	200	60	80

Binary search practice

Apply basic search algorithms and describe the limitations and advantages of each algorithm

An array contains 12 numbers 5, 13, 16, 19, 26, 35, 37, 57, 86, 90, 93, 98

Trace through the binary search algorithm to find how many items have to be examined before the number 90 is found. The first row of the trace table is filled in below.

itemSought
90

found	first	upper	midpoint	aList(midpoint)
false	0	11	5	35

Number 0 , 11 and 5 represent index positions and not actual numbers in the list

Sorting algorithms

Apply basic sort algorithms and describe the limitations and advantages of each algorithm

- A sorting algorithm is a method for reorganising a large number of items into a specific order, such as alphabetical, highest-to-lowest value or shortest-to-longest distance.
- Sorting algorithms take lists of items as input data, perform specific operations on those lists and deliver ordered arrays as output.

Recall how data needs to be arranged for linear and binary search algorithms, when might a sorting algorithm be used WITH a search algorithm?

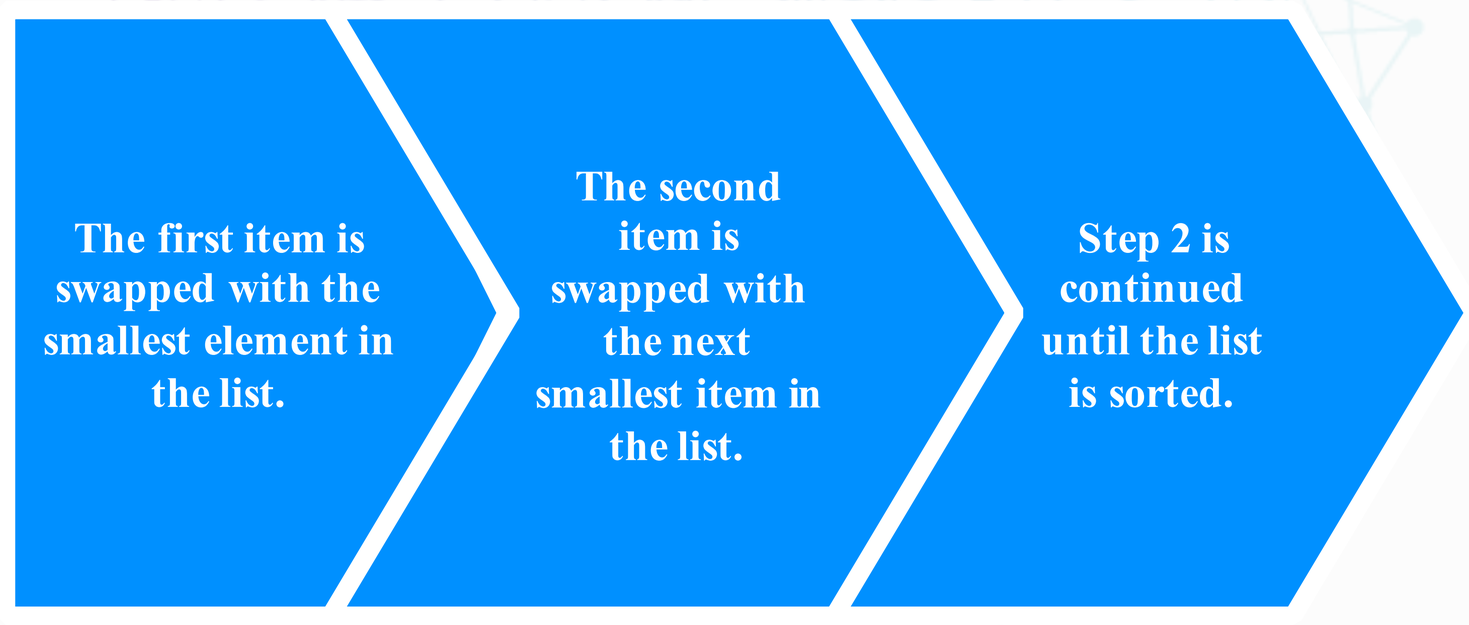
- There are loads of different sorting implementations and applications you can use to make your code more efficient.

Why do we need to study sorting algorithms if python already has a sort() method?

Selection Sort

Apply basic sort algorithms and describe the limitations and advantages of each algorithm

- A selection sort algorithm works by finding the smallest value in a list and then placing it at the start of the list, it then finds the next smallest element and places it in the second position of the list, and then repeats for each value in the list.



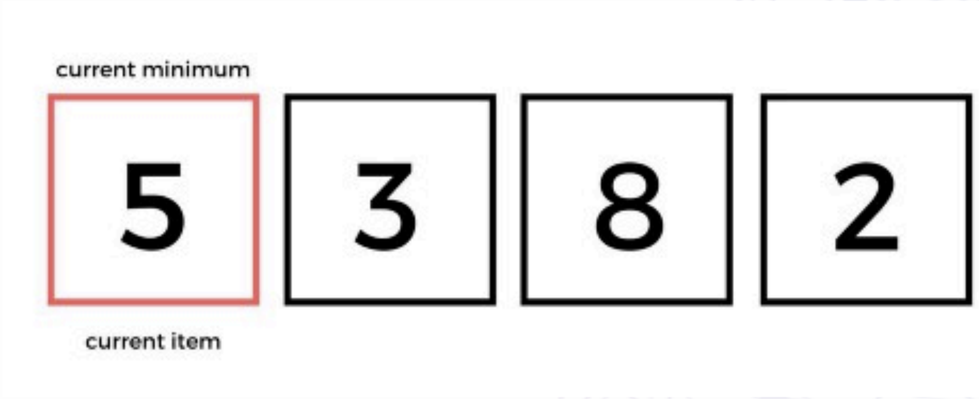
Selection Sort

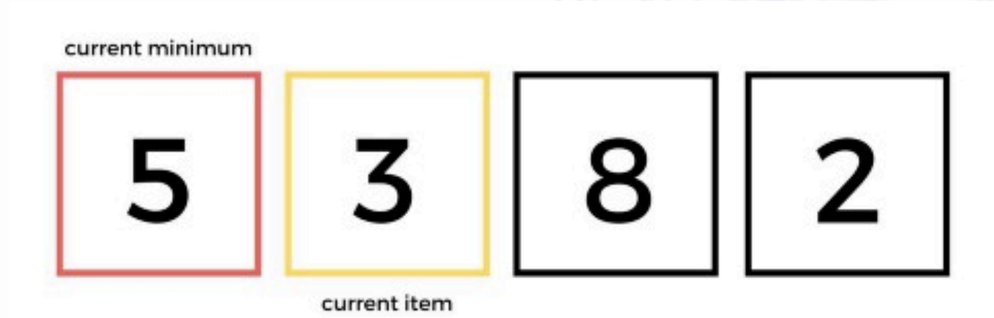
Apply basic sort algorithms and describe the limitations and advantages of each algorithm

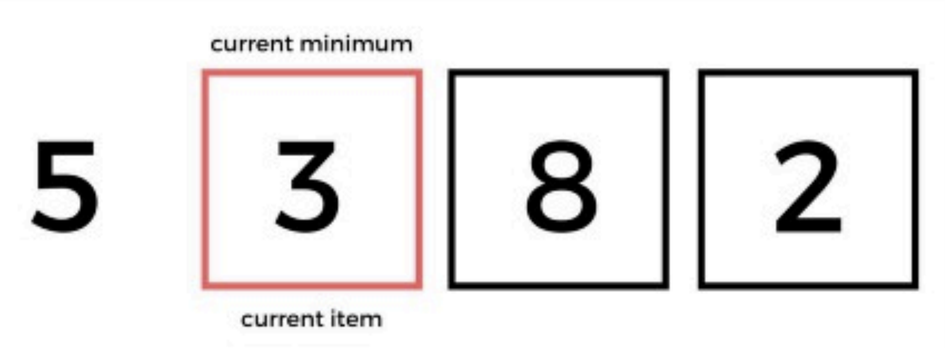
- We use selection sort when we have a small list, the larger the list the longer it takes to sort which makes it inefficient
- We use selection sort if memory space is limited as the algorithm takes the minimum number of swaps so it uses less memory during its operation.
- It is a very simple sorting algorithm
- Selection sort would not be used in many applications and is used more to confirm a list is already sorted.

Selection Sort

Apply basic sort algorithms and describe the limitations and advantages of each algorithm







- We are always dealing with two values, the **current minimum value** and the **current item** being compared to the current minimum value.
- In our code we always initially assign the value at mylist[0] as our minimum value.
- The current item moves through the array comparing itself with our current minimum.
- If the current item is smaller than the current minimum, it becomes the new current minimum.
- This happens all the way to the end of the array, then the current minimum gets put at the start of the array.

Selection Sort Code

Apply basic sort algorithms and describe the limitations and advantages of each algorithm

Input list

Assign value at first index position as the smallest value

Compare smallest value with the next item in the list

if the next item in the list is smaller make this item the smallest item

Compare each element in the list to find the smallest item

Place the smallest item as the first item in the list

Set the second item in the list as the smallest and repeat

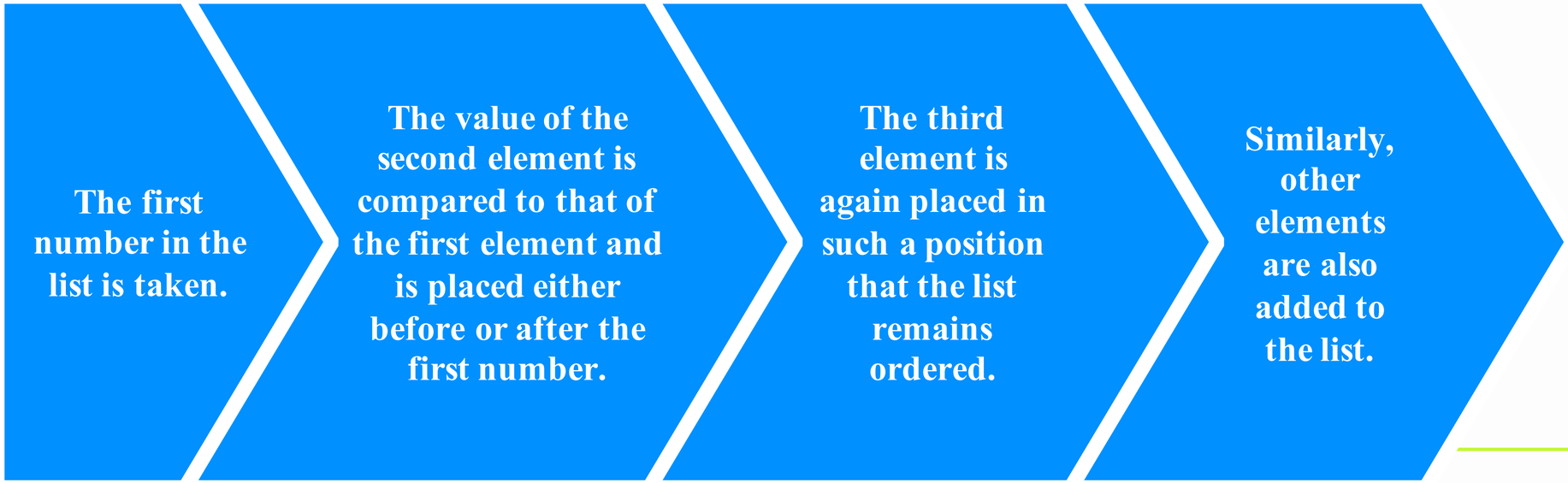
Insertion Sort

Apply basic sort algorithms and describe the limitations and advantages of each algorithm

Imagine you have a number of unsorted playing cards, how would you sort them in order of increasing value?

- A insertion sort algorithm is a simple sorting algorithm that **virtually** splits an array or list into two sub arrays, a sorted and unsorted sub array.

- The current value is compared to the item before it and then sorted accordingly. The next item in the list is then compared to the previous values and sorted accordingly.



Insertion Sort

Apply basic sort algorithms and describe the limitations and advantages of each algorithm

- Insertion sorts are used for short lists and partially sorted lists.
- Can be used for incremental sorting where elements are added to a list one at a time over an extended period
- It is a very slow but very efficient.
- Normally it is used with in combination with Quick sort or Merge sort algorithms, as these algorithms have difficulty dealing with small lists or arrays.

A dataset from Kaggle shows the top Korean Idol pop stars, however the Korean Idols full names are not organised in alphabetical order so we can't find the information about the Idol we want.

Using an insertion sort arrange the Idols full names in alphabetical order, you do not care about the order of any other information given.

When you have your names sorted you will find there are a number of blank values at the start, adjust your code to ignore these data points.

A dataset from Kaggle shows 20,000 reviews left for hotels on Trip Advisor. We want to sort these star ratings and analyse what star rating is the most common.

Open the csv file using python.

Append the values in the column ‘Rating’ to a an appropriately named list.

Process the data to convert it from a string to an int.

Modify your insertion sort code to sort the data.

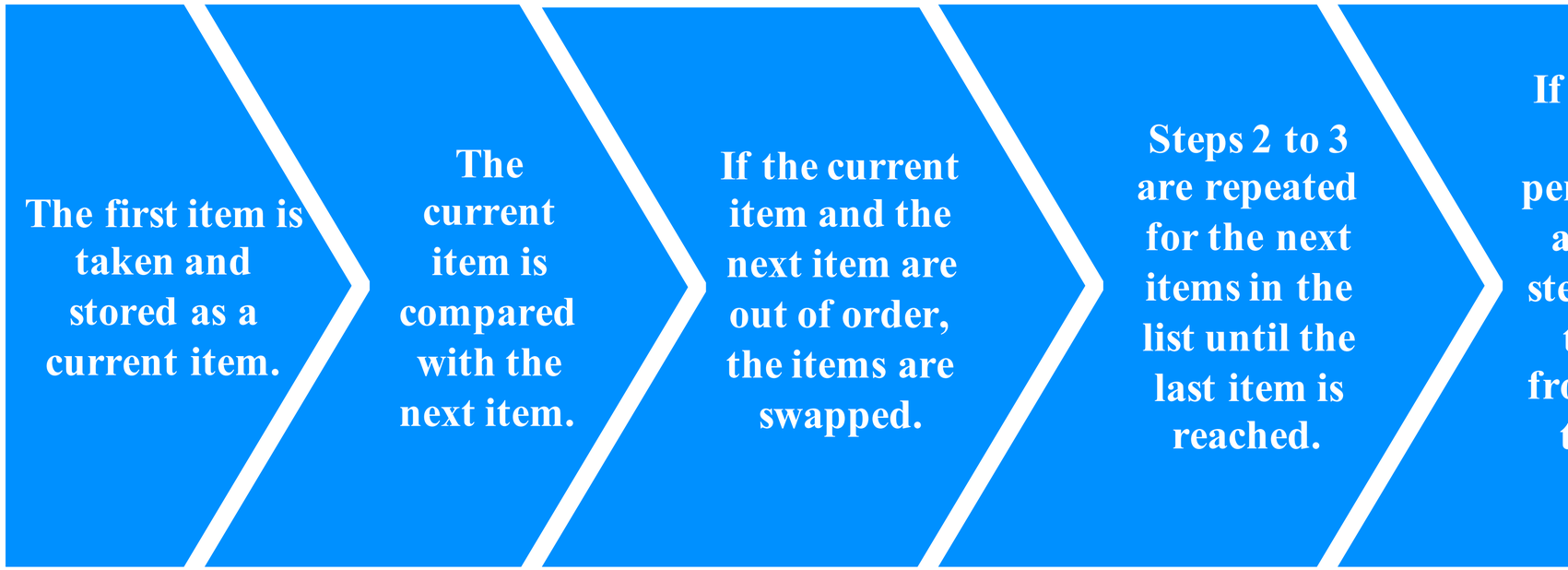
Count how many times each star (1-5) appears in the data set

What conclusion can you draw from this analysis

Bubble Sort

Apply basic sort algorithms and describe the limitations and advantages of each algorithm

- Bubble sort is one f the most basic sorting algorithms and probably the easier to understand.
- In a Bubble sort algorithm, each element in an array is compared with the next element and swapped if necessary.
- It is a very slow and inefficient method of sorting data and is almost never used because of this.



3, 2, 4, 1, 5

The first loop of the algorithm would produce:

- **3**, **2**, 4, 1, 5 (2<3 so the two values are swapped)
- 2, **3**, **4**, 1, 5 (3<4 so the two values are **not** swapped)
- 2, 3, **4**, **1**, 5 (1<4 so the two values are swapped)
- 2, 3, 1, **4**, **5** (4<5 so the two values are **not** swapped)
- 2, 3, 1, 4, 5 (**First pass completed**)

Values were swapped so the algorithm needs to run again. The second loop of the algorithm would start with the final list and run again as follows:

- **2**, **3**, 1, 4, 5 (2<3 so the values are **not** swapped)
- 2, **3**, **1**, 4, 5 (1<3 so the values are swapped)
- 2, 1, **3**, **4**, 5 (3<4 so the values are **not** swapped)
- 2, 1, 3, **4**, **5** (4<5 so the values are **not** swapped)
- 2, 1, 3, 4, 5 (**Second pass completed**)

- **2**, **1**, 3, 4, 5 (1<2 so the values are swapped)
- 1, **2**, **3**, 4, 5 (2<3 so the values are **not** swapped)
- 1, 2, **3**, **4**, 5 (3<4 so the values are **not** swapped)
- 1, 2, 3, **4**, **5** (4<5 so the values are **not** swapped)
- 1, 2, 3, 4, 5 (**Third pass completed**)

Bubble Sort pseudocode

- Find length of the list
- For len(mylist) compare elements of the lists
- Compare values at index position 0 and 1
- Is index[0] greater than index[1]? If so, swap the two numbers around. If not, do not swap.
- Compare the next two index positions.
- Swap elements if necessary.
- Repeat until list is sorted.

Use your pseudocode to create a bubble sort algorithm.

Algorithm efficiency

Explain the common measures of algorithmic efficiency using any algorithms studied

- We have looked at three different sorting algorithms so far and all of them can be used to solved the same problem, e.g sorting the Korean Pop Idol star names in alphabetical order.
- However, how would we know which algorithm is the best algorithm to use?
- A key factor in comparing algorithms is efficiency.
- Efficiency is the amount of work done to achieve a specified task.

We have to be careful how we measure efficiency, our natural reaction is to measure how quick an action is completed, however, using a human measure of time can lead to problems.

Algorithm efficiency

Explain the common measures of algorithmic efficiency using any algorithms studied

- A key factor in comparing algorithms is efficiency.
- Efficiency is the amount of work done to achieve a specified task.
- Running the same algorithm on different computers can lead to different values being returned.
- This can be due to different factors such as:
    - Processor load
    - Availability of free memory
    - Tasks being run in the background
- When comparing algorithms we need to look at its complexity
- By comparing algorithm complexity we can decide which algorithm is the best for our scenario.

Algorithm complexity

Explain the common measures of algorithmic efficiency using any algorithms studied

- Complexity can be measured in two ways:
  - Time complexity - Which looks at how many steps an algorithm will take with a given input ’n’.
  - Space - Which measures how much memory is taken up during execution.

```
14  import random
15  l = []
16
17  for i in range (1,30):
18      l.append (random.randint (1,30000))
19  m = l[0]
20
21  for current in l:
22      if current < m:
23          m = current
24
25  print(l)
26  print(m)
```

Why would simply counting the number of lines of code not be a good way to measure time complexity?

- Each line would have to be translated to machine code and therefore be dependent on factors such as efficiency of the compiler, the language, the CPU.
- Loops will iterate code a number of times that isn’t counted in the number of lines of code in a program.

Counting operations

Explain the common measures of algorithmic efficiency using any algorithms studied

If we can't use 'clock time' as a metric to time complexity, then how do we compare the algorithmic complexity of two programs in a standardised manner?

```
def add_up_to_n_1(x):
    return (x*(x+1)/2)

print(add_up_to_n_1(5))
```

```
def add_up_to_n_2(x):
    total=0
    for i in range (1,x+1):
        total+=i
    return(total)

print(add_up_to_n_2(5))
```

- Both algorithms above calculate the sum of all the numbers up to 5.

Which algorithm do you think is more efficient and why?

What would happen to the number of operations that will be carried out by each algorithm if we want to find the sum of all the values up to 100, 1000, 1 million!

We need a way of technically talking about the efficiency of these algorithms, I can't say things like 1 is good and 2 is meh This is where something called bigO notation comes in..

Big (O) Notation

Explain the common measures of algorithmic efficiency using any algorithms studied

- We can express algorithmic complexity using something called the ‘Big O notation’.
- Big(O) notation defines the relationship between a number of inputs and the steps taken by an algorithm to process those inputs.
- Big(O) is NOT about measuring speed, it is about **measuring the amount of work** a program has to do based on the size of an input of size N.
- Big O is written in the form O(N), which reads as ‘Order of N’ or for short ‘O of N’.

```
def add_up_to_n_1(x):
    return (x*(x+1)/2)

print(add_up_to_n_1(5))
```

- Think back to this algorithm, no matter what the value of N is, only three operations will be carried out.
- We say this algorithm has a constant time as it will always take the same length of time to carry out the operation.
- We would say this algorithm has an efficiency of O(1), Order of 1.

Time complexity - Big O

Explain the common measures of algorithmic efficiency using any algorithms studied

- Big O always refers to the **worst case scenario,** which means it tells you the maximum amount of steps an algorithm will take to give you an output.

It may not be apparent straight away, but O(n) is a function. In Maths you normally see functions written as f(x) or g(x). We will look at O(n) as a linear, quadratic or constant function.

- Algorithm speed isn't measured in seconds it is measured in growth of the number of operations.

- Run time of algorithms is expressed using Big O notation
  - Constant time O(1) (Simple instructions)
  - Linear O(N) (Linear Search)
  - Polynomial O(N^x) (Algorithms with nested loops, eg selection sort)
  - Logarithmic O(logN) (Binary Search)

Constant Time - O(1)

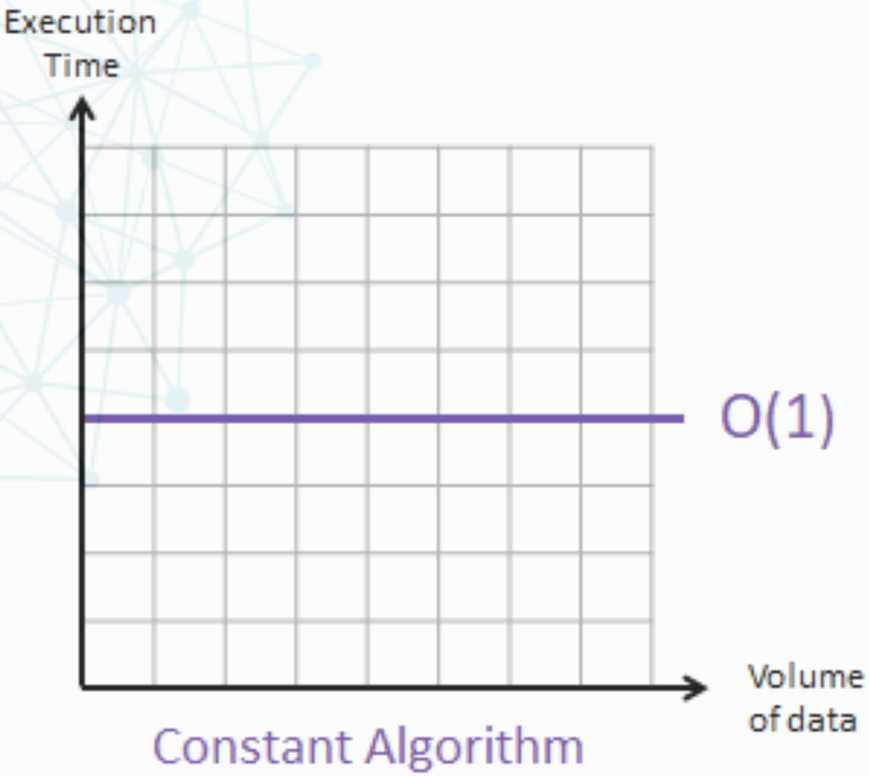
Explain the common measures of algorithmic efficiency using any algorithms studied

- Constant time algorithms will always take the same amount of time to be executed.
- The execution time/number of operations is independent of the size of the input

```
def add_up_to_n_1(x):
    return (x*(x+1)/2)

print(add_up_to_n_1(5))
```

- Examples of constant time operations are:
  - Assigning a value to a variable
  - Inserting an element into a list
  - Retrieving element[i] from a list



Linear time complexity - O(n)

Explain the common measures of algorithmic efficiency using any algorithms studied

- Consider an algorithm to add together two six digit numbers:

```
  1 1 3 4 5 6
+ 7 8 6 0 1 2
```

How would we do this sum? How many individual additions would we have to do get the final total?

- What if we had to add together two ten digit numbers, how many individual additions would we have to do then?

```
  1 2 1 4 5 6 7 8 9 0
+ 6 7 8 0 1 2 2 1 0 6
```

Have you noticed a pattern? What if we were to add two numbers that are both made up of a thousand digits?

Linear time complexity

Explain the common measures of algorithmic efficiency using any algorithms studied

- In the addition example, the complexity (number of operations) is directly proportional to the number of digits in the number. i.e if the number of digits increases the number of operations increases.

- The complexity of an algorithm is said to be linear if the steps required to complete the execution of an algorithm increase or decrease linearly with the number of inputs. Linear complexity is denoted by O(n).

Quadratic time complexity - $O(n^2)$

Explain the common measures of algorithmic efficiency using any algorithms studied

```
1  mylist=[1,2,3,4,5,6]
2  for x in mylist:
3      for y in mylist:
4          print(x,y)
5
6  # What would be the output from this code?
7  # If 'n' is the size of the input, (6) in this case,
8  # what is the relationship between the number of outputs
9  # and 'n' value
```

What if an algorithm is $O(n^2)$?

The number of operations it performs scales in proportion to the *square* of the input.

How does this compare to an algorithm with a linear time complexity O(n)?

If the size of an input doubles, what will happen to the number of operations in an algorithm with linear time complexity O(n), and quadratic complexity $O(n^2)$?

Quadratic time complexity - O(n^2)

Explain the common measures of algorithmic efficiency using any algorithms studied

- Quadratic time complexity algorithms appear when an algorithm contains nested for loops.
- Insertion and Bubble sort algorithms are both algorithms with a big O notation of O(n^2)

```
myList=[107,24,-3,5,85,32,-99,50]

# Selection sort algorithm start
for outerloopindex in range (len(myList)):

    current_min = outerloopindex

    for innerloopindex in range(outerloopindex+1,len(myList)):

        if myList[current_min] > myList[innerloopindex]:
            current_min=innerloopindex

    # Swapping element position
    myList[outerloopindex],myList[current_min]=myList[current_min],myList[outerloopindex]
# Selection sort algorithm end
```

Quadratic time complexity - $O(n^2)$

Explain the common measures of algorithmic efficiency using any algorithms studied

```
myList=[107,24,-3,5,85,32,-99,50]

# Selection sort algorithm start
for outerloopindex in range (len(myList)):

    current_min = outerloopindex

    for innerloopindex in range(outerloopindex+1,len(myList)):

        if myList[current_min] > myList[innerloopindex]:
            current_min=innerloopindex

    # Swapping element position
    myList[outerloopindex],myList[current_min]=myList[current_min],myList[outerloopindex]
# Selection sort algorithm end
```

How many operations are performed?

- Our outer loop performs *n* iterations.
- Our inner loop also performs *n* iterations, but it performs *n* iterations for every iteration of the outer loop.

Quadratic time complexity - $O(n^2)$

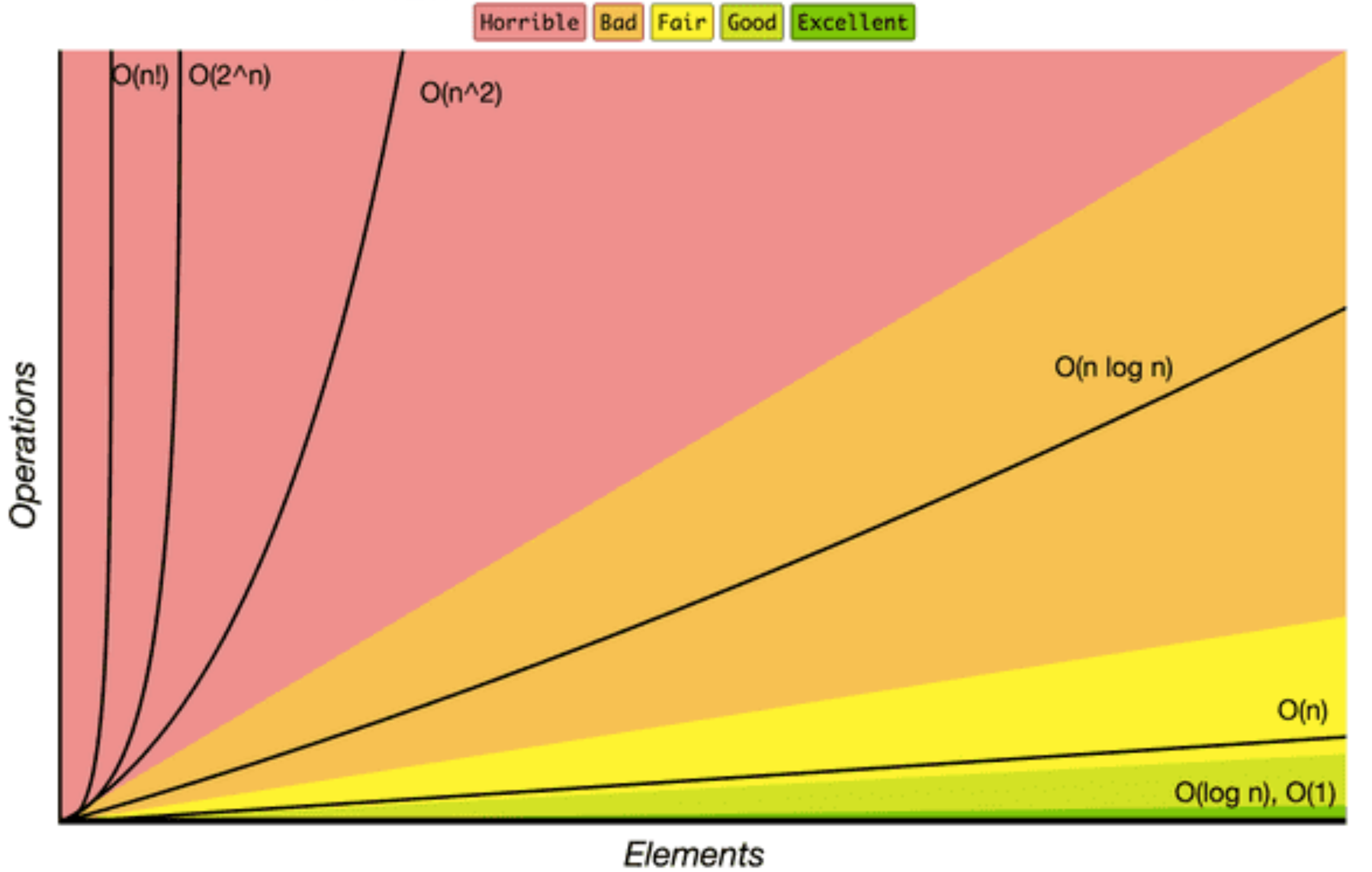
Explain the common measures of algorithmic efficiency using any algorithms studied

Big O	# operations for 10 elements	# operations for 100 elements	# operations for 1000 elements
O(1)	1	1	1
O(N)	10	100	1000
O(N^2)	100	10000	1000000

- As we can see $O(n^2)$ isn't great and gets progressively worse as the number of inputs increases.

Quadratic time complexity - $O(n^2)$

Explain the common measures of algorithmic efficiency using any algorithms studied



How do we know which to use?

Explain the common measures of algorithmic efficiency using any algorithms studied

```
1  #Pseudocode for algorithm with all three complexities.
2
3  if condition_is_met == 'quadratic':
4      for outerloop in range len(list):
5          for innerloop in range len(list):
6              sum=sum+innerloop
7
8  elif condition_is_met == 'linear':
9      for counter in range len(list):
10         sum=sum+counter
11
12 elif condition_is_met== 'constant':
13     sum=(number*(number+1)/2)
```

- The pseudocode above is for an algorithm that outputs to the user the sum of values in a list.
- Which Big O notation will best describe it? Constant O(1), linear O(n) or quadratic $O(n^2)$?
- When considering time complexity we always only consider the dominant term.
- The dominant term is the term that has the most operations, we say it grows the fastest.