technoteach

technocamps

Llywodraeth Cymru
Welsh Government

Prifysgol Abertawe
Swansea University

Cardiff University
Prifysgol Caerdydd

Prifysgol Bangor University

Cardiff Metropolitan University
Prifysgol Metropolitan Caerdydd

University of South Wales
Prifysgol De Cymru

Cyngor Cyllido Addysg Uwch Cymru
Higher Education Funding Council for Wales
hefcw

Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Prifysgol Aberystwyth University

Prifysgol Glyndŵr Wrecsam
Wrexham Glyndŵr University

institute of CODING in wales
technocamps

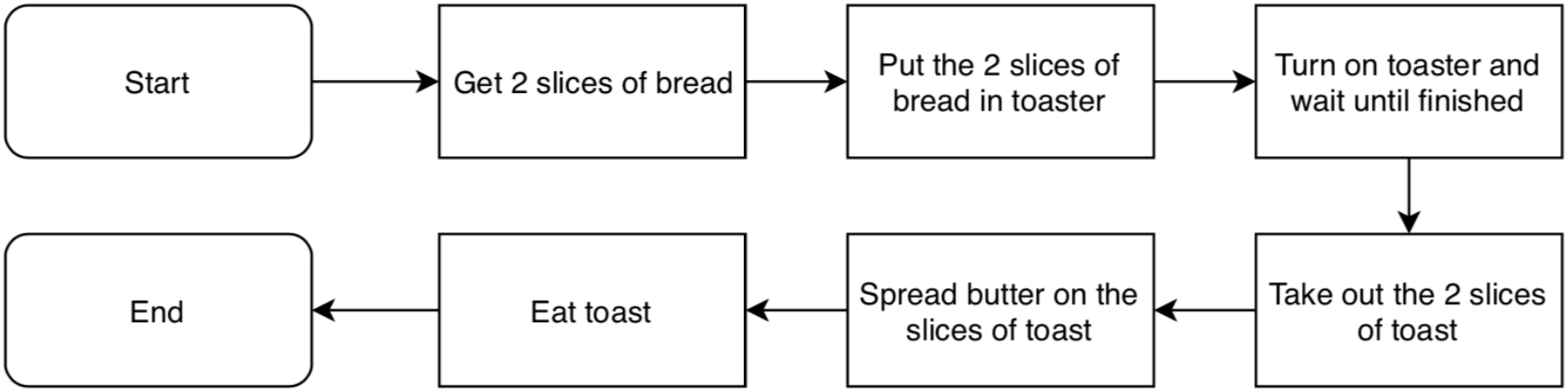
Algorithms

Activity: What is an Algorithm?

Algorithms

Algorithms are sets of simple instructions that are done in a certain order to solve a problem.

We use algorithms all the time in everyday life. An example is making and eating toast.

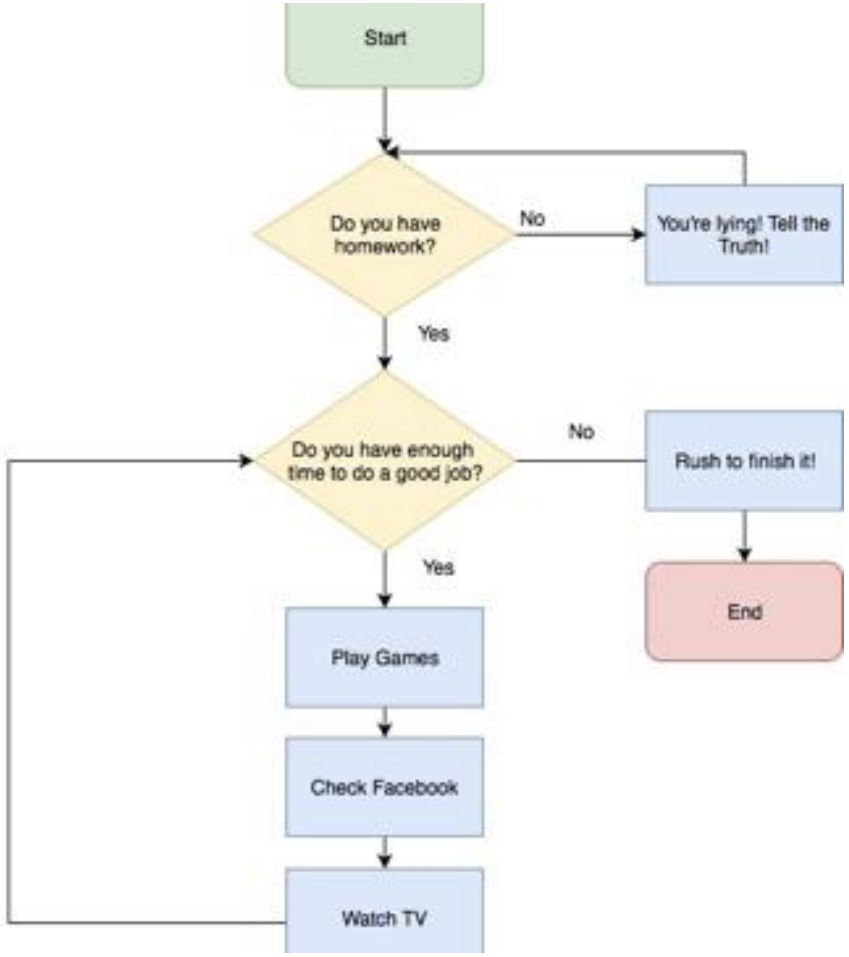


Algorithms

It is important to remember when writing an algorithm to keep instructions:

- Simple
- In the correct order
- Unambigious
- Relevant to solving the problem at hand

Where do we use algorithms in everyday life?

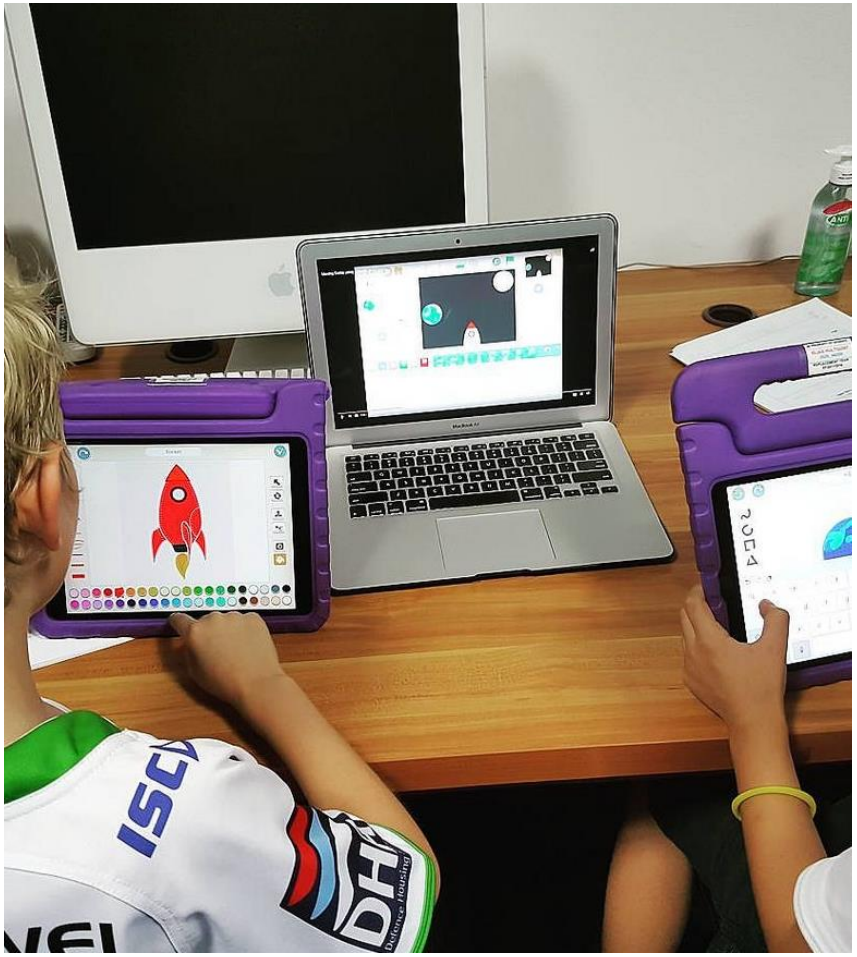


Recap Decomposition

Decomposition is the process of breaking a complex problem down into smaller component parts.

Real world examples of using decomposition include:

- Creating a video game
- Complex maths problems
- Cooking
- Cleaning your room



Recap Abstraction

Abstraction is the process of removing unnecessary detail and simplifying.

Abstraction is used to remove unnecessary detail from a real-world situation and to model the simplified result in an algorithm or program.

Simple Interest vs. Compound Interest

Most bank accounts will pay you **interest** on money you have deposited with them.

Imagine that you have £1000 in a bank account. Would you rather I gave you:

- 100% interest after 10 years
- 10% interest every year for 10 years

Calculating Compound Interest

Year	Base Amount	Interest *(Base * Interest Rate)*	Total
1	£1000	£1000 * 0.1 = **£100**	£1000 + £100 = **£1100**
2	£1100	£1100 * 0.1 = **£110**	£1100 + £110 = **£1210**
3			
4			
…	…	…	…
10			**???**

GCSE Compound Interest

GCSE MATHEMATICS - NUMERACY Specimen Assessment Materials 100

9. Carys decides to invest £380 in a savings account for 6 years. The account pays a rate of 2·54% AER.

Will Carys have sufficient money in her savings account to be able to buy a motor scooter costing £460 in 6 years' time? You must show all your working and give a reason for your answer.

[4]

First, Abstract Away The Unimportant Details

GCSE MATHEMATICS - NUMERACY Specimen Assessment Materials 100

9. Carys decides to invest £380 in a savings account for 6 years. The account pays a rate of 2·54% AER.

Will Carys have sufficient money in her savings account to be able to buy a motor scooter costing £460 in 6 years' time? You must show all your working and give a reason for your answer.

[4]

First, Abstract Away The Unimportant Details

GCSE MATHEMATICS - NUMERACY Specimen Assessment Materials 100

9. Carys decides to invest £380 in a savings account for 6 years. The account pays a rate of 2·54% AER.

Will Carys have sufficient money in her savings account to be able to buy a motor scooter costing £460 in 6 years' time? You must show all your working and give a reason for your answer.

[4]

startingAmount = £380

First, Abstract Away The Unimportant Details

GCSE MATHEMATICS - NUMERACY Specimen Assessment Materials 100

9. Carys decides to invest £380 in a savings account for 6 years. The account pays a rate of 2·54% AER.

Will Carys have sufficient money in her savings account to be able to buy a motor scooter costing £460 in 6 years' time? You must show all your working and give a reason for your answer.

[4]

startingAmount = £380

time = 6 years

First, Abstract Away The Unimportant Details

GCSE MATHEMATICS - NUMERACY Specimen Assessment Materials 100

9. Carys decides to invest £380 in a savings account for 6 years. The account pays a rate of 2·54% AER.

Will Carys have sufficient money in her savings account to be able to buy a motor scooter costing £460 in 6 years' time? You must show all your working and give a reason for your answer.

[4]

startingAmount = £380

time = 6 years

interestRate = 2.54% (Divide this by 100 to get the decimal 0.0254)

First, Abstract Away The Unimportant Details

GCSE MATHEMATICS - NUMERACY Specimen Assessment Materials 100

9. Carys decides to invest £380 in a savings account for 6 years. The account pays a rate of 2·54% AER.

Will Carys have sufficient money in her savings account to be able to buy a motor scooter costing £460 in 6 years' time? You must show all your working and give a reason for your answer.

[4]

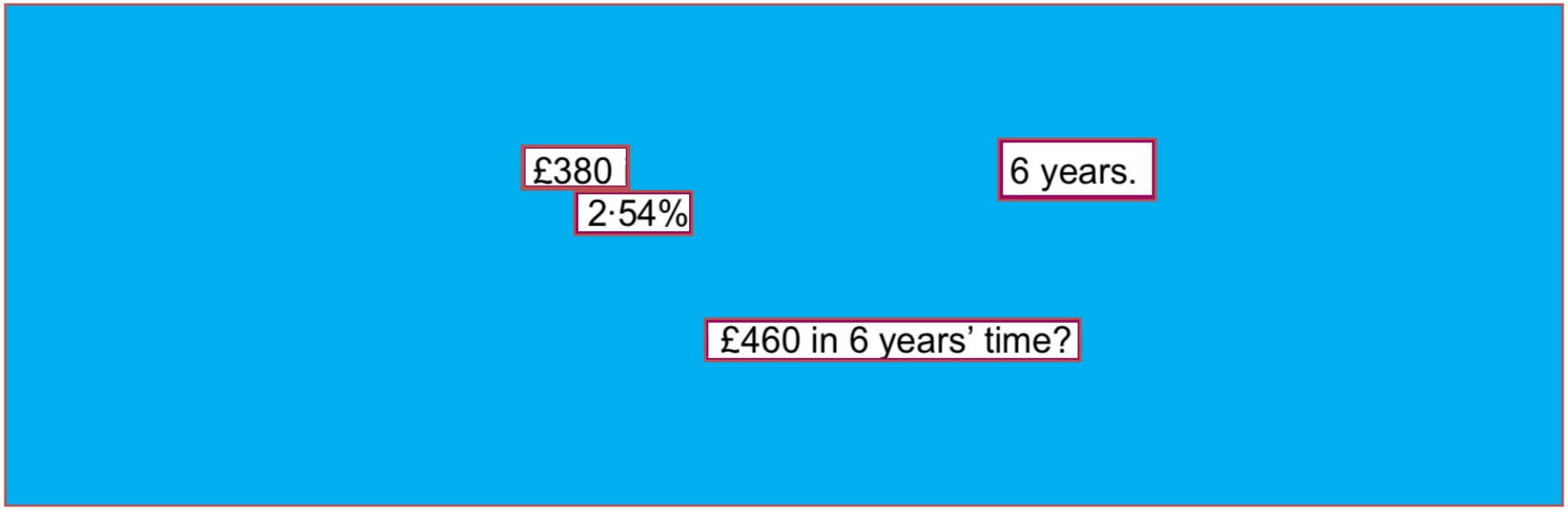
startingAmount = £380

time = 6 years

interestRate = 2.54% (Divide this by 100 to get the decimal 0.0254)

costOfBike = £460

First, Abstract Away The Unimportant Details



startingAmount = £380

time = 6 years

interestRate = 2.54% (Divide this by 100 to get the decimal 0.0254)

costOfBike = £460

The First Year

We have the initial £380. The interest gained which would be:

£380 x 0.0254 = £9.652

Therefore after the **first** year we have: £380 + £9.652 = £389.652

The Second Year

After the first year we have: £389.652

Now we do the same for the second year, however the amount we have to start with has increased:

£389.652 + (£389.652 x 0.0254) = £399.54916

The Final Year

We **repeat** this process until we have done it 6 times and we reach the final value of £441.72:

Starting amount: £380

1st Year: £389.652

2nd Year: £399.549

3rd Year: £409.697

4th Year: £420.104

5th Year: £430.774

6th Year: £441.716

Activity: Compound Interest

Activity: Compound Interest in Python

Create a Python program which will allow a user to input an **initial value**, a **number of years** for saving, an **annual interest rate** (you'll need to decide whether to ask for a decimal or percentage value) and the **cost of an item to compare to** at the end.

The program will then need to calculate and output the final balance and compare it to the cost of the item and output if the balance is enough to purchase the item.

Is There An Easier Way?

Can we implement this without using a loop?

Is There An Easier Way?

Can we implement this without using a loop?

First we have an initial amount $\boldsymbol{A}$, and in the first year we are adding to it this amount multiplied by the interest rate $\boldsymbol{R}$. therefore after the first year the total $\boldsymbol{T}$ is given by:

Is There An Easier Way?

Can we implement this without using a loop?

First we have an initial amount $\boldsymbol{A}$, and in the first year we are adding to it this amount multiplied by the interest rate $\boldsymbol{R}$. therefore after the first year the total $\boldsymbol{T}$ is given by:

$T = A + AR$ which we rewrite as $T = A(1 + R)$

Is There An Easier Way?

Can we implement this without using a loop?

First we have an initial amount $\boldsymbol{A}$, and in the first year we are adding to it this amount multiplied by the interest rate $\boldsymbol{R}$. therefore after the first year the total $\boldsymbol{T}$ is given by:

$T = A + AR$ which we rewrite as $T = A(1 + R)$

Why?

Equation for Compound Interest

After the second year we just repeat the process of multiplying our amount by the ratio 1.0254

So every year we just multiply by another 1.0254 or another $(1 + R)$.

Equation for Compound Interest

So every year we just multiply by another 1.0254 or another $(1 + R)$.

Written out another way it looks like this:

First Year: $T = A(1 + R)$

Second Year: $T = A(1 + R)(1 + R)$

Third Year: $T = A(1 + R)(1 + R)(1 + R)$

Fourth Year: $T = A(1 + R)(1 + R)(1 + R)(1 + R)$

Fifth Year: $T = A(1 + R)(1 + R)(1 + R)(1 + R)(1 + R)$

Sixth Year: $T = A(1 + R)(1 + R)(1 + R)(1 + R)(1 + R)(1 + R)$

Equation for Compound Interest

So every year we just multiply by another 1.0254 or another $(1 + R)$.

First year: $T = A(1 + R)$

Second Year: $T = A(1 + R)(1 + R)$

Third Year: $T = A(1 + R)(1 + R)(1 + R)$

Fourth Year: $T = A(1 + R)(1 + R)(1 + R)(1 + R)$

Fifth Year: $T = A(1 + R)(1 + R)(1 + R)(1 + R)(1 + R)$

Sixth Year: $T = A(1 + R)(1 + R)(1 + R)(1 + R)(1 + R)(1 + R)$

Simplifying:

$$T = A(1 + R)(1 + R)(1 + R)(1 + R)(1 + R)(1 + R) = \boldsymbol{A(1 + R)^6}$$

So with n amount of years:

$$\boldsymbol{T = A(1 + R)^n}$$

technoteach
technocamps

Sort & Search Algorithms

Sorting and Searching

Sorting and searching through lists is a very important part of computer science

There are various algorithms that approach doing so in different ways

They often have different trade-offs: simplicity vs efficiency

For many search algorithms, the lists must be sorted first

But first, a recap on lists!

What are Lists?

A computer program often needs to store a sequence of values and then process them.

For example if you had to store the below sequence of values, how many variables would you need?

32	**54**	**67.5**	**29**	**35**	**80**	**115**	**44.5**	**100**	**65**

This is where lists become very useful, saving us time and making the process of storing a sequence of values much easier.

List definition: A List is a collection of values which is ordered and changeable.

Lists in Python

```
#Introduction to Lists

#Two ways of initializing a list

initialisingAnEmptyList = []

initialisingAListWithValues = [32,54,66,28,39,87,111]
```

Each item in a list has a corresponding **index number**, which is an integer value, starting with the index number 0. We use this index number to access an item in the list.

index	0	1	2	3	4	5	6
values	32	54	66	28	39	87	111

Accessing Lists in Python

```
#Introduction to Lists

#Two ways of initializing a list

initialisingAnEmptyList = []

initialisingAListWithValues = [32,54,66,28,39,87,111]

#Accessing values in a List

thirdItem = initialisingAListWithValues[2]

firstItem = initialisingAListWithValues[0]

#Error accessing the list

errorInAccess = initialisingAListWithValues[7]
```

Causes runtime error

Replacing Values in a List

To replace a value in the list we first identify the index number of the value to be changed and then set the value in the corresponding index number to a different value.

```
initialisingAListWithValues = [32,54,66,28,39,87,111]

#Replacing values in a List

initialisingAListWithValues[5] = 88
```

List Boundaries

You have to be careful that the index number stays within the valid range.

Attempting to access an element whose index number is not within the valid index range is called an **out-of-range error** or a bounds error which in turn causes a run-time exception.

Example:

```
values = [10,20,30,40,50,60,70,80,90]

values[9] = number #results in

#IndexError: list index out of range
```

Length of a List

We can use the **len()** function to obtain the length of the list; that is, the number of elements:

```
values = [10,20,30,40,50,60,70,80,90]

numElements = len(values)

#The value of numElements will be 9
```

Iterating a List

A **for** loop is ideal to iterate through the items in a list. We use a variable in the **for** loop that corresponds to the index number.

The **range(n)** function yields the numbers 0, 1, ... n-1, and **range(a, b)** returns a, a+1, ... b-1 up to but not including the last number (**b**). The combination of the for-loop and the range() function allows you to iterate through the list easily.

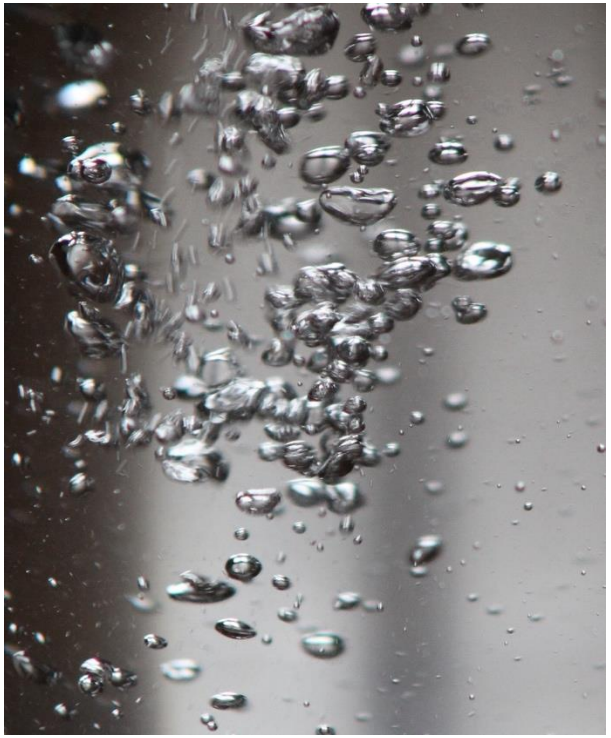
```
#Initialising a list with values
aListWithValues = [32,54,66,28,30,87,111,72,94,16]
#Loop over the index values
for i in range(len(aListWithValues)):
      print(i, aListWithValues[i])
```

Sorting Algorithms

Bubble Sort

Sometimes we have a list of numbers that we would like to be sorted. Algorithms that can do this are called **sorting algorithms** and an example of a sorting algorithm is Bubble sort.

In Bubble sort, the larger numbers 'bubble' up to the top of the list.



Bubble Sort

How to Bubble Sort:

From left to right, compare two numbers, swap if needed. Repeat until all numbers in correct order.

6 5 3 1 8 7 2 4

Bubble Sort

How to Bubble Sort:

From left to right, compare two numbers, swap if needed. Repeat until all numbers in correct order.

5 6 3 1 8 7 2 4

Bubble Sort

How to Bubble Sort:

From left to right, compare two numbers, swap if needed. Repeat until all numbers in correct order.

5 6 3 1 8 7 2 4

Bubble Sort

How to Bubble Sort:

From left to right, compare two numbers, swap if needed. Repeat until all numbers in correct order.

5 3 6 1 8 7 2 4

Bubble Sort

How to Bubble Sort:

From left to right, compare two numbers, swap if needed. Repeat until all numbers in correct order.

5 3 6 1 8 7 2 4

Bubble Sort

How to Bubble Sort:

From left to right, compare two numbers, swap if needed. Repeat until all numbers in correct order.

5 3 1 6 8 7 2 4

Bubble Sort

How to Bubble Sort:

From left to right, compare two numbers, swap if needed. Repeat until all numbers in correct order.

5 3 1 6 8 7 2 4

Bubble Sort

How to Bubble Sort:

From left to right, compare two numbers, swap if needed. Repeat until all numbers in correct order.

5 3 1 6 8 7 2 4

Bubble Sort

How to Bubble Sort:

From left to right, compare two numbers, swap if needed. Repeat until all numbers in correct order.

5 3 1 6 7 8 2 4

Bubble Sort

How to Bubble Sort:

From left to right, compare two numbers, swap if needed. Repeat until all numbers in correct order.

5 3 1 6 7 8 2 4

Bubble Sort

How to Bubble Sort:

From left to right, compare two numbers, swap if needed. Repeat until all numbers in correct order.

5 3 1 6 7 2 8 4

Bubble Sort

How to Bubble Sort:

From left to right, compare two numbers, swap if needed. Repeat until all numbers in correct order.

5 3 1 6 7 2 8 4

Bubble Sort

How to Bubble Sort:

From left to right, compare two numbers, swap if needed. Repeat until all numbers in correct order.

5 3 1 6 7 2 4 8

Bubble Sort

How to Bubble Sort:

From left to right, compare two numbers, swap if needed. **Repeat until all numbers in correct order.**

5 3 1 6 7 2 4 8

Bubble Sort

How to Bubble Sort:

From left to right, compare two numbers, swap if needed. **Repeat until all numbers in correct order.**

5 3 1 6 7 2 4 8

Repeat process until all numbers in correct order.

Bubble Sort

How to Bubble Sort:

From left to right, compare two numbers, swap if needed. **Repeat until all numbers in correct order.**

1 2 3 4 5 6 7 8

Repeat process until all numbers in correct order.

Activity: Bubble Sort

Using the following numbers '7 4 1 5 8 3 6 2' perform a bubble sort to sort the numbers in a numerical order from smallest to largest. Write down the steps you performed to complete this task in your workbooks.

Again the list of numbers to sort are:

7 4 1 5 8 3 6 2

Activity: Bubble Sort Solution

7 4 1 5 8 3 6 2

Activity: Bubble Sort Solution

4 1 5 7 3 6 2 8

Activity: Bubble Sort Solution

1 4 5 3 6 2 7 8

Activity: Bubble Sort Solution

1 4 3 5 2 6 7 8

Activity: Bubble Sort Solution

1 3 4 2 5 6 7 8

Activity: Bubble Sort Solution

1 3 2 4 5 6 7 8

Activity: Bubble Sort Solution

1 2 3 4 5 6 7 8

Activity: Bubble Sort Solution

7 4 1 5 8 3 6 2

4 1 5 7 3 6 2 8

1 4 5 3 6 2 7 8

1 4 3 5 2 6 7 8

1 3 4 2 5 6 7 8

1 3 2 4 5 6 7 8

1 2 3 4 5 6 7 8

A-Level Activity: Bubble Sort in Python

Understanding Required

Bubble Sort Solution – main Function

```
# Bubble sort main

def main():

    unorderedList = [34,23,56,89,23,43,55,75,4,2,6,10,11]

    print(bubbleSort(unorderedList))



main()
```

Bubble Sort Solution – Sort Function

```
# Bubble sort function
def bubbleSort(aList):
    n = len(aList)
    swapped = False
    while n > 0:
        swapped = False
        for i in range(1, n):
            if aList[i-1] > aList[i]:
                temp = aList[i]
                aList[i] = aList[i-1]
                aList[i-1] = temp
                                #aList[i],  aList[i-1]  =  aList[i-
1],aList[i]
                swapped = True
        n = n - 1
    return alist
```

Merge Sort

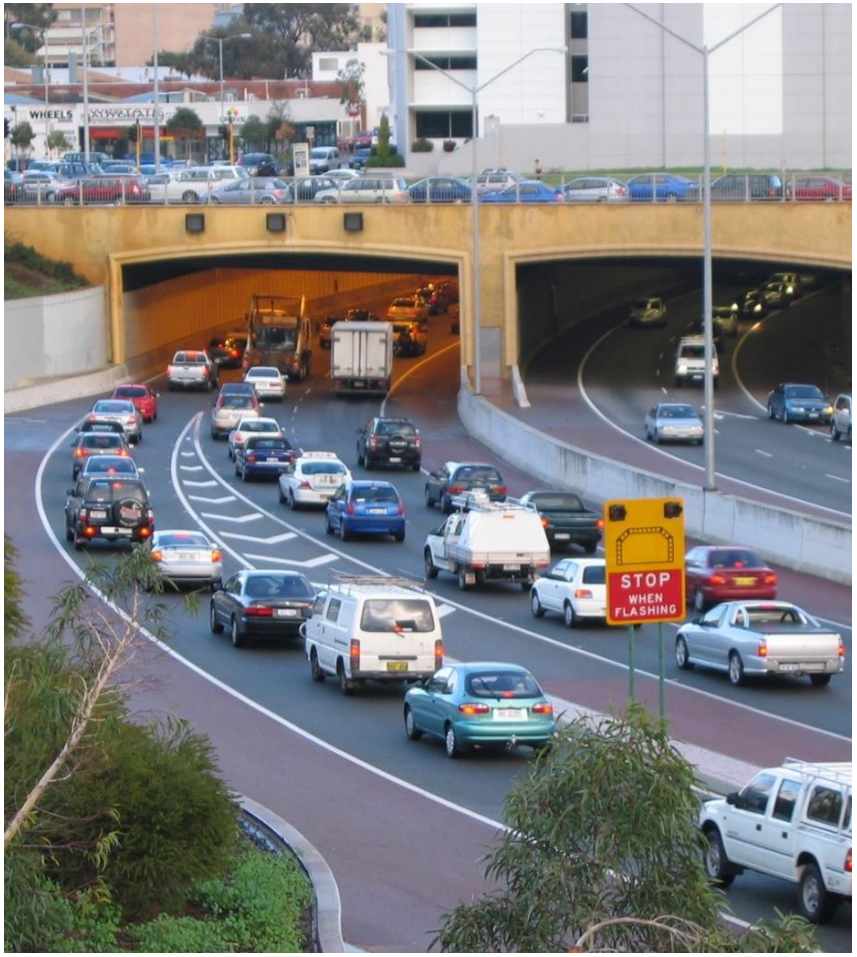
Merge Sort is a "divide and conquer" sorting algorithm. We repeatedly **divide** the list of numbers into smaller and smaller lists until each list contains only one item. We then **merge** these smaller lists together, making sure that the newly merged list is sorted. We keep on merging smaller lists into bigger lists until the whole original list is sorted.

Merge sort is much faster at sorting than Bubble sort. It is important to remember than Bubble sort is OK for small lists but big lists should be sorted with a faster sorting algorithm, like Merge sort.

Merge Sort

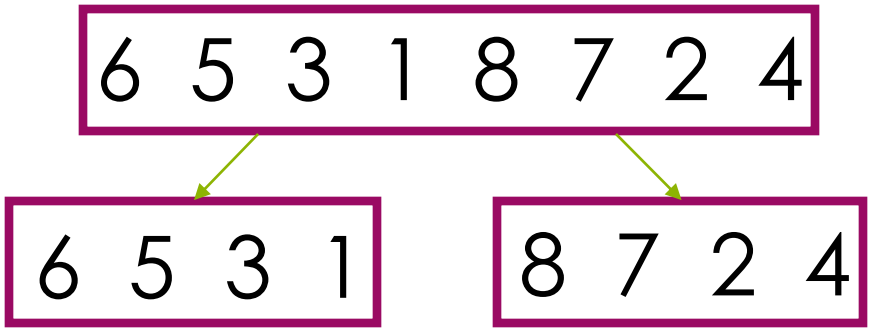
Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. Then we merge the lists together making sure the items are in the correct order.

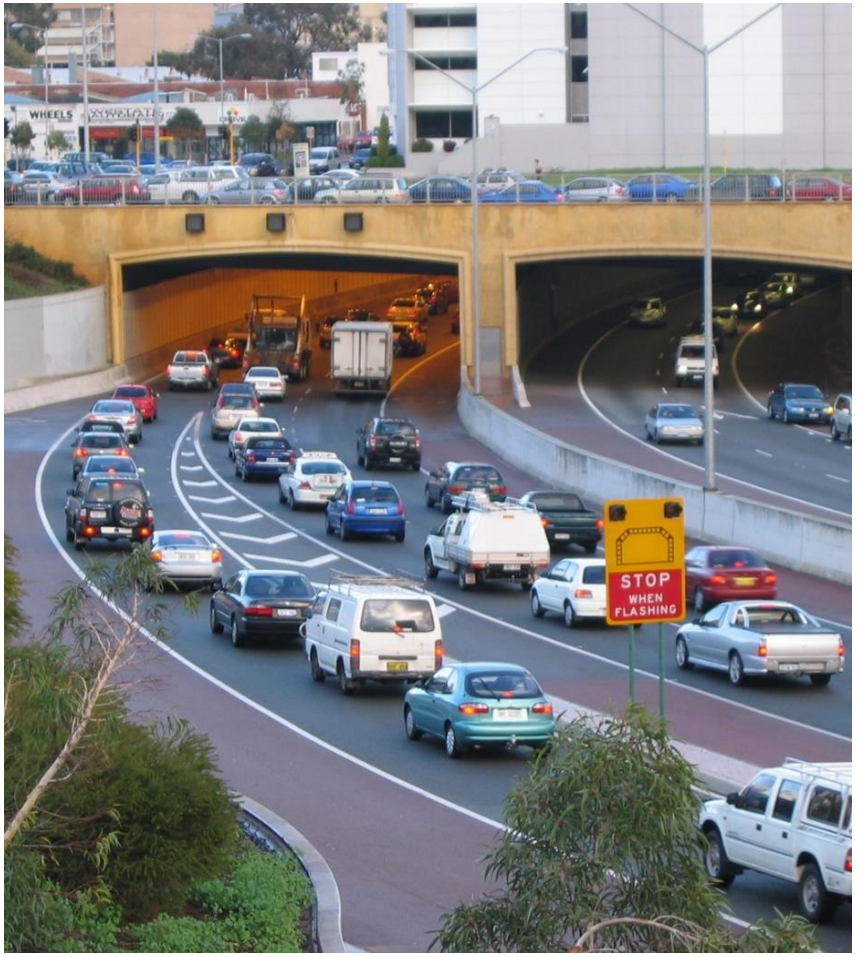
6 5 3 1 8 7 2 4



Merge Sort

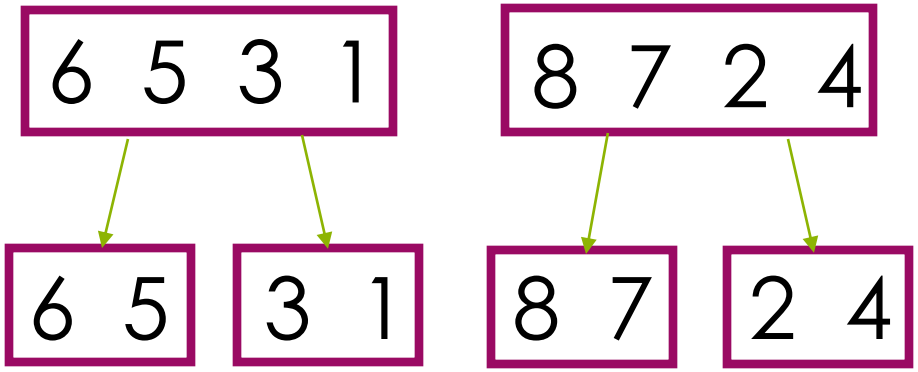
Unlike Bubble Sort, we repeatedly **split the lists into smaller and smaller lists until there is only one item in each list**. Then we merge the lists together making sure the items are in the correct order.

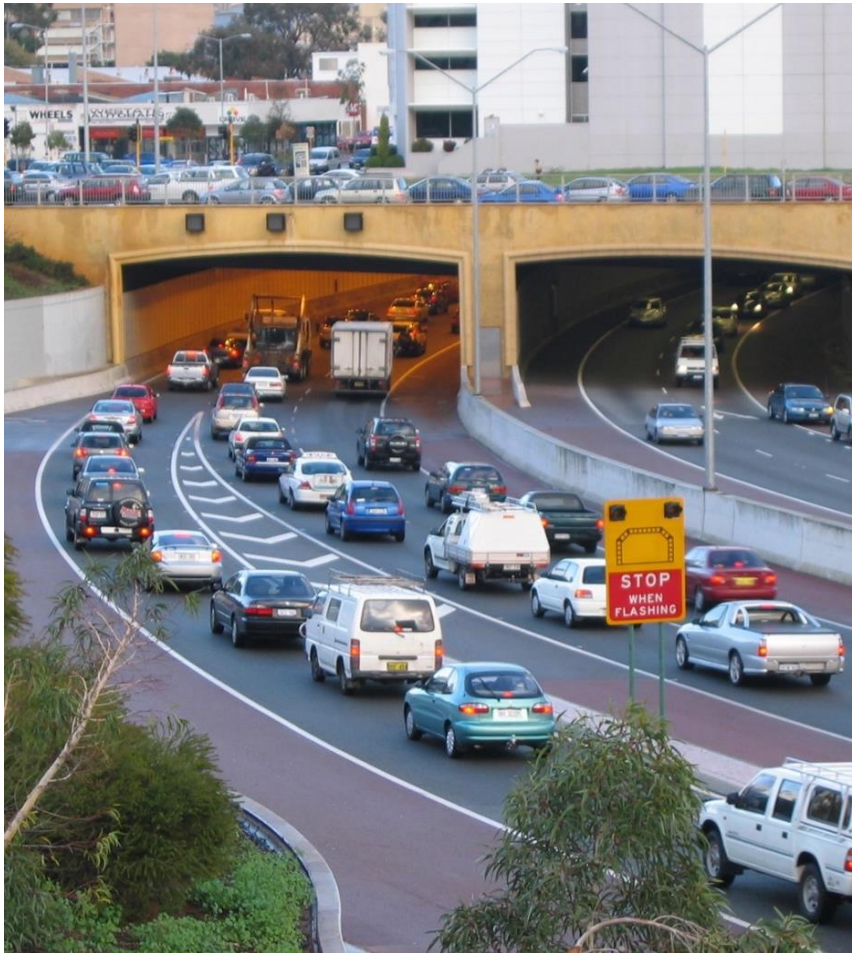




Merge Sort

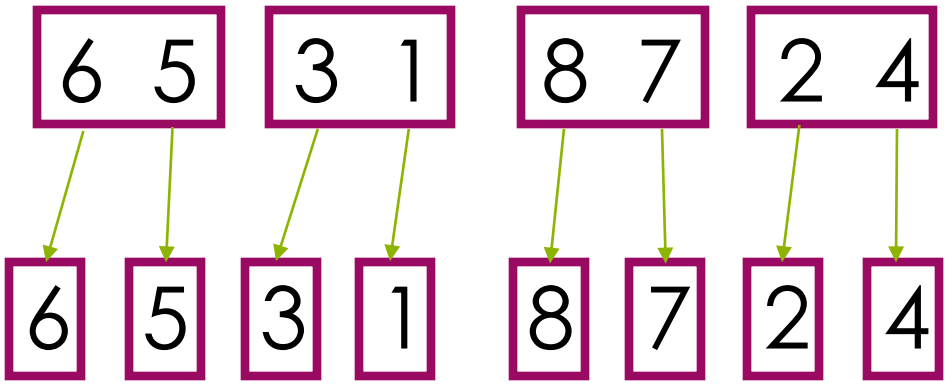
Unlike Bubble Sort, we repeatedly **split the lists into smaller and smaller lists until there is only one item in each list.** Then we merge the lists together making sure the items are in the correct order.

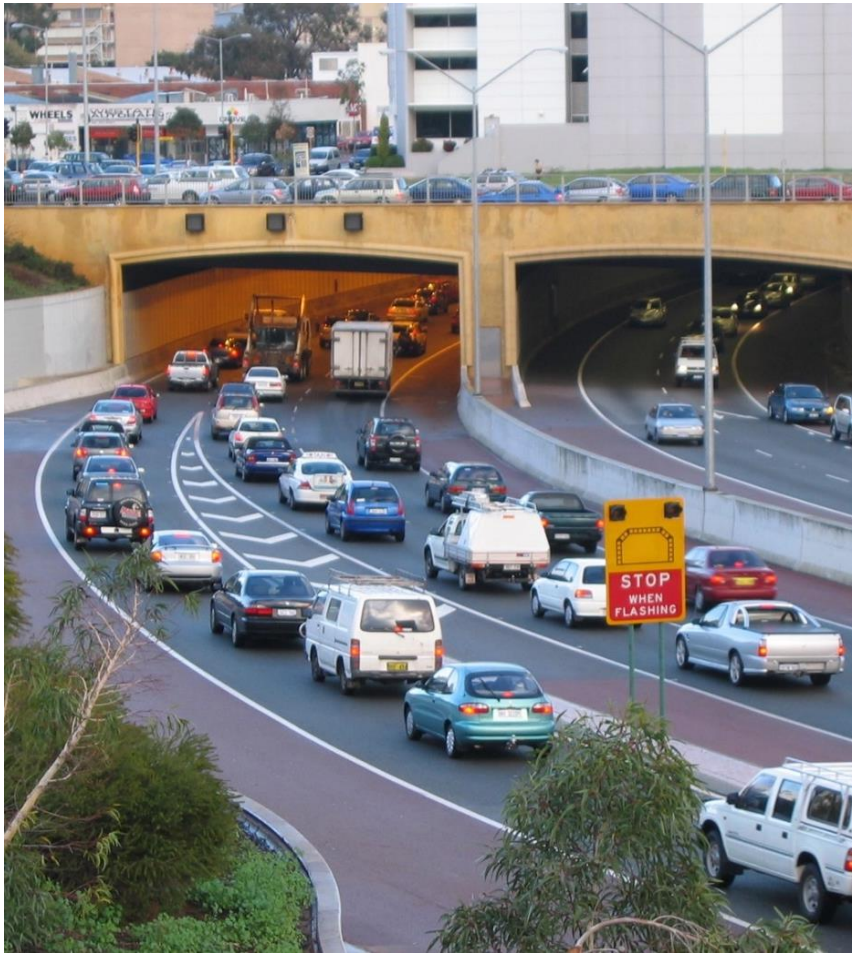




Merge Sort

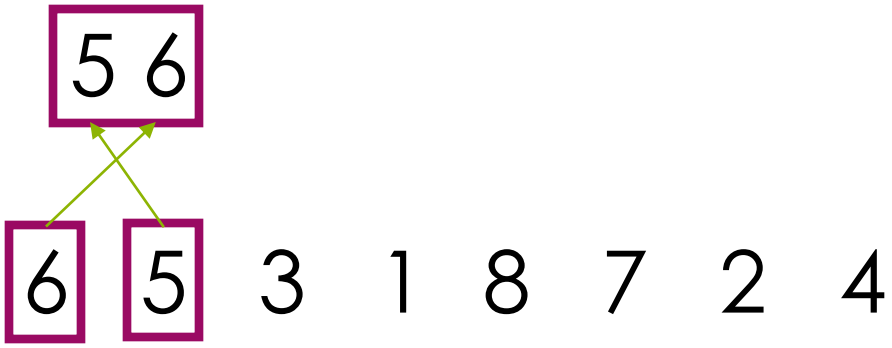
Unlike Bubble Sort, we repeatedly **split the lists into smaller and smaller lists until there is only one item in each list.** Then we merge the lists together making sure the items are in the correct order.

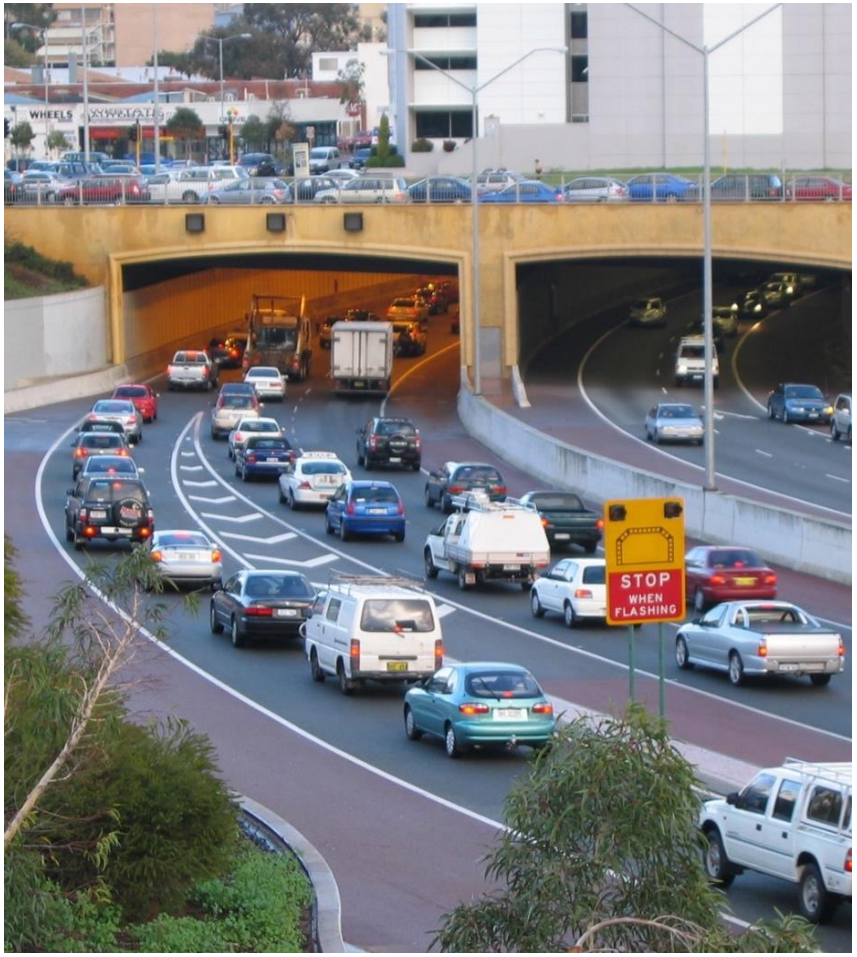




Merge Sort

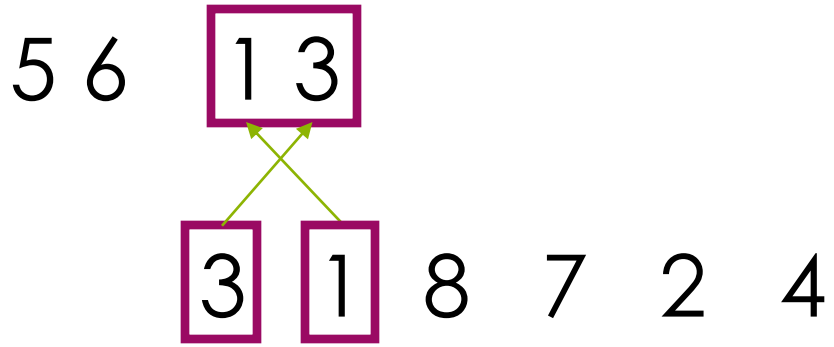
Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**

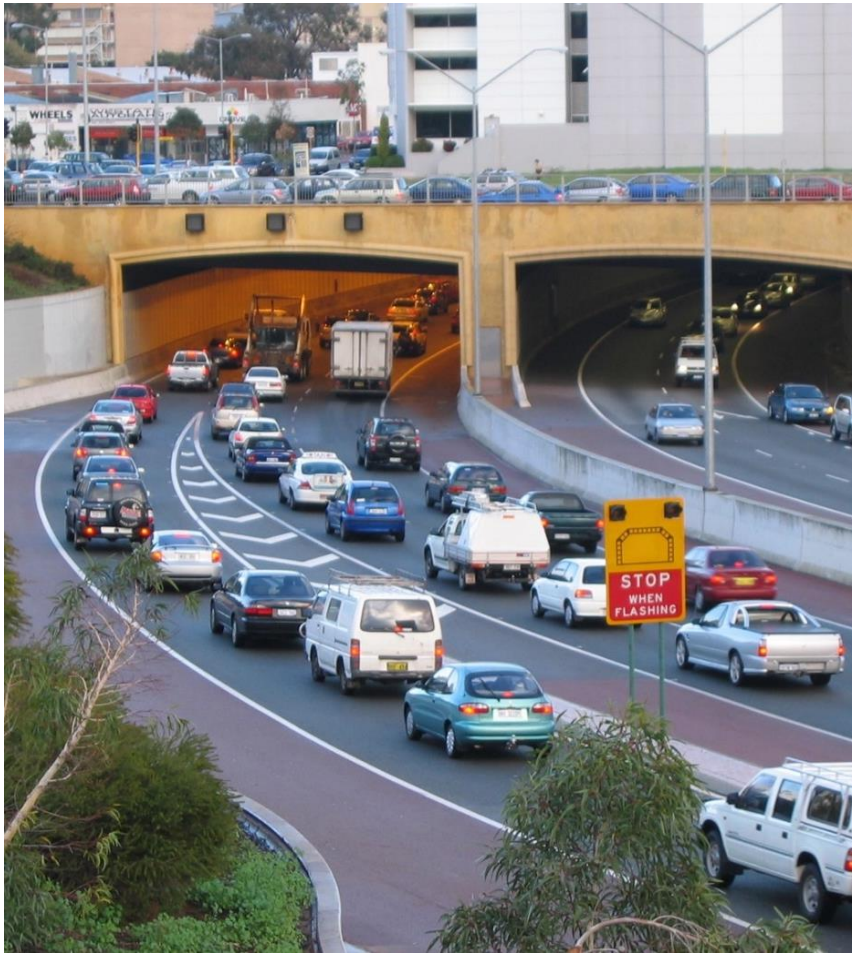




Merge Sort

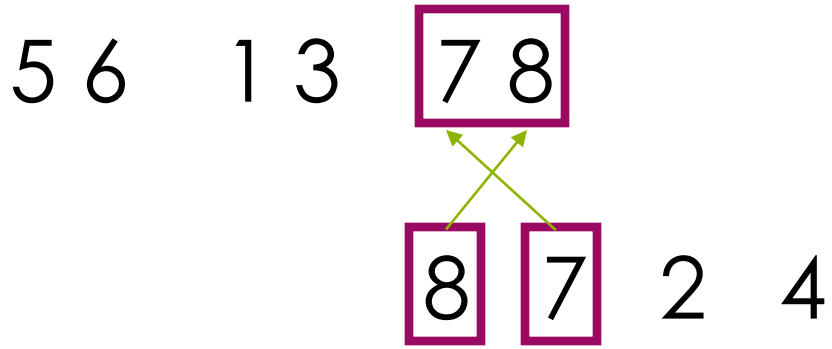
Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**

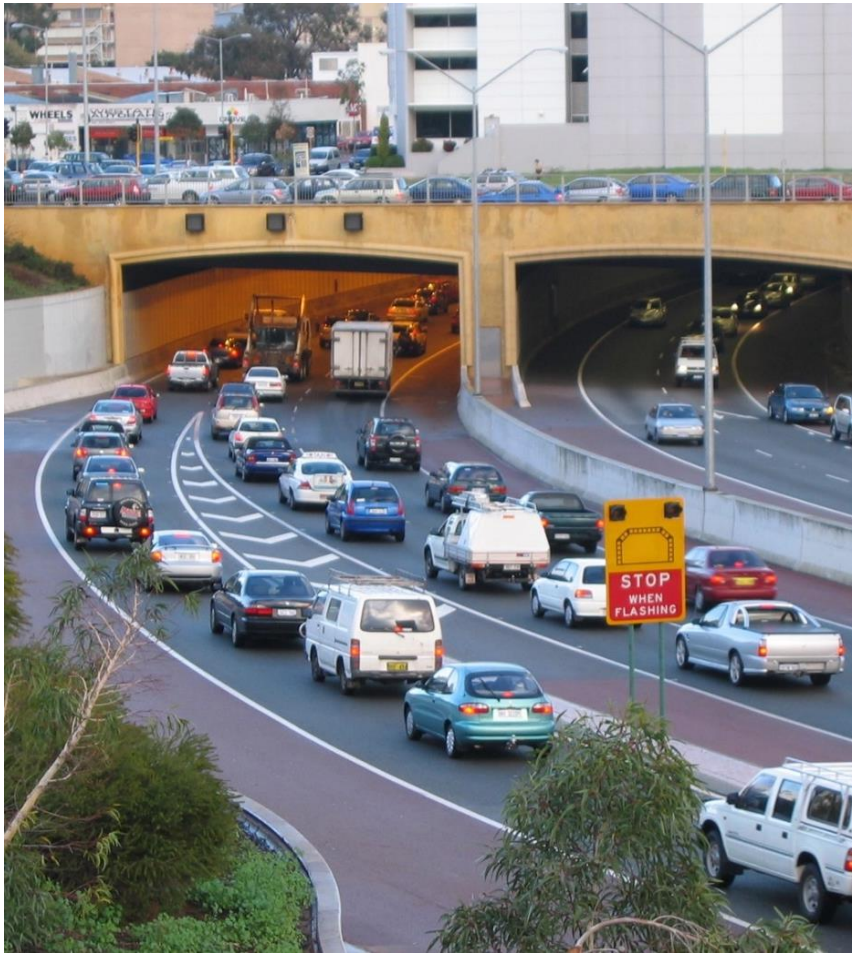




Merge Sort

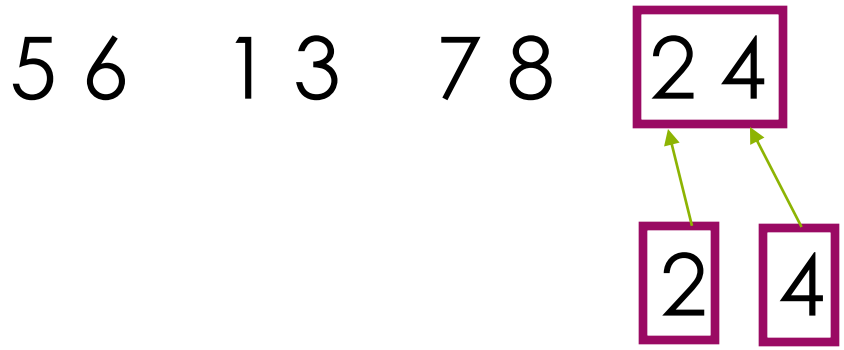
Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. Then we merge the lists together making sure the items are in the correct order.





Merge Sort

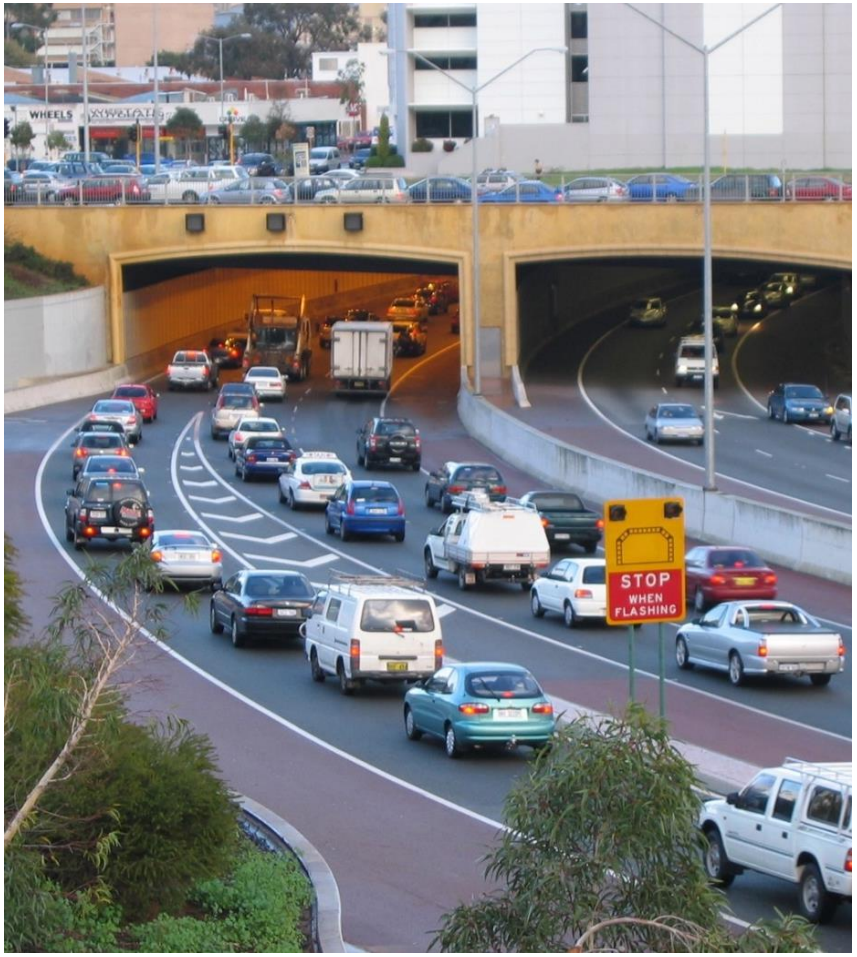
Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**



Merge Sort

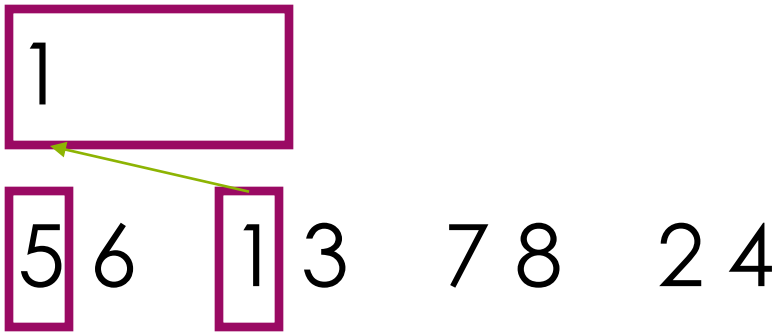
Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**

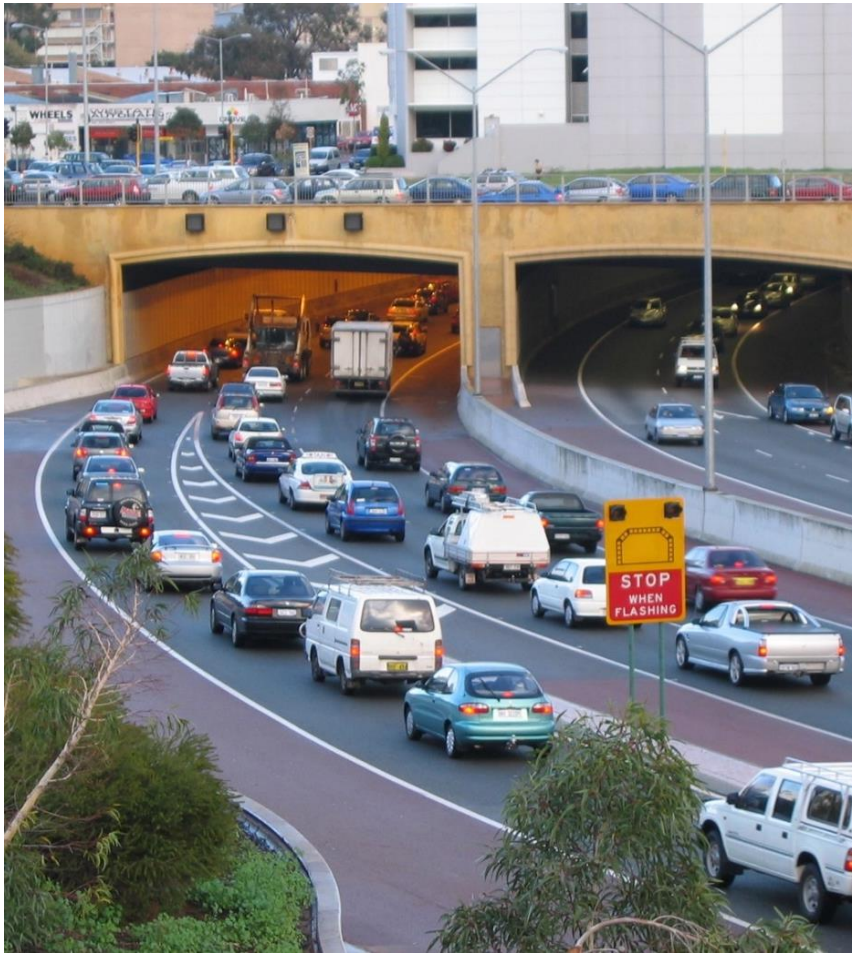
5 6 1 3 7 8 2 4



Merge Sort

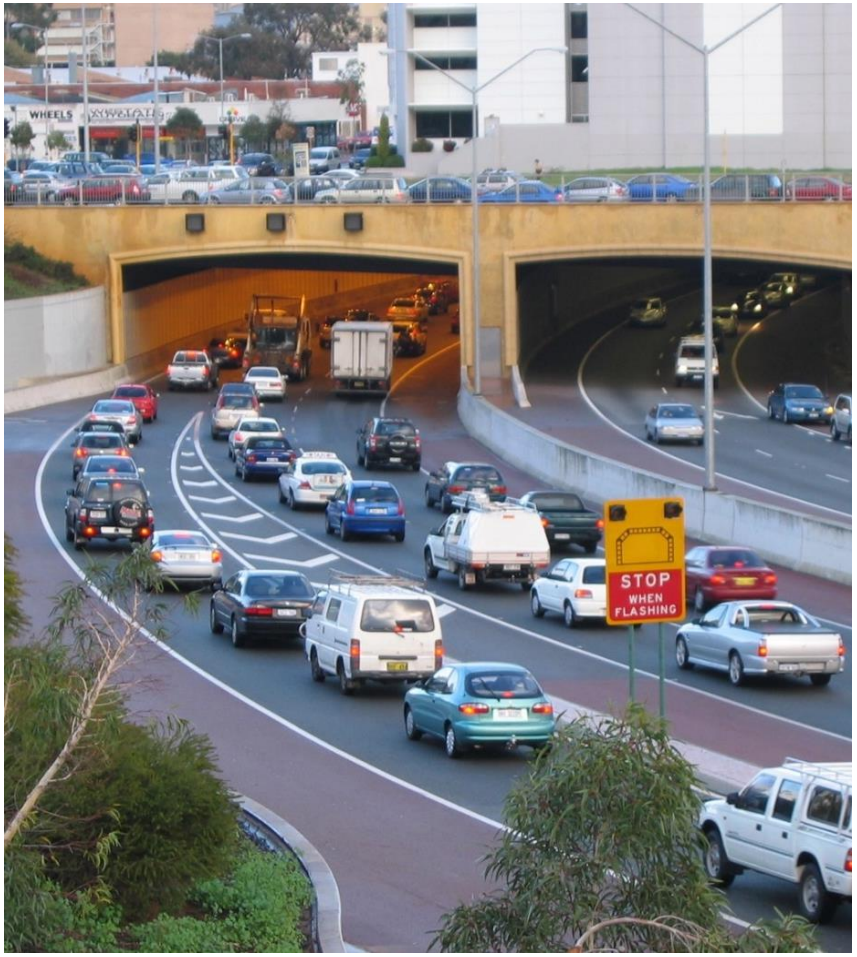
Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**





Merge Sort

Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**

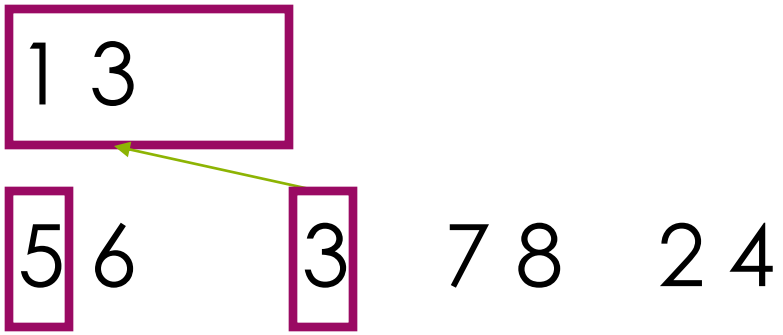


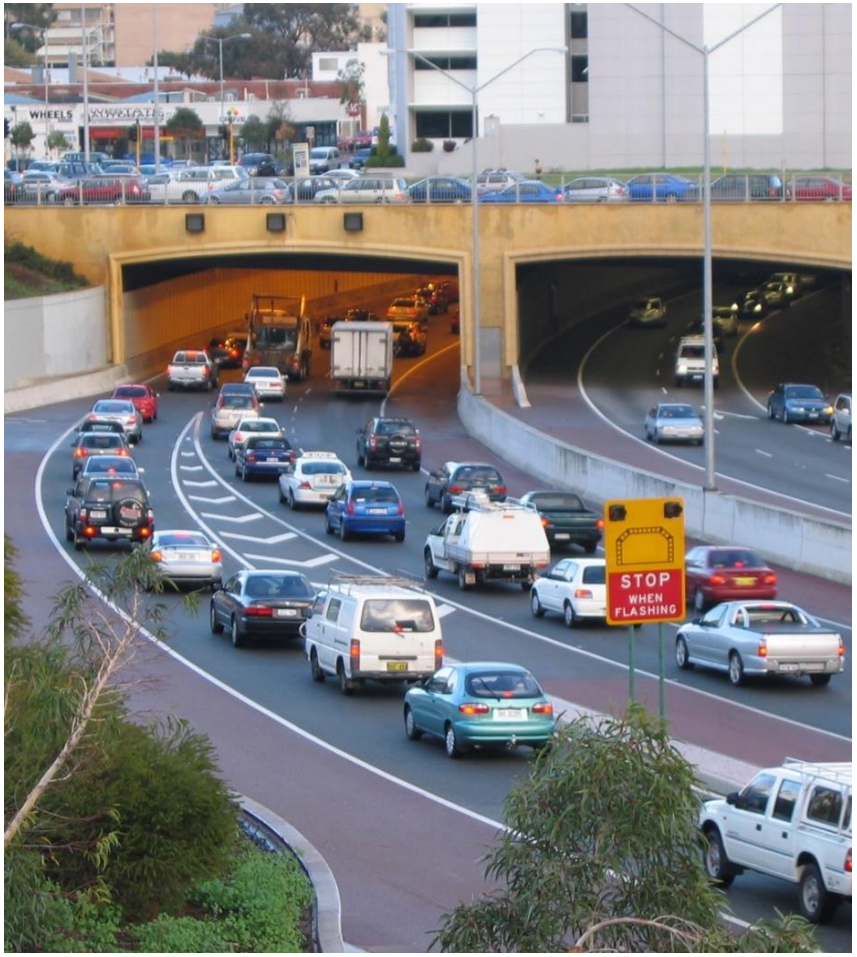
1

5 6 3 7 8 2 4

Merge Sort

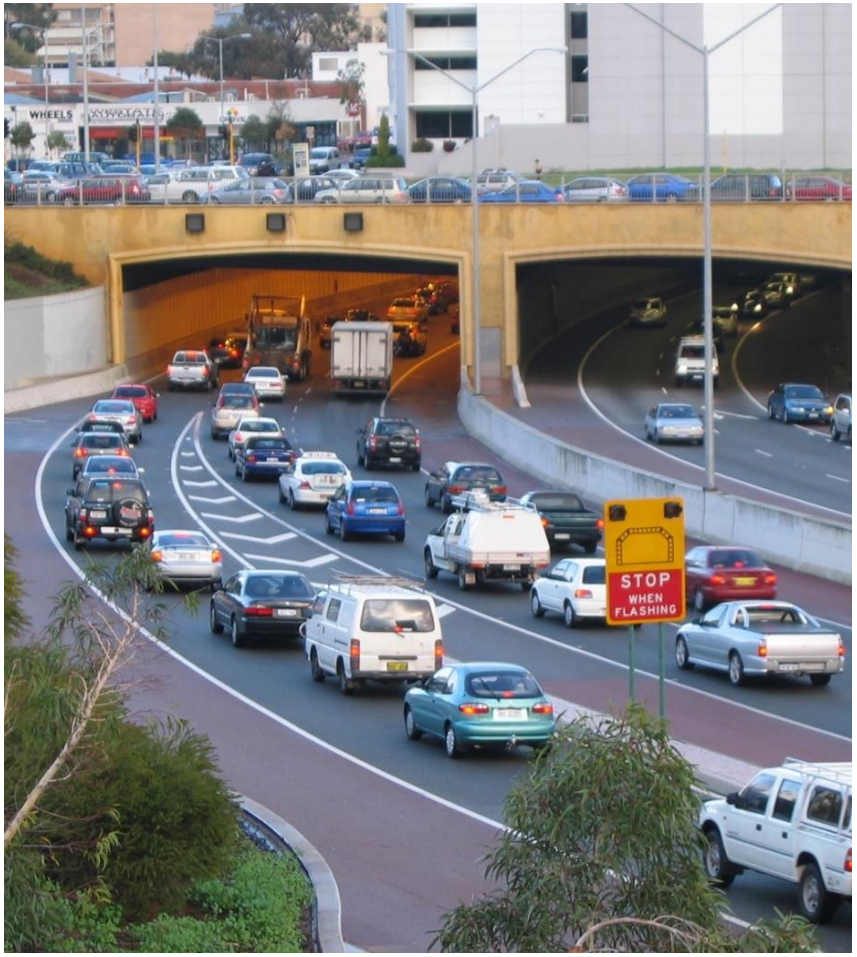
Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**





Merge Sort

Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**

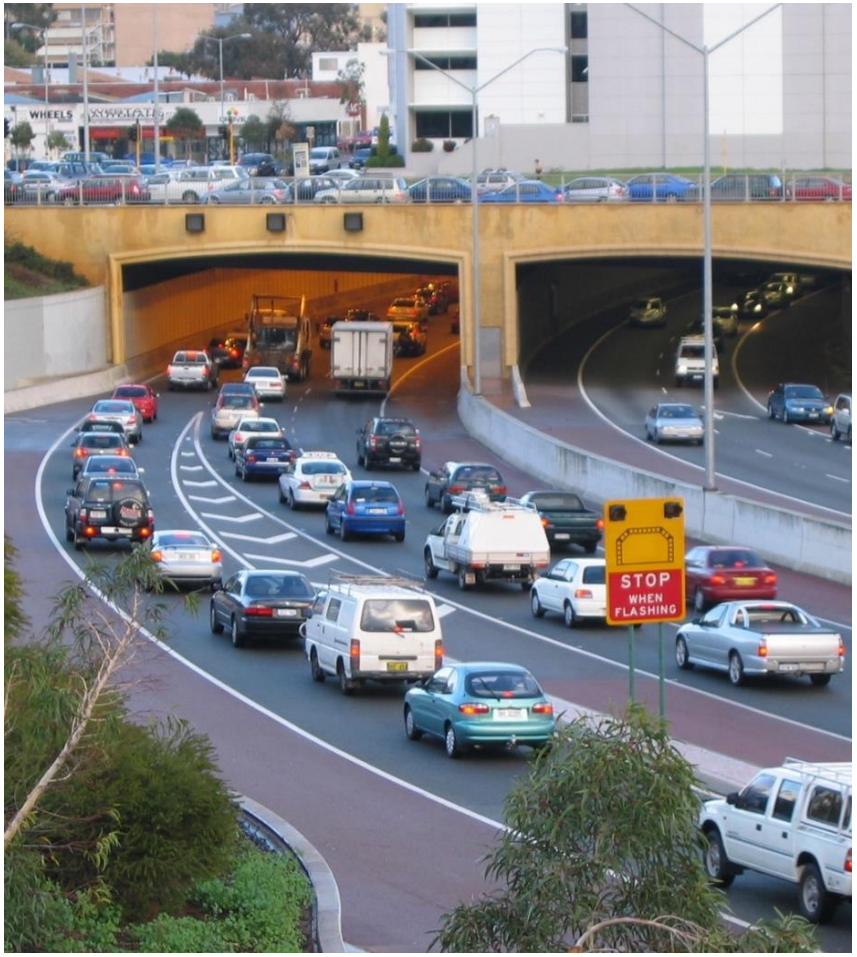


1 3

5 6 7 8 2 4

Merge Sort

Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**



1 3 5

5 6 7 8 2 4

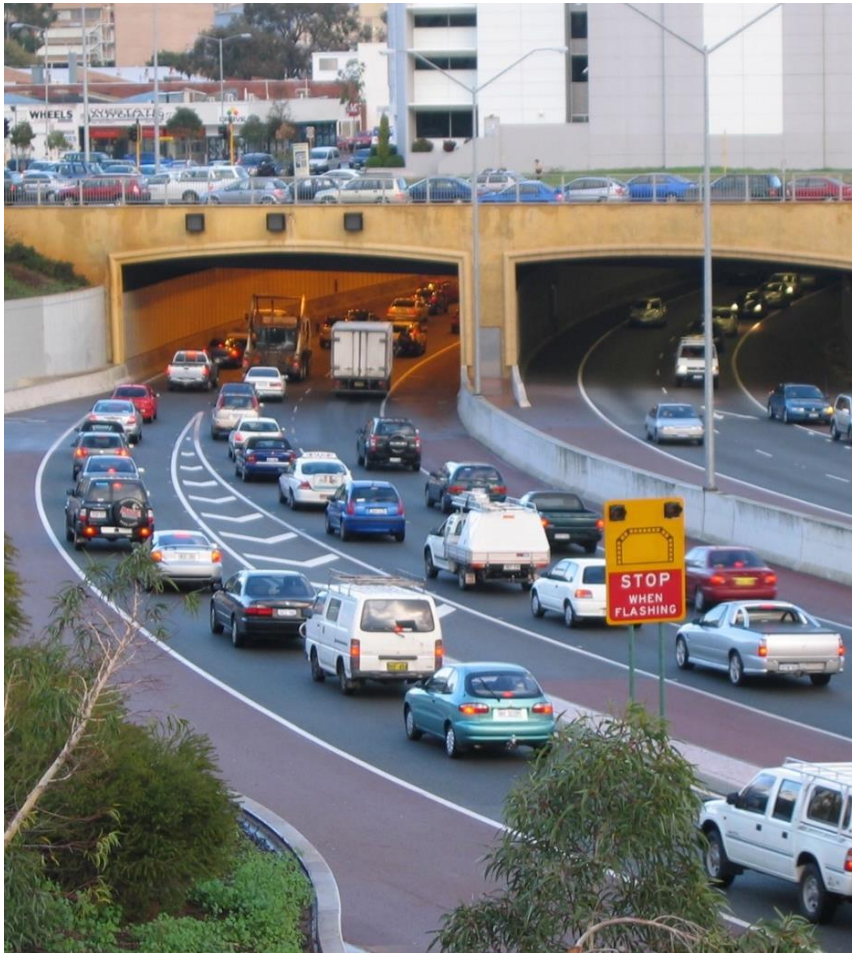
Merge Sort

Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**

1 3 5

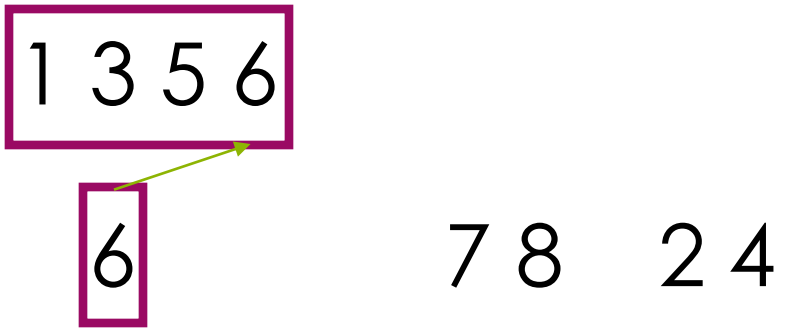
6

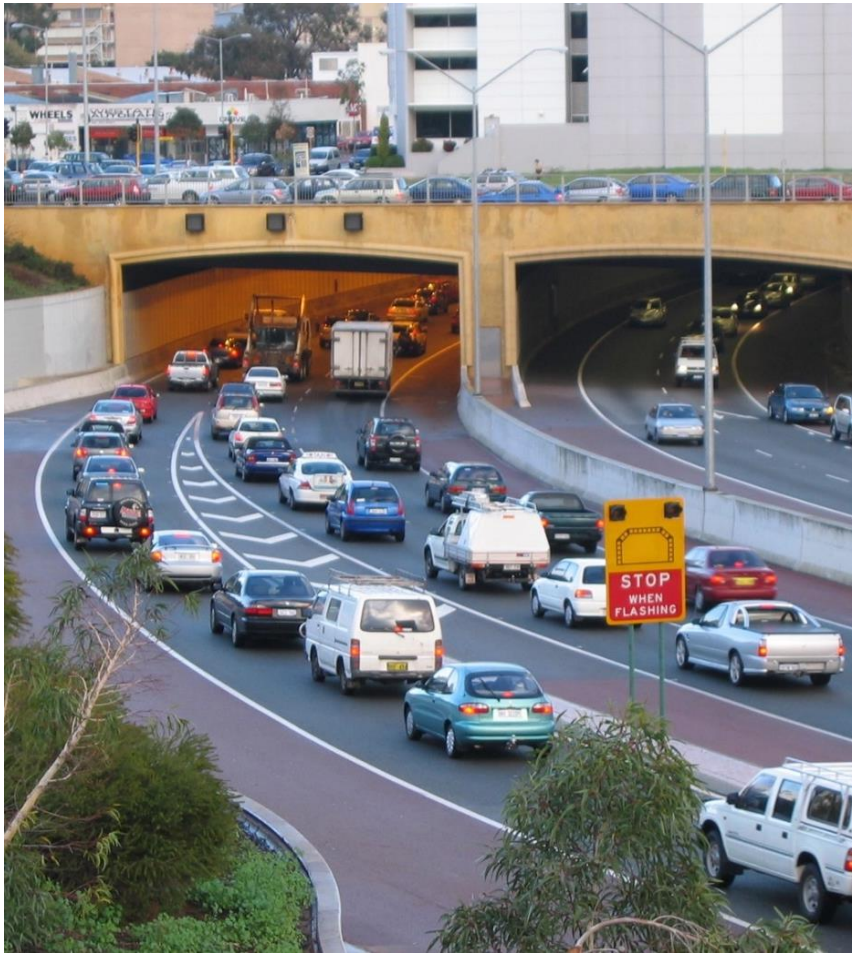
7 8 2 4



Merge Sort

Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**



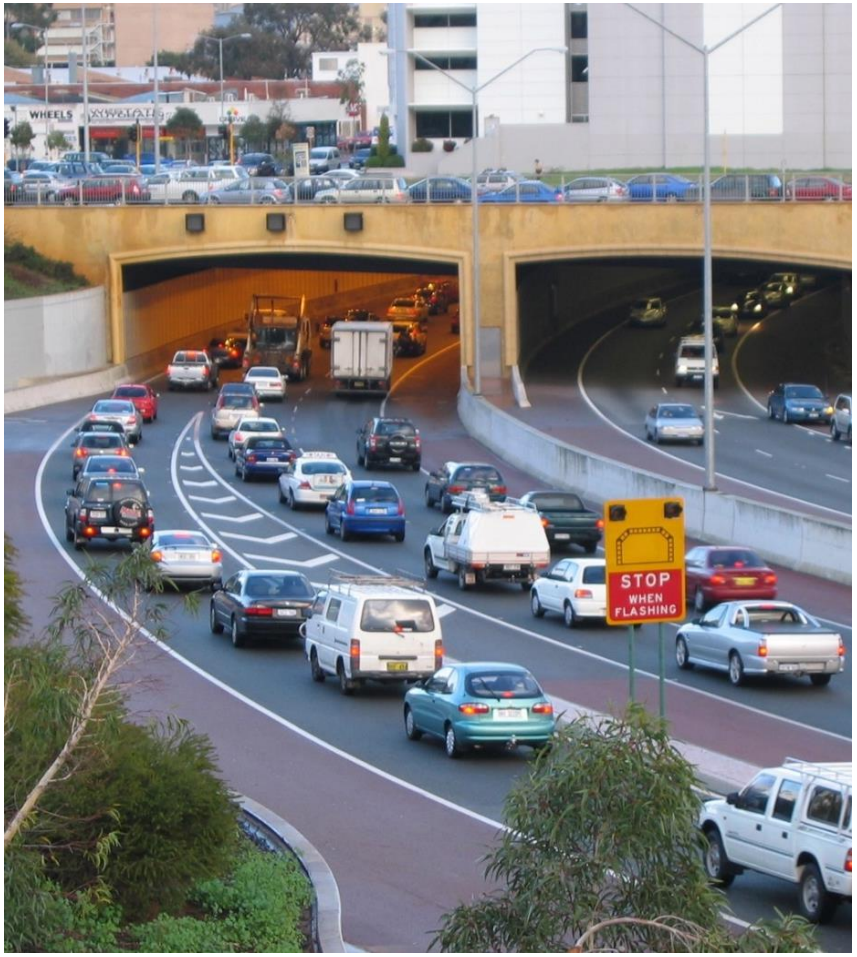


Merge Sort

Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**

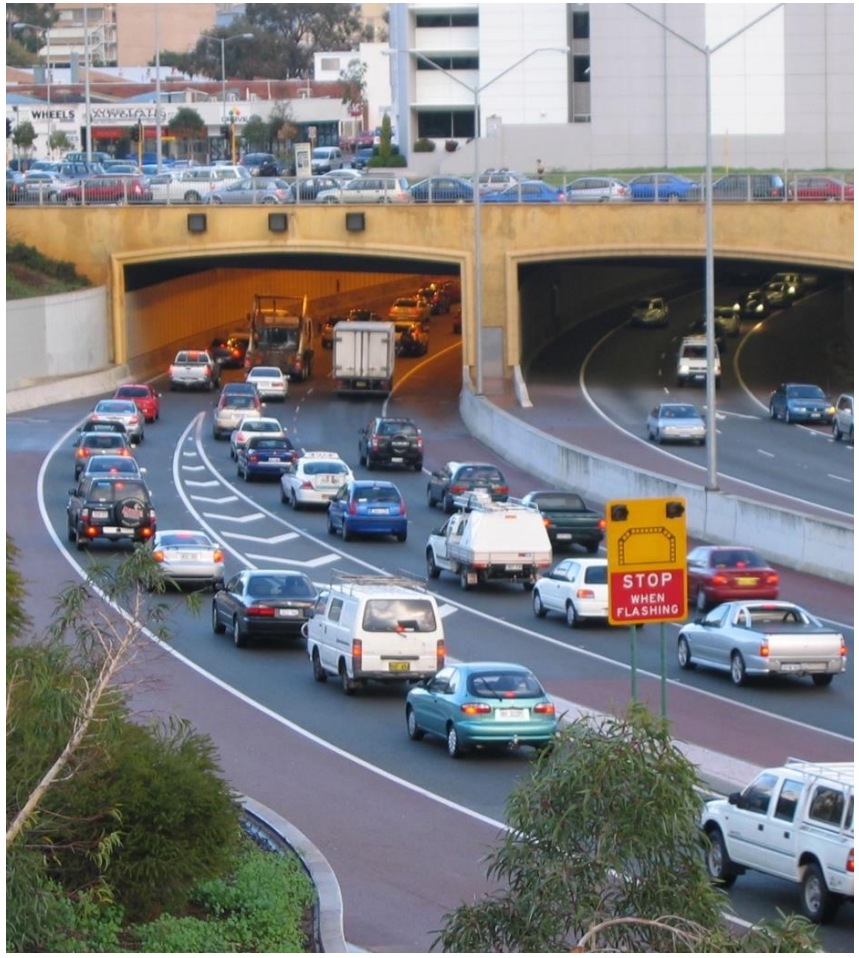
1 3 5 6

7 8 2 4

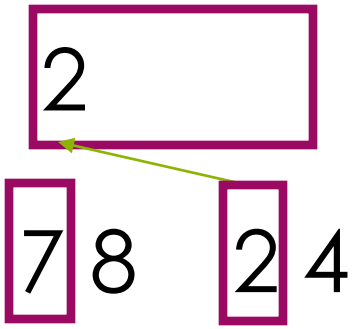


Merge Sort

Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**



1 3 5 6



Merge Sort

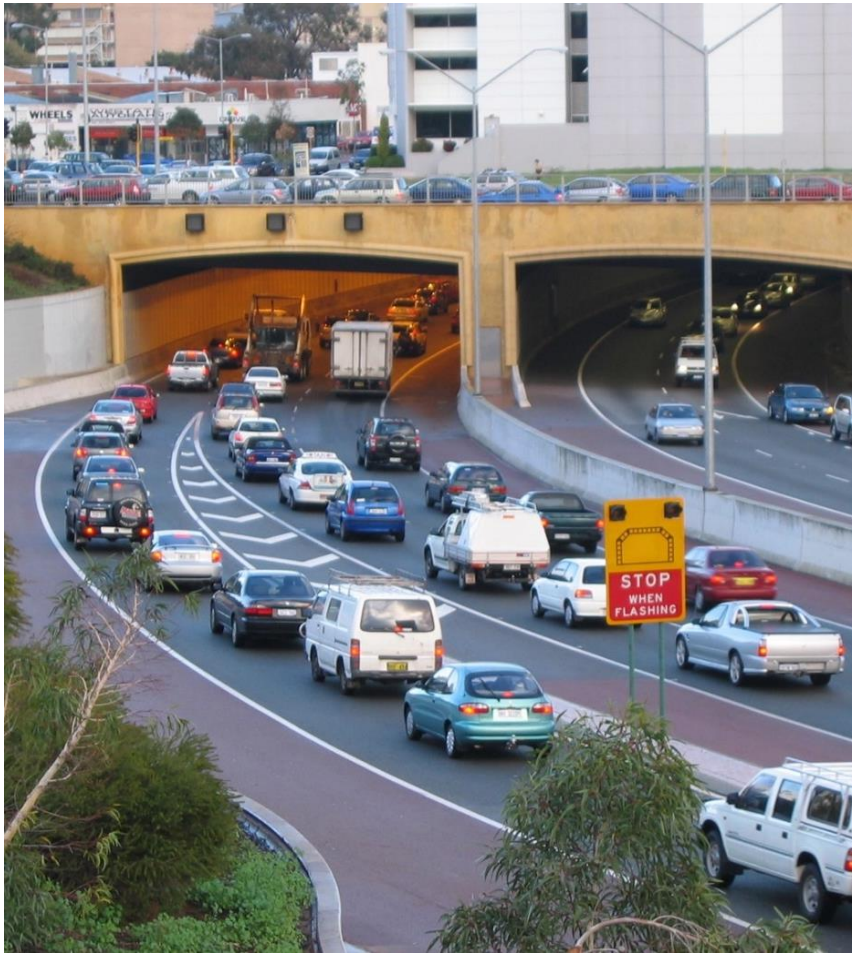
Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**

1 3 5 6

2

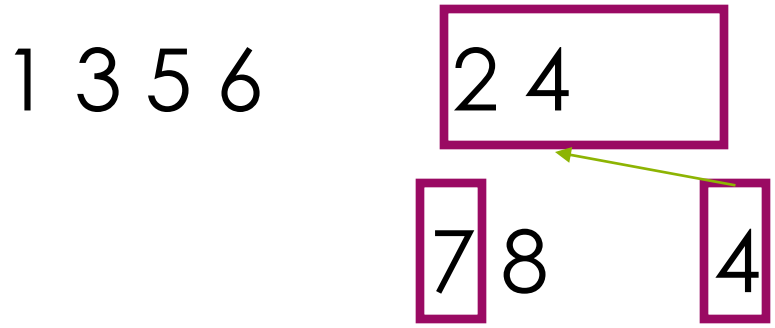
7 8

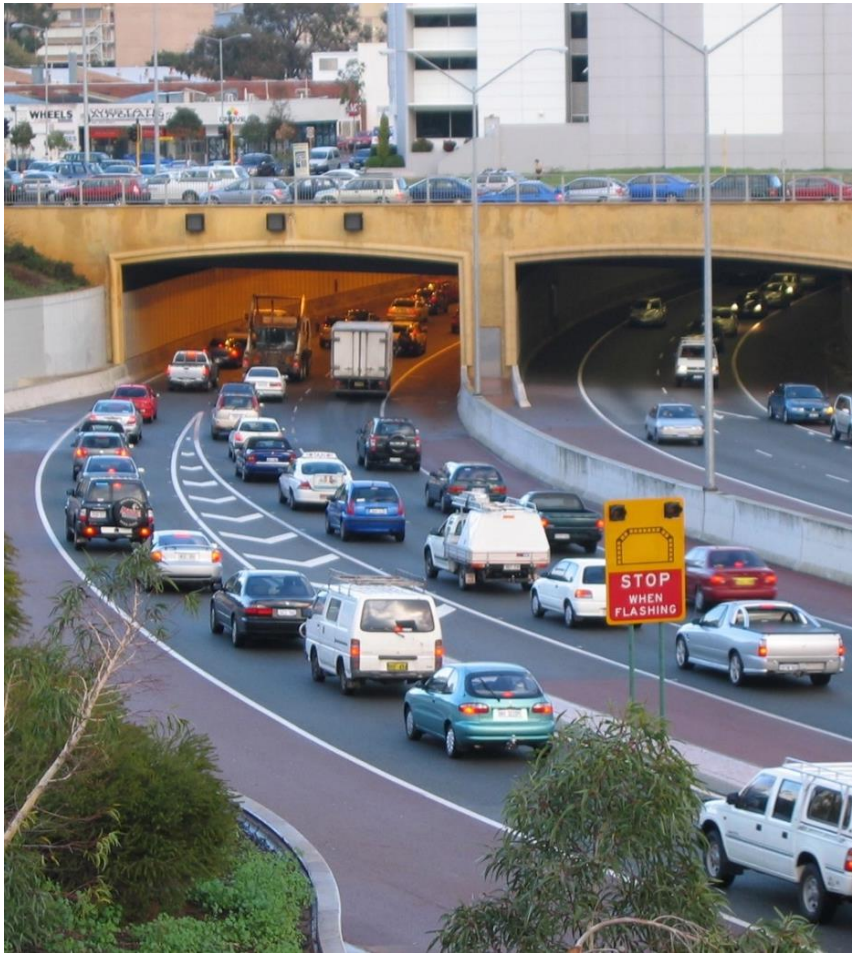
4



Merge Sort

Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**





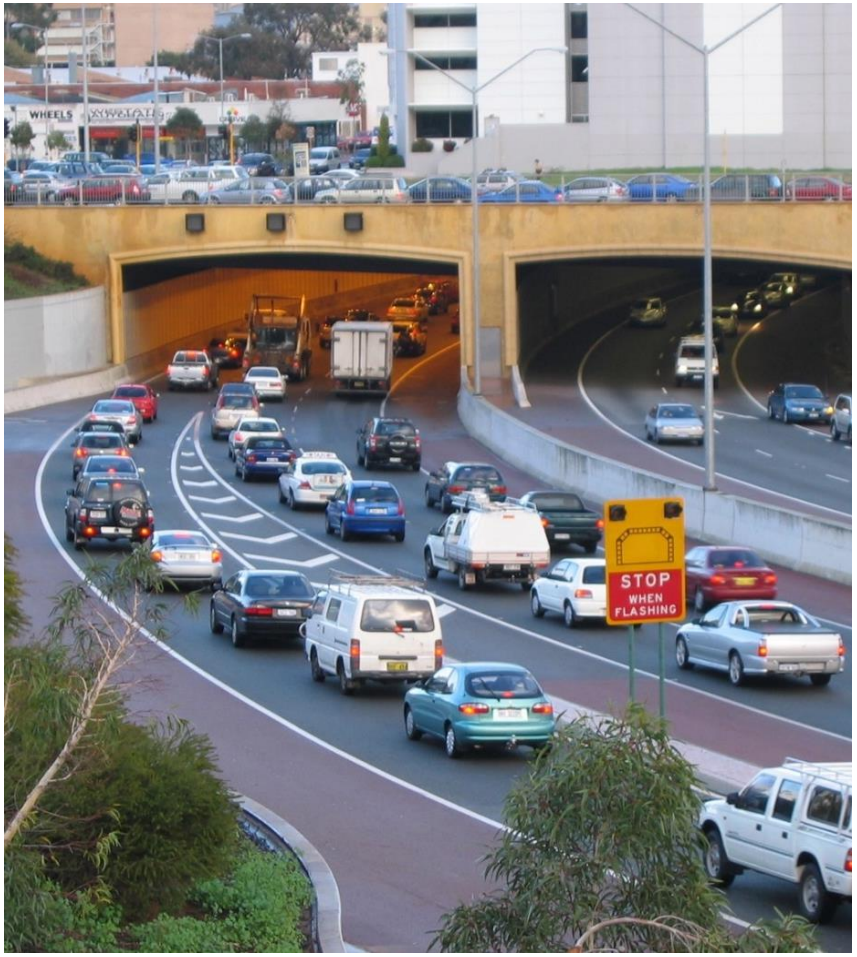
Merge Sort

Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**

1 3 5 6

2 4 7

7 8



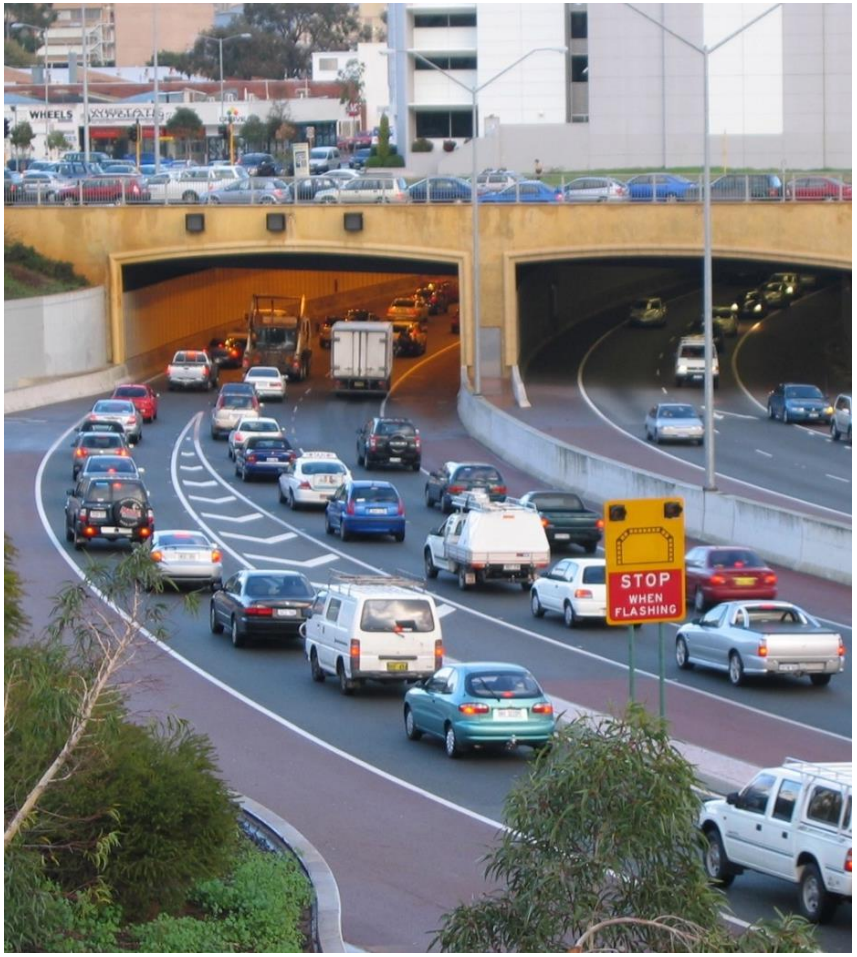
Merge Sort

Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**

1 3 5 6

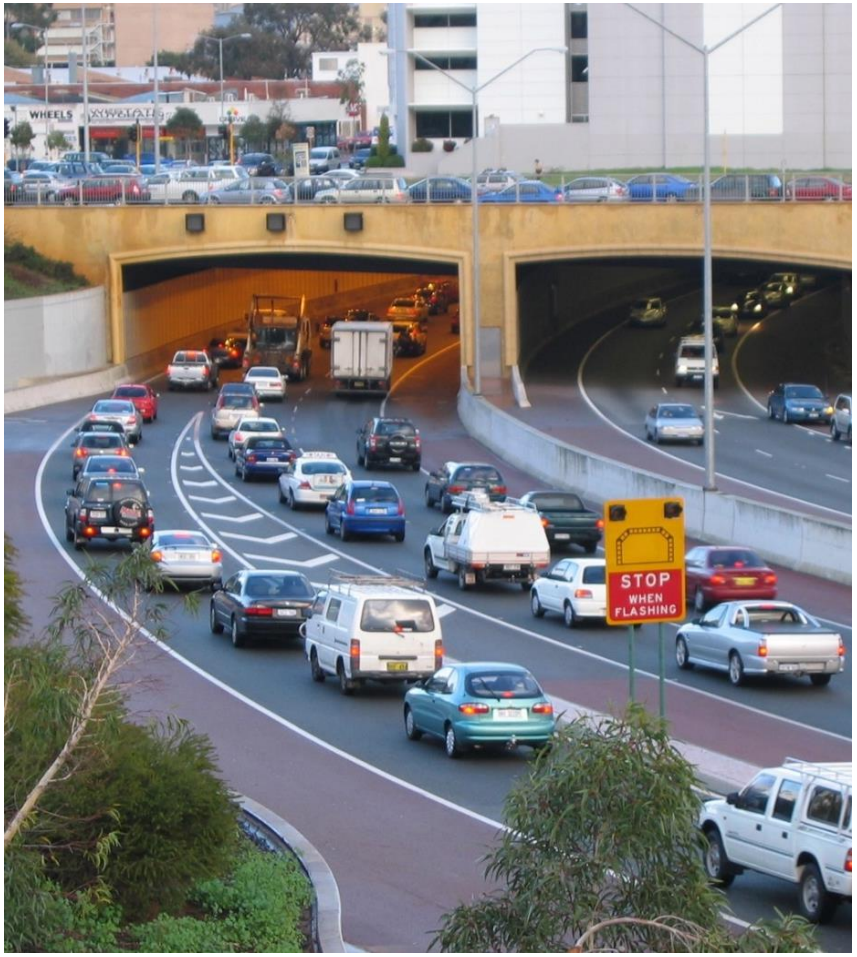
2 4 7 8

8



Merge Sort

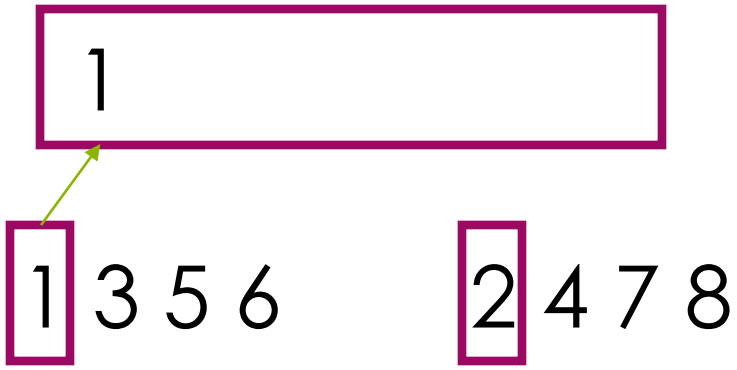
Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**

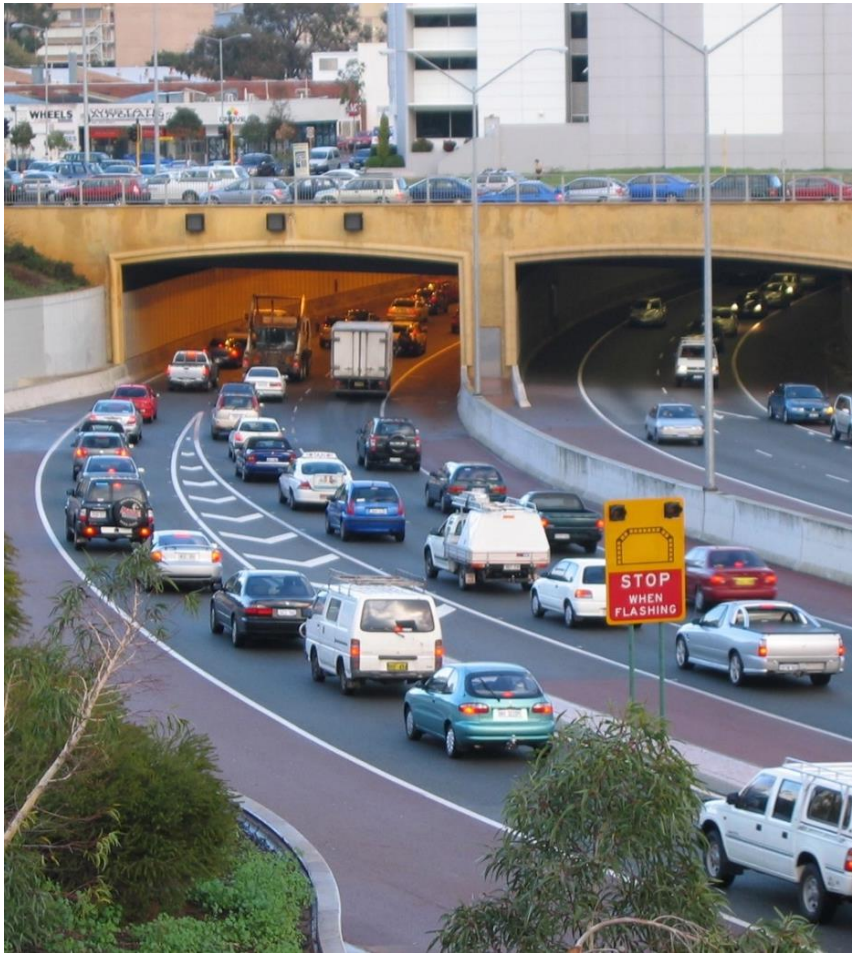


1 3 5 6 2 4 7 8

Merge Sort

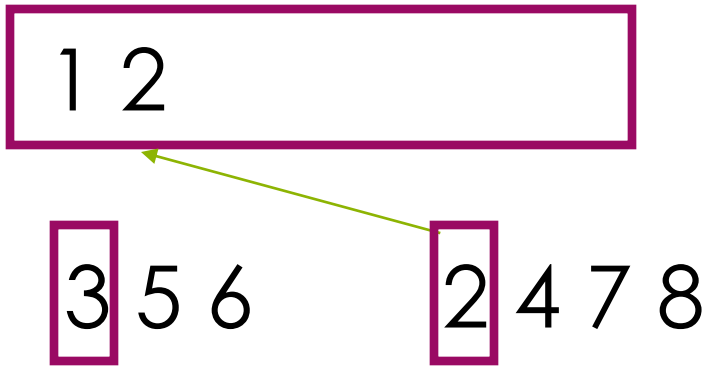
Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**

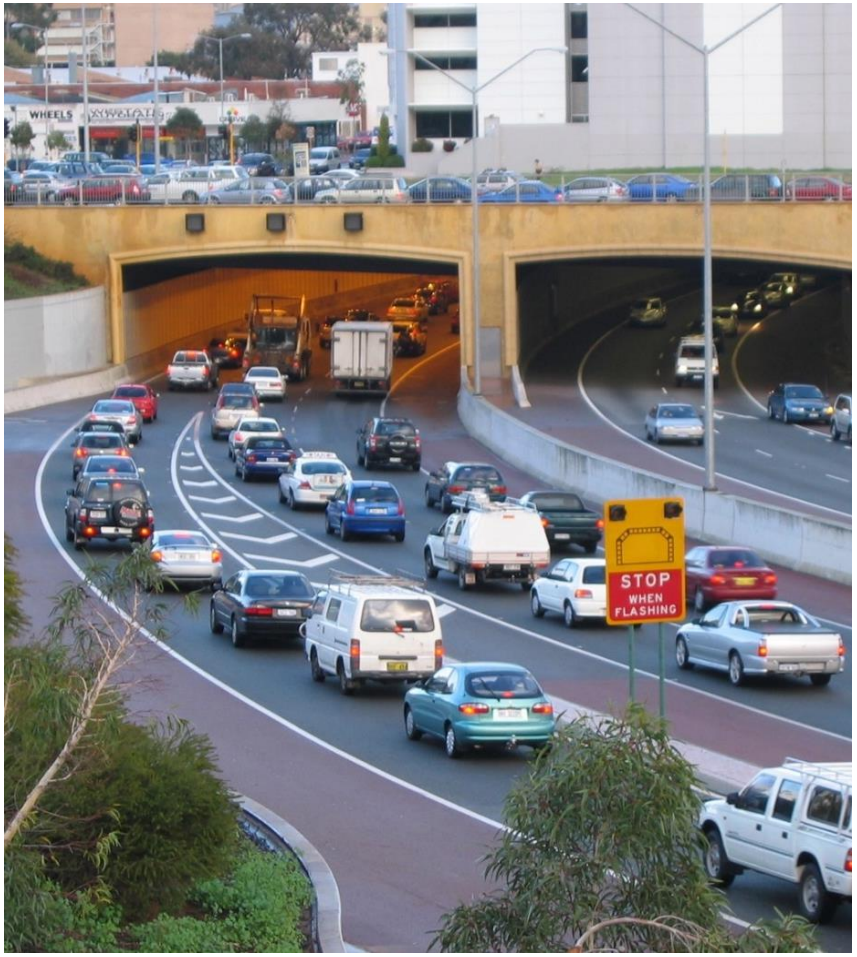




Merge Sort

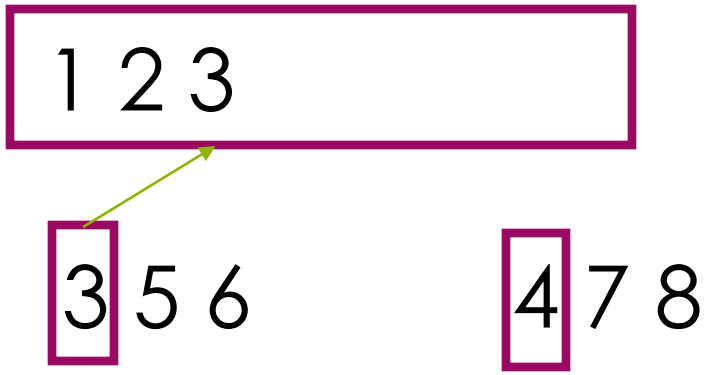
Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**

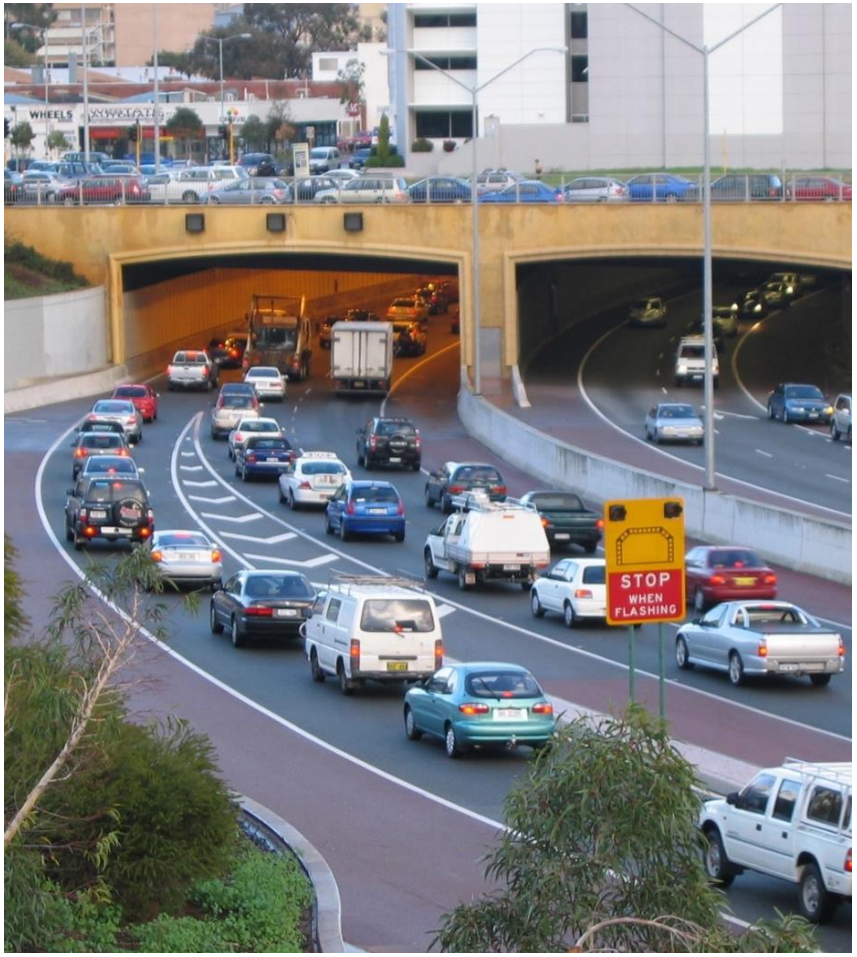




Merge Sort

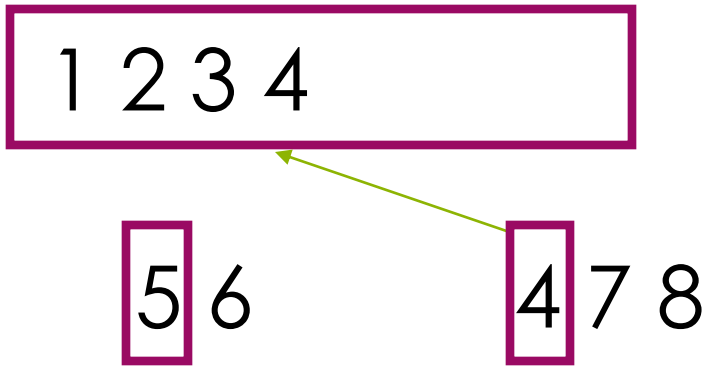
Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**

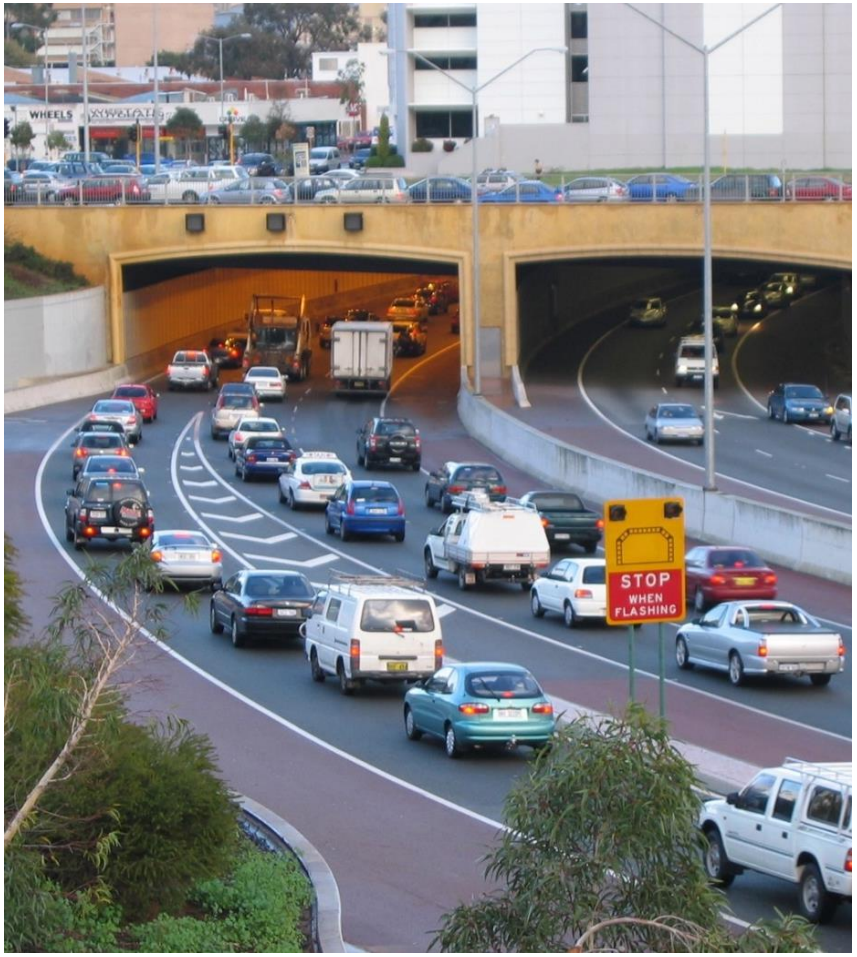




Merge Sort

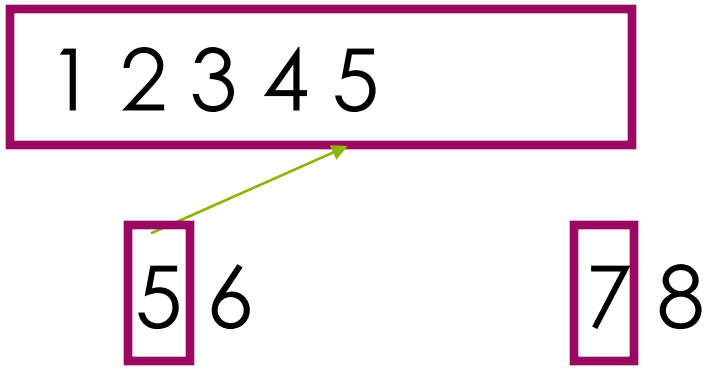
Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**

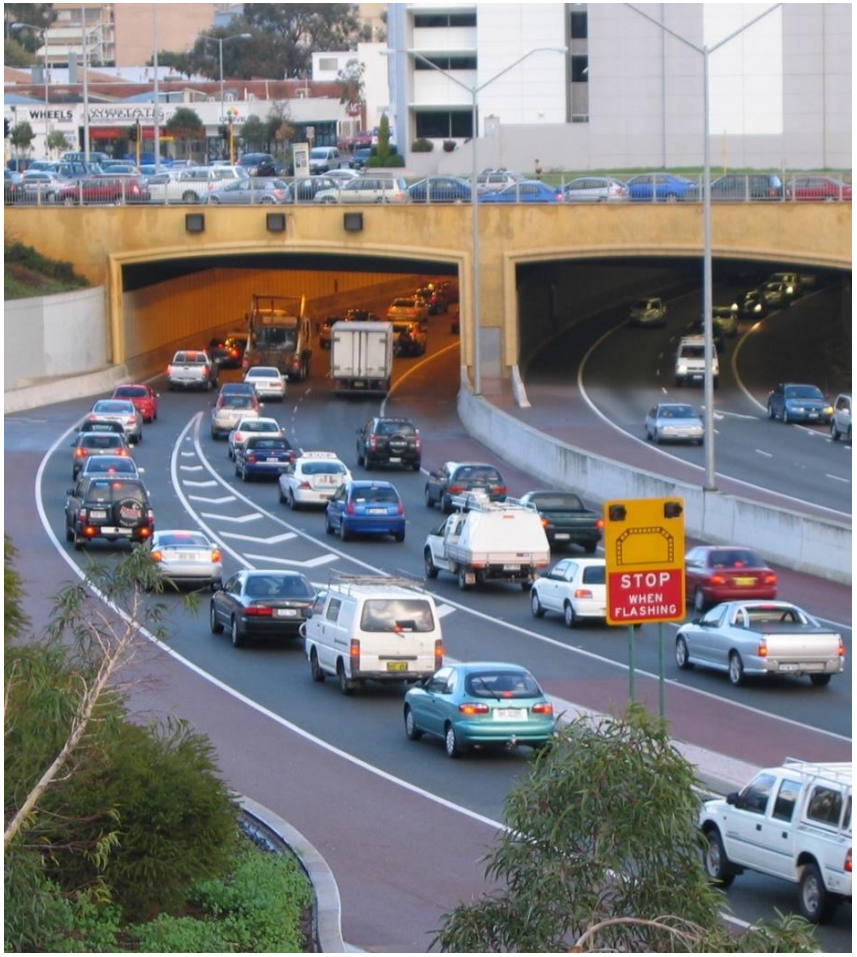




Merge Sort

Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**

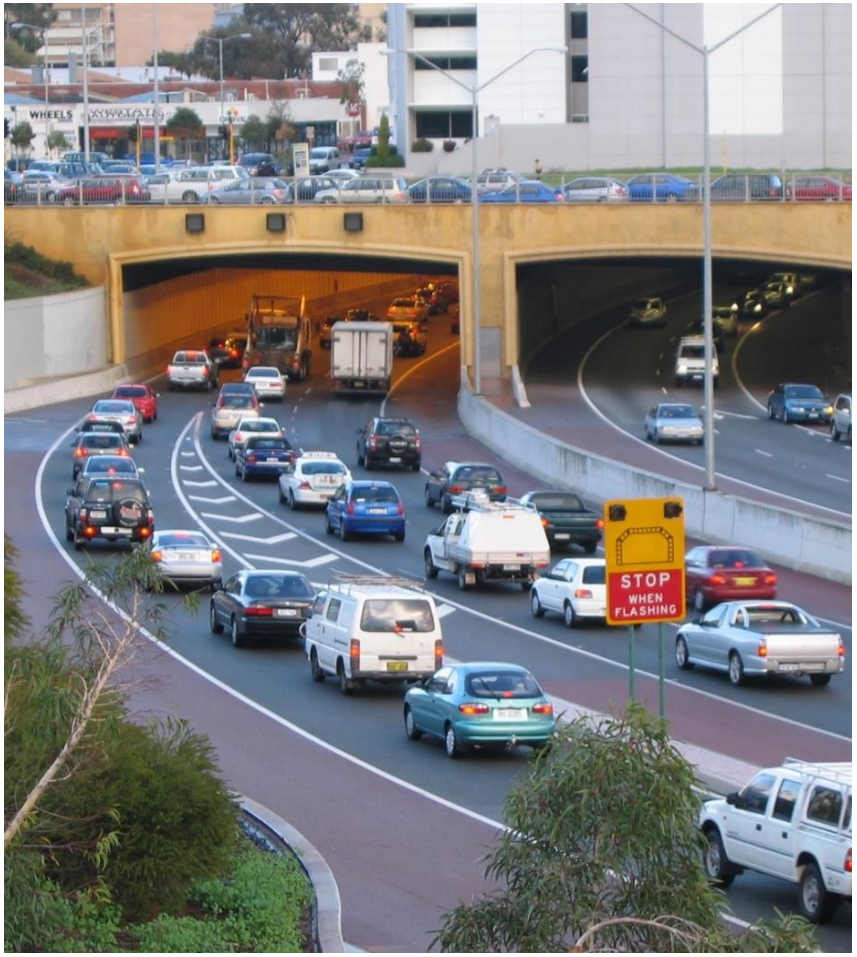




Merge Sort

Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. **Then we merge the lists together making sure the items are in the correct order.**

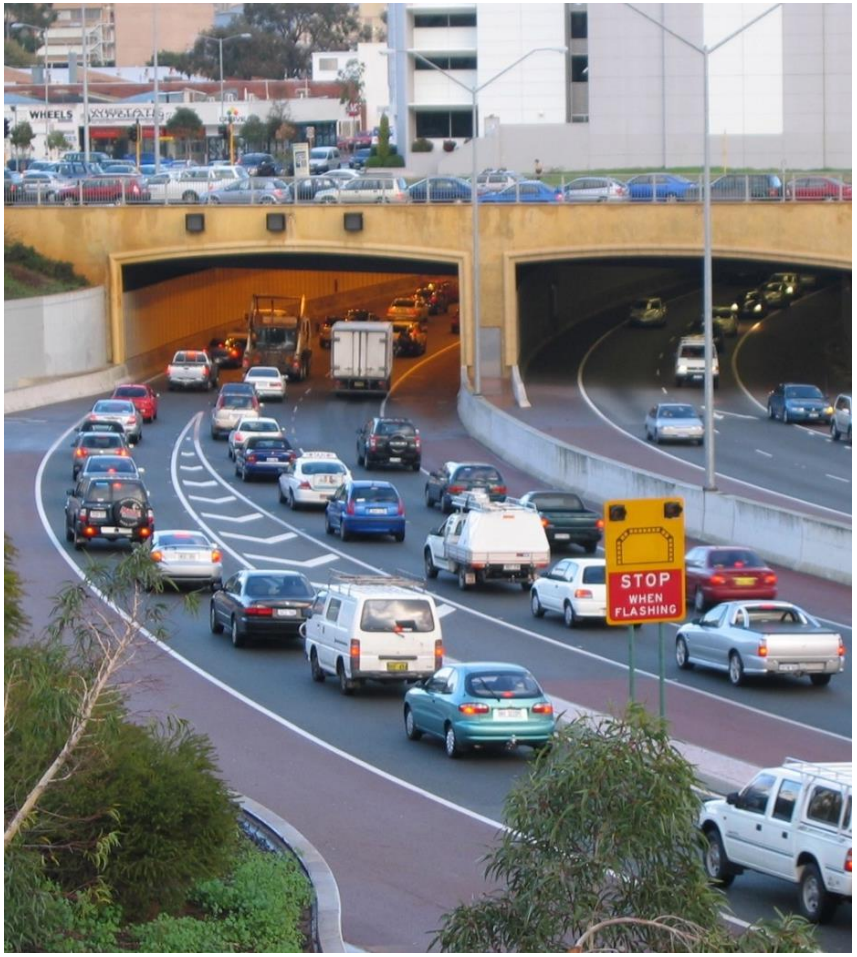
And so on...



Merge Sort

Unlike Bubble Sort, we repeatedly split the lists into smaller and smaller lists until there is only one item in each list. Then we merge the lists together making sure the items are in the correct order.

1 2 3 4 5 6 7 8



Activity: Merge Sort

Using the following numbers, '4 8 2 6 7 3 5 1' perform a merge sort to sort the numbers in a numerical order from smallest to largest. Afterwards write down the steps you performed to complete this task in your workbooks.

The list of numbers to sort is:

4 8 2 6 7 3 5 1

Activity: Merge Sort Solution

4 8 2 6 7 3 5 1

Activity: Merge Sort Solution

4 8 2 6 7 3 5 1

4 8 2 6

7 3 5 1

Activity: Merge Sort Solution

4 8 2 6 7 3 5 1

4 8 2 6 7 3 5 1

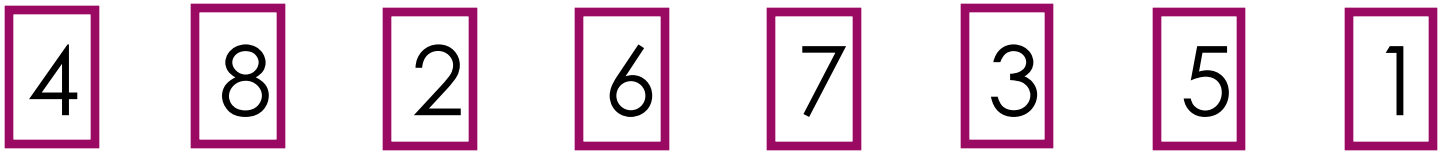
4 8 | 2 6 | 7 3 | 5 1

Activity: Merge Sort Solution

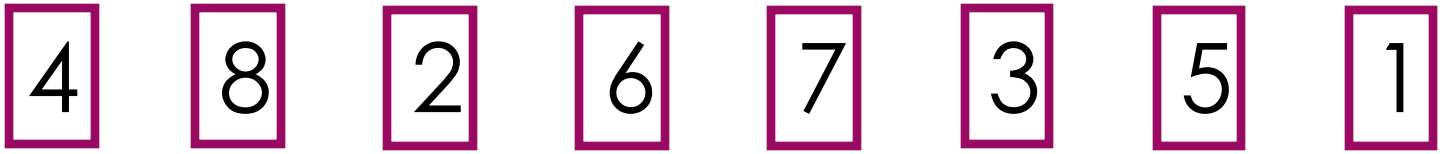
4 8 2 6 7 3 5 1

4 8 2 6 7 3 5 1

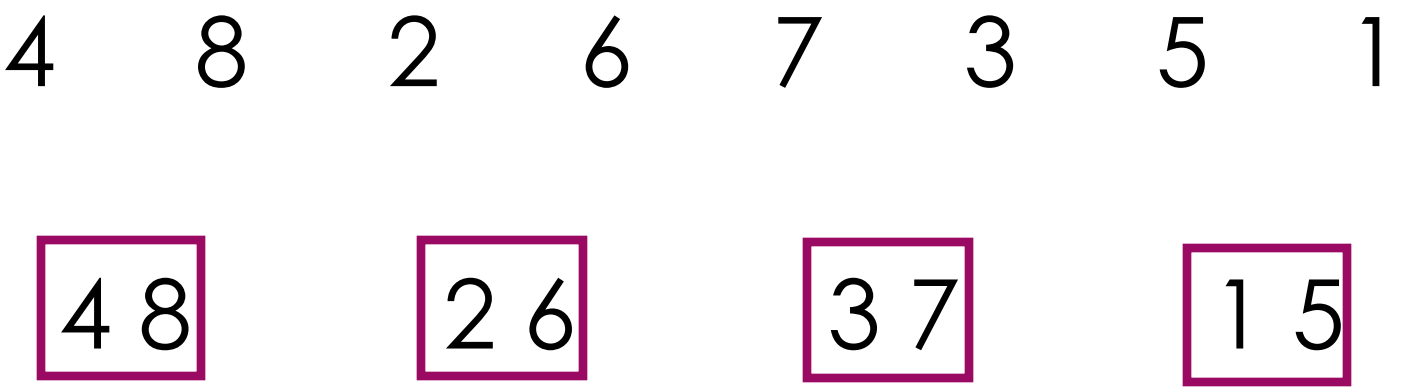
4 8 2 6 7 3 5 1



Activity: Merge Sort Solution



Activity: Merge Sort Solution



Activity: Merge Sort Solution

4 8 2 6 7 3 5 1

4 8 2 6 3 7 1 5

2 4 6 8 1 3 5 7

Activity: Merge Sort Solution

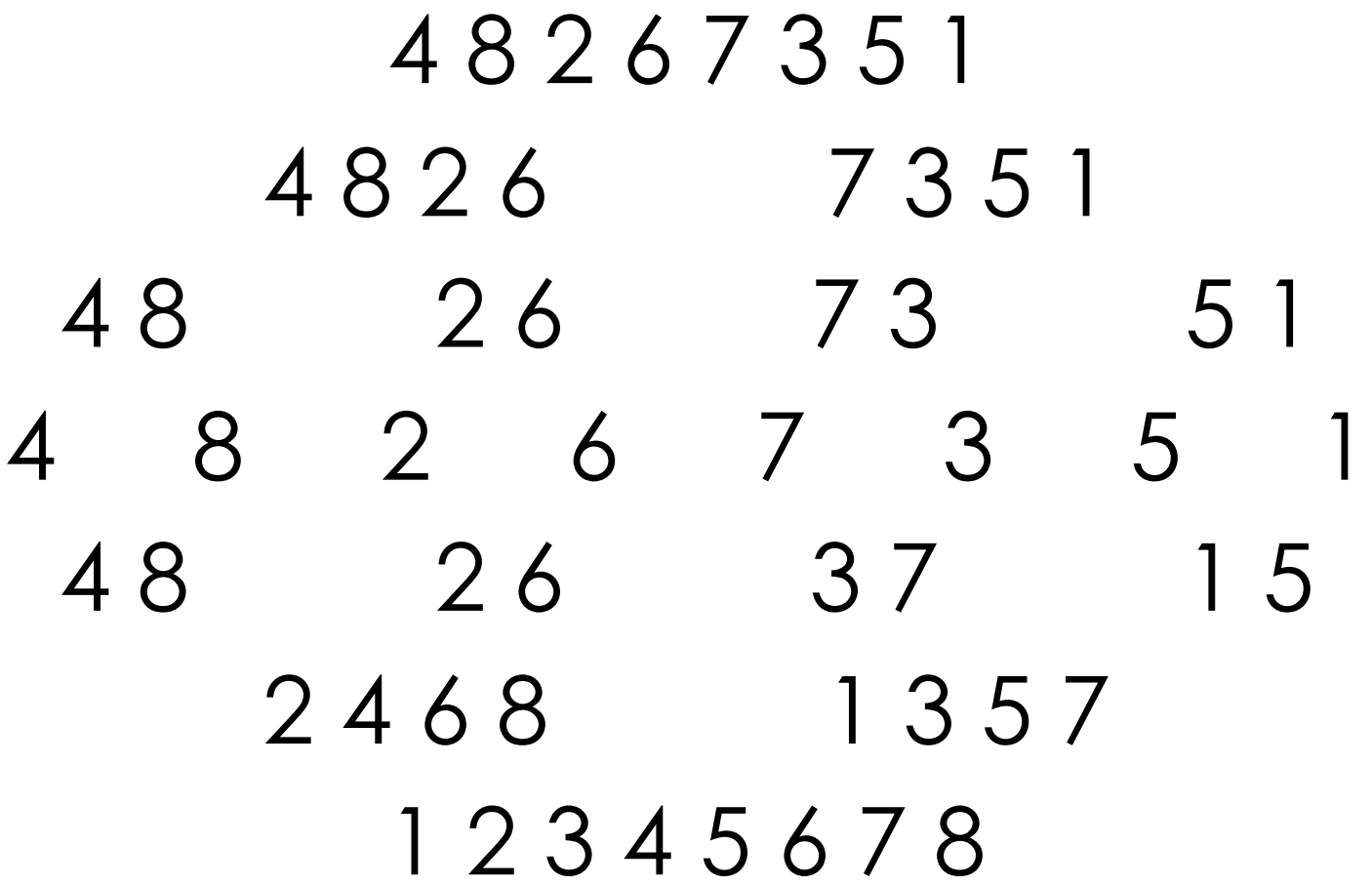
4 8 2 6 7 3 5 1

4 8 2 6 3 7 1 5

2 4 6 8 1 3 5 7

1 2 3 4 5 6 7 8

Activity: Merge Sort Solution



A-Level Activity: Merge Sort in Python

Understanding Required

Bubble Sort Solution – main Function

```
# Bubble sort main

def main():

    unorderedList = [34,23,56,89,23,43,55,75,4,2,6,10,11]

    print(bubbleSort(unorderedList))



main()
```

A-Level – Insertion Sort

Insertion sort is another sorting algorithm

It is less efficient on large datasets than merge sort, but remains good for smaller datasets and is considered simpler to implement

The algorithm works by creating a second, empty list. It then iterates through the original unsorted list and inserts each element into the correct position within the new list.

Insertion Sort

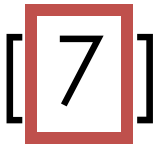
7 4 1 5 8 3 6 2

Insertion Sort

[ ]

Insertion Sort

Insertion Sort



Insertion Sort

Insertion Sort

Insertion Sort

[7 4 1 5 8 3 6 2]

[4 7]

Insertion Sort

[7 4 1 5 8 3 6 2]

[4 7]

Insertion Sort

Insertion Sort

[7 4 1 5 8 3 6 2]

[1 4 7]

Insertion Sort

Insertion Sort

Insertion Sort

Insertion Sort

Insertion Sort

Insertion Sort

Insertion Sort

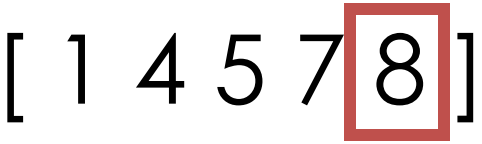
[7 4 1 5 8 3 6 2]

[1 4 5 7]

Insertion Sort

Insertion Sort

Insertion Sort



Insertion Sort

Insertion Sort

Insertion Sort

Insertion Sort

[7 4 1 5 8 3 6 2]

[1 2 3 4 5 6 7 8]

Bubble Sort Solution – main Function

```
# Bubble sort main

def main():

    unorderedList = [34,23,56,89,23,43,55,75,4,2,6,10,11]

    print(insertionSort(unorderedList))



main()
```

Search Algorithms

Linear Search

A linear search is a simple search algorithm where a list is searched until the required value is found.

For example if the value we are looking for is 4.

Linear Search

A linear search is a simple search process where a list is searched until the required value is found.

For example if the required value is 4.

Linear Search

A linear search is a simple search process where a list is searched until the required value is found.

For example if the required value is 4.

Linear Search

A linear search is a simple search process where a list is searched until the required value is found.

For example if the required value is 4.

Linear Search

A linear search is a simple search process where a list is searched until the required value is found.

For example if the required value is 4.

Linear Search

A linear search is a simple search process where a list is searched until the required value is found.

For example if the required value is 4.

Linear Search

A linear search is a simple search process where a list is searched until the required value is found.

For example if the required value is 4.

6 5 3 1 8 7 2 4

Linear Search

A linear search is a simple search process where a list is searched until the required value is found.

For example if the required value is 4.

Activity: Linear Search

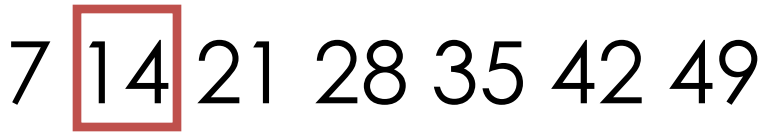
Using the following numbers shown below, perform a linear search to locate the value '**42**'. Afterwards write down the steps you performed to complete this task in your workbooks.

The list of numbers to search through is:

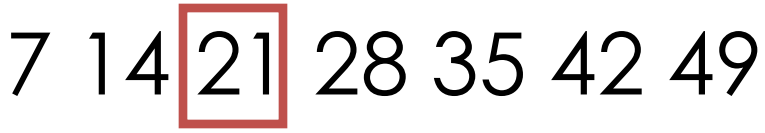
7, 14, 21, 28, 35, 42, 49

Activity: Linear Search Solution

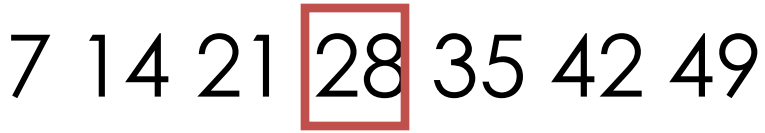
Activity: Linear Search Solution



Activity: Linear Search Solution



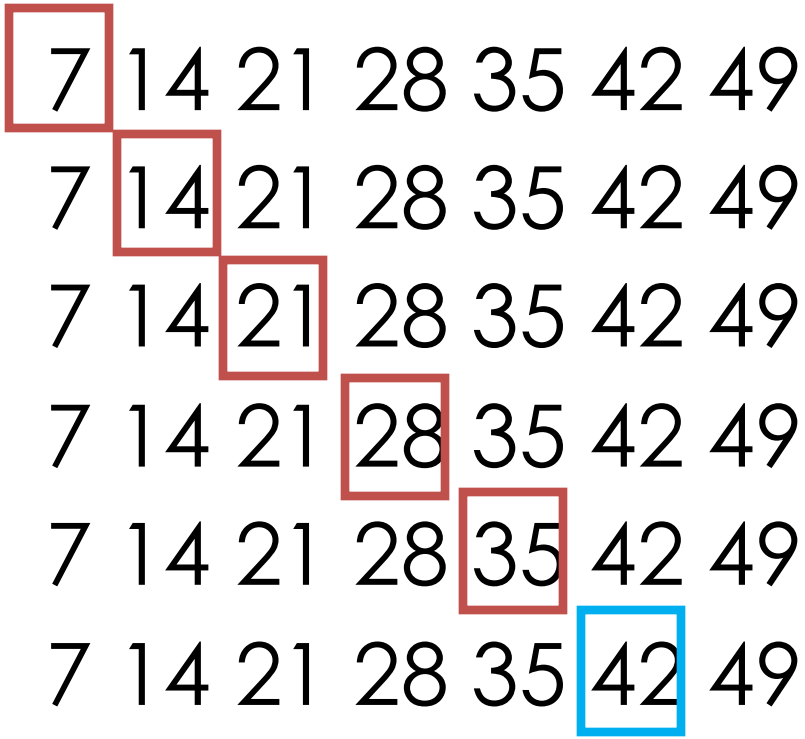
Activity: Linear Search Solution



Activity: Linear Search Solution

Activity: Linear Search Solution

Activity: Linear Search Solution



A Level Activity: Linear Search in Python

Implementation Required

Binary Search Solution – main Function

```
# Binary Search main entry

def main():

    mySortedList                                                    =
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

    itemToFind = 14

    print(linearSearch(mySortedList,itemToFind))



main()
```

Binary Search

The binary search algorithm (also known as a half interval search) works like this:

1. The middle value in a **sorted** list is inspected to see if it matches the search value.
2. If the middle value is greater than the search value, the upper half of the list is discarded. If it is less than the search value, the lower half is discarded.
3. This process is repeated, with the list halving in size each time until the search value is found.

For example if the value we are searching for is **4**.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Binary Search

The binary search algorithm (also known as a half interval search) works like this:

1. The middle value in a **sorted** list is inspected to see if it matches the search value.
2. If the middle value is greater than the search value, the upper half of the list is discarded. If it is less than the search value, the lower half is discarded.
3. This process is repeated, with the list halving in size each time until the search value is found.

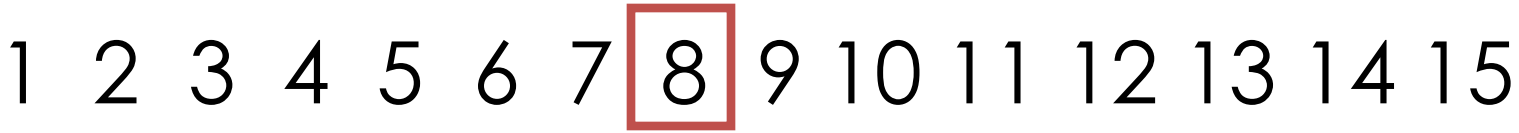
For example if the value we are searching for is **4**.

Binary Search

The binary search algorithm (also known as a half interval search) works like this:

1. The middle value in a **sorted** list is inspected to see if it matches the search value.
2. If the middle value is greater than the search value, the upper half of the list is discarded. If it is less than the search value, the lower half is discarded.
3. This process is repeated, with the list halving in size each time until the search value is found.

For example if the value we are searching for is **4**.



Binary Search

The binary search algorithm (also known as a half interval search) works like this:

1. The middle value in a **sorted** list is inspected to see if it matches the search value.
2. If the middle value is greater than the search value, the upper half of the list is discarded. If it is less than the search value, the lower half is discarded.
3. This process is repeated, with the list halving in size each time until the search value is found.

For example if the value we are searching for is **4**.

Binary Search

The binary search algorithm (also known as a half interval search) works like this:

1. The middle value in a **sorted** list is inspected to see if it matches the search value.
2. If the middle value is greater than the search value, the upper half of the list is discarded. If it is less than the search value, the lower half is discarded.
3. This process is repeated, with the list halving in size each time until the search value is found.

For example if the value we are searching for is **4**.

1 2 3 4 5 6 7

Binary Search

The binary search algorithm (also known as a half interval search) works like this:

1. The middle value in a **sorted** list is inspected to see if it matches the search value.
2. If the middle value is greater than the search value, the upper half of the list is discarded. If it is less than the search value, the lower half is discarded.
3. This process is repeated, with the list halving in size each time until the search value is found.

For example if the value we are searching for is **4**.

Binary Search

The binary search algorithm (also known as a half interval search) works like this:

1. The middle value in a **sorted** list is inspected to see if it matches the search value.
2. If the middle value is greater than the search value, the upper half of the list is discarded. If it is less than the search value, the lower half is discarded.
3. This process is repeated, with the list halving in size each time until the search value is found.

For example if the value we are searching for is **4**.

Activity: Binary Search

Using the following numbers shown below, perform a linear search to locate the value '**35**'. Afterwards write down the steps you performed to complete this task in your workbooks.

The list of numbers to search through are :

7, 14, 21, 28, 35, 42, 49, 56, 63, 70, 77

(remember with a Binary search algorithm the list of values must be sorted first before the search is performed)

Activity: Binary Search Solution

7 14 21 28 35 42 49 56 63 70 77

Activity: Binary Search Solution

7 14 21 28 35 42 49 56 63 70 77

Activity: Binary Search Solution

7 14 21 28 35 42 49 56 63 70 77

Activity: Binary Search Solution

Activity: Binary Search Solution

Activity: Binary Search Solution

Activity: Binary Search Solution

Activity: Binary Search Solution

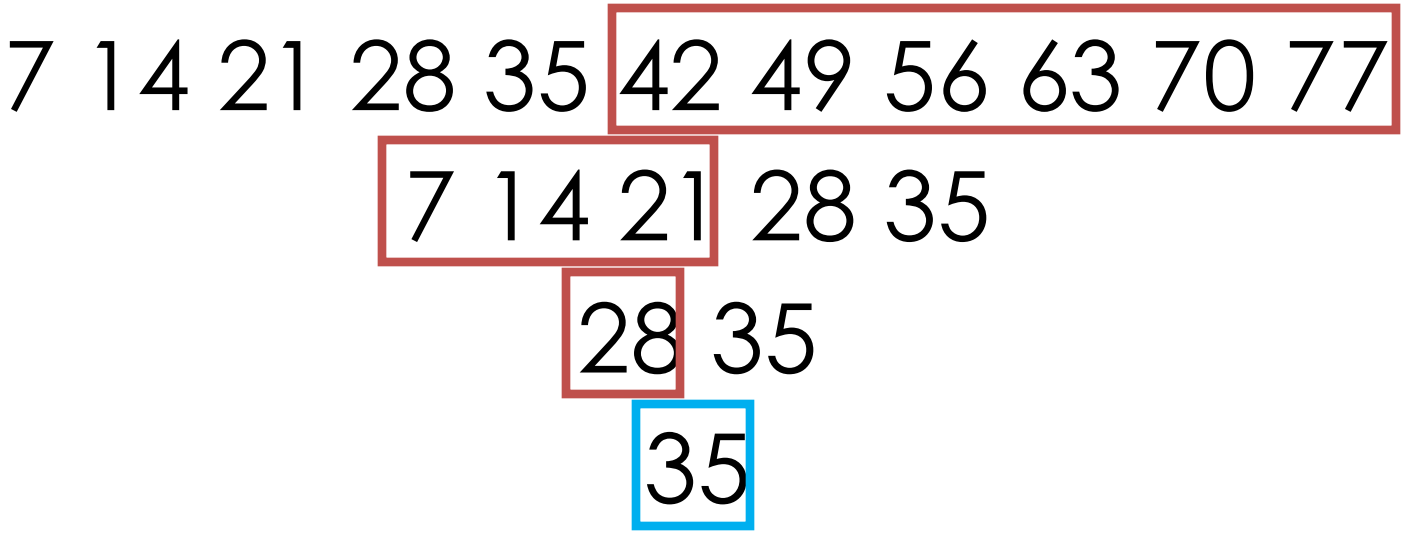
Activity: Binary Search Solution

35

Activity: Binary Search Solution

35

Activity: Binary Search Solution



How a Computer Sees This

A computer doesn't have eyes so it only "sees" one element in the list at a time. Let's look for the number 73 in this sorted list via binary search.

How a Computer Sees This

We check the middle element in the list and compare it to 73.

How a Computer Sees This

5 is less than 73 so we throw away the bottom half of the list.

How a Computer Sees This

We check the middle element in the list and compare it to 73.

How a Computer Sees This

13 is less than 73 so we throw away the bottom half of the list.

How a Computer Sees This

We check the middle element in the list and compare it to 73.

How a Computer Sees This

21 is less than 73 so we throw away the bottom half of the list.

How a Computer Sees This

We check the middle element in the list and compare it to 73.

How a Computer Sees This

34 is less than 73 so we throw away the bottom half of the list.

How a Computer Sees This

We have searched our entire list and haven't found 73 so we can conclude it doesn't exist in our list.

A Level Activity: Binary Search in Python

Implementation Required

Binary Search Solution – main Function

```
# Binary Search main entry

def main():

    mySortedList                                                    =
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

    itemToFind = 7

    print(binarySearch(mySortedList,itemToFind))



main()
```

Binary Search Solution – Search Function

```
# Binary Search
def binarySearch(sortedList, item):
    first = 0
    last = len(sortedList) - 1
    found = False

    while first <= last and not found:
        midpoint = round((first + last) / 2)

        if sortedList[midpoint] == item:
            found = True
        else:
            if item < sortedList[midpoint]:
                last = midpoint - 1
            else:
                first = midpoint + 1
    return found
```