



Sorting Algorithms

GCSE Student Booster

Key Information

- 1) Remember this booster is here to **help you**. Please consider your behaviour in the chat.
- 2) If you are in a room with a teacher/group, please login to the meeting. This is so we can mark your attendance. This information goes into a **prize draw**.
- 3) Make sure the name on the meeting is the **SAME** as the name on your Isaac account. We can't mark you present if they don't match.



Intended Learning Outcomes

By the end of this session, you will be able to:

- Use the key steps to perform a Bubble Sort
- Use the key steps to perform a Merge Sort
- Use the key steps to perform an Insertion Sort
- Explain the efficiency of the methods.



Why sorting algorithms?

Organising data is a crucial aspect of data management. Whether you aim to present information in a particular **sequence** or expedite searches within a dataset, **sorting** is a fundamental task.



Types of Sorting Algorithms

Bubble Sort

Merge Sort

Insertion Sort
(OCR only)



Bubble Sort

1. Compare the first value in the list with the next one up. If the first value is bigger, swap.
2. Move to the second value in the list. Compare value with the next. If bigger, swap.
3. Repeat until there are no more items to compare. Each time the algorithm goes through the list, it is called a **pass**.

Repeat steps 1 to 4 (several passes may be needed to sort a list)



Bubble Sort Example

CardList [0]	CardList [1]	CardList [2]	CardList [3]	CardList [4]	CardList [5]	temp
1	10	9	2	4	5	

Compare the first item in the list with the next. If it is bigger then swap them



Bubble Sort Example

CardList [0]	CardList [1]	CardList [2]	CardList [3]	CardList [4]	CardList [5]	temp
1	10	9	2	4	5	

Compare the second item in the list with the next. If it is bigger than swap them



Bubble Sort Example

CardList [0]	CardList [1]	CardList [2]	CardList [3]	CardList [4]	CardList [5]	temp
1	9	10	2	4	5	

Compare the third item in the list with the next. If it is bigger than swap them



Bubble Sort Example

CardList [0]	CardList [1]	CardList [2]	CardList [3]	CardList [4]	CardList [5]	temp
1	9	2	10	4	5	

Compare the fourth item in the list with the next. If it is bigger than swap them



Bubble Sort Example

CardList [0]	CardList [1]	CardList [2]	CardList [3]	CardList [4]	CardList [5]	temp
1	9	2	4	10	5	

Compare the fifth item in the list with the next. If it is bigger than swap them



Bubble Sort Example

CardList [0]	CardList [1]	CardList [2]	CardList [3]	CardList [4]	CardList [5]	temp
1	9	2	4	5	10	

Pass 1

Pass one complete. The list is not order so repeat the process



Bubble Sort Example

CardList [0]	CardList [1]	CardList [2]	CardList [3]	CardList [4]	CardList [5]
1	9	2	4	5	10

Pass 1

temp

1

9

2

4

5

10

Pass two. Compare the first item in the list with the next. If it is bigger than swap them

Bubble Sort Example

CardList [0]	CardList [1]	CardList [2]	CardList [3]	CardList [4]	CardList [5]
1	9	2	4	5	10

Pass 1

temp

1

9

2

4

5

10

Pass two. Compare the second item in the list with the next. If it is bigger than swap them

Bubble Sort Example

CardList [0]	CardList [1]	CardList [2]	CardList [3]	CardList [4]	CardList [5]
1	9	2	4	5	10

Pass 1

temp

1

2

9

4

5

10

Pass two. Compare the third item in the list with the next. If it is bigger than swap them

Bubble Sort Example

CardList [0]	CardList [1]	CardList [2]	CardList [3]	CardList [4]	CardList [5]
1	9	2	4	5	10
1	2	4	9	5	10

Pass 1

temp

Pass two. Compare the fourth item in the list with the next. If it is bigger than swap them

Bubble Sort Example

CardList [0]	CardList [1]	CardList [2]	CardList [3]	CardList [4]	CardList [5]
1	9	2	4	5	10
1	2	4	5	9	10

Pass 1

temp

Pass two. Compare the fifth item in the list with the next. If it is bigger than swap them

Bubble Sort Example

CardList [0]	CardList [1]	CardList [2]	CardList [3]	CardList [4]	CardList [5]
1	9	2	4	5	10
1	2	4	5	9	10
1	2	4	5	9	10

Pass 1

temp

Pass 2

Pass 3

Bubble Sort Task

Complete activities in "Handout 1 Task 1"

Sorting Algorithms

Handout 1 – Sorting Algorithms

Task 1 – Bubble sort

- For each of the below, show the stages of the bubble sort when applied to the data sets

Chris	Ben	Fran	Vishwa	Neil	Lauran
Pass 1					



Isaac Computing

Link to Bubble Sort Algorithm topic on the Isaac Computer Science website

[Bubble sort link on Isaac](#)



Pseudocode – Bubble sort

```

1 PROCEDURE bubble_sort_version_1(items)
2
3   // Initialise the variables
4   num_items = LEN(items)
5
6   // Pass through the array of items n-1 times
7   FOR pass_num = 1 TO num_items - 1
8     // Perform a pass
9     FOR index = 0 TO num_items - 2
10      // Compare items to check if they are out of order
11      IF (items[index] > items[index + 1]) THEN
12        // Swap the items
13        temp = items[index]
14        items[index] = items[index + 1]
15        items[index + 1] = temp
16      ENDIF
17    NEXT index
18  NEXT pass_num
19 ENDPROCEDURE

```



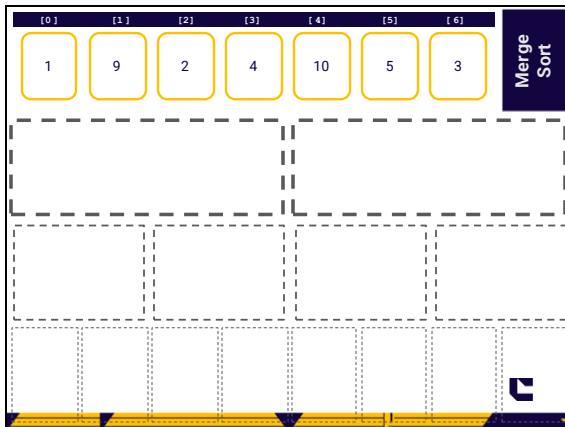
Merge Sort

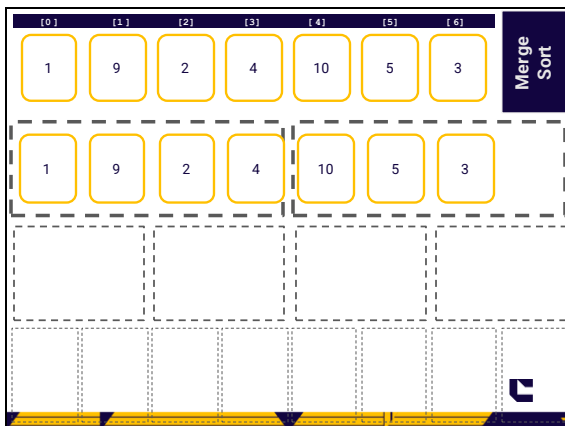
Merge sort involves splitting and merging:

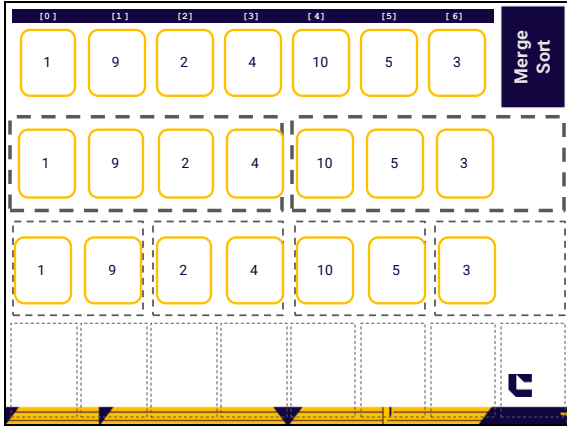
1. Divide a list into sublists repeatedly, breaking them down to single-item.
2. Pair and merge these single items with ordered items.
3. Repeat the process until a fully sorted list is achieved.

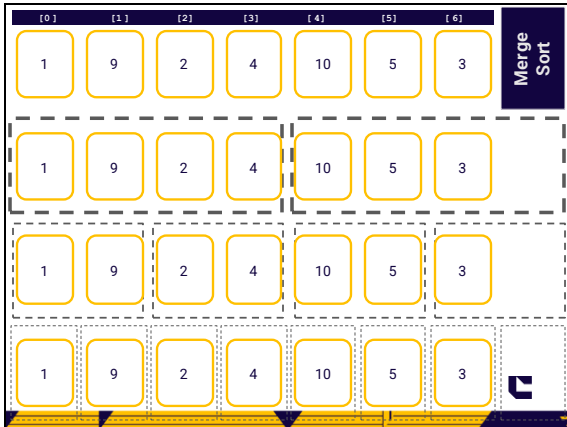
Merge sort demonstrates a "divide and conquer" problem-solving approach.

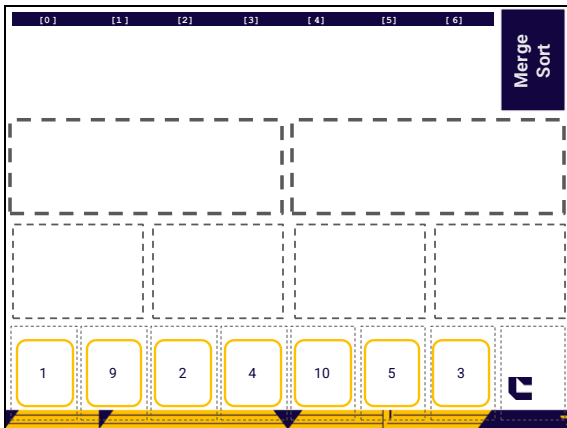


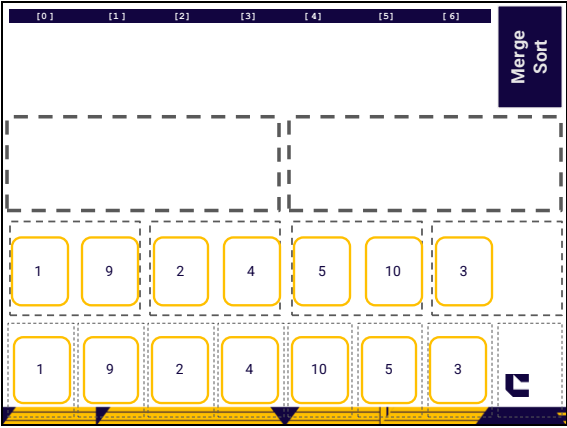


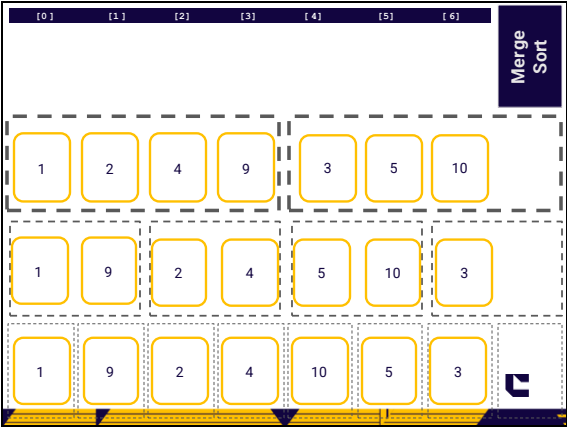


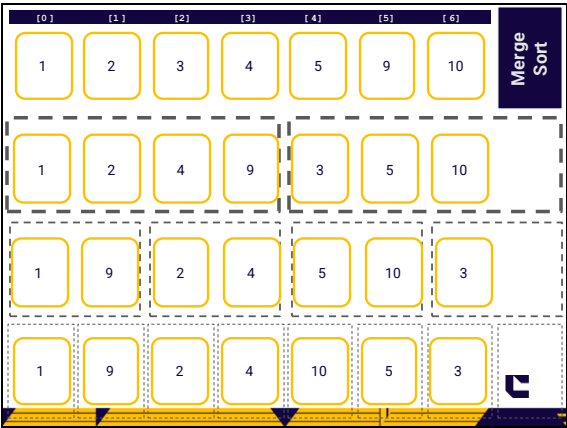












Merge Sort Task

Handout 1 Task 2

Task 2 - Merge sort

1. Show the steps that a Merge sort would take to put the following names into ascending alphabetical order (from A to Z).

Chris	Ben	Fran	Vienna	Ned	Lauran
-------	-----	------	--------	-----	--------



Isaac Computing

Link to Merge Sort Algorithm topic on the Isaac Computer Science website

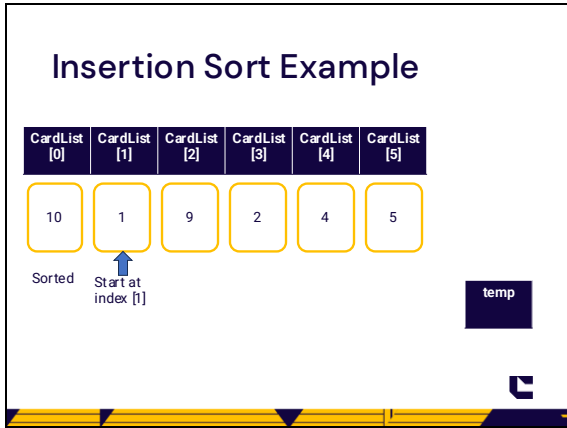
[Merge sort link on Isaac](#)

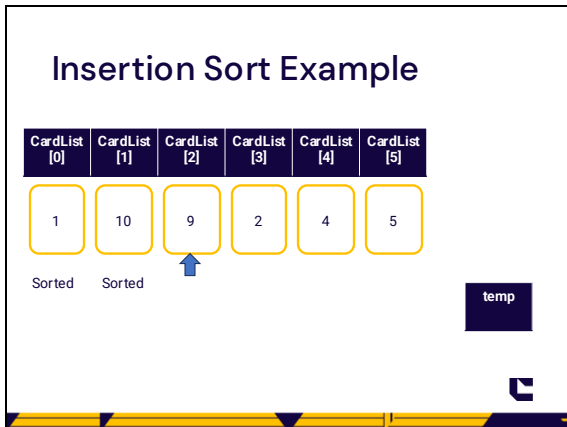


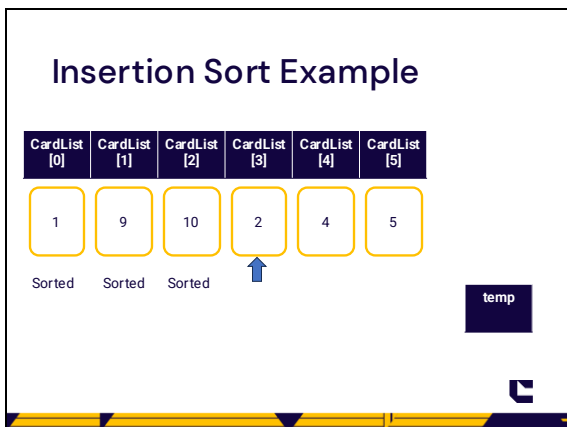
Insertion Sort

- Insertion sort builds a sorted sublist at the start, initially with just the first item.
- The rest of the list remains unsorted.
- The algorithm goes through the unsorted part, putting each item in its correct position within the sorted sublist.









Insertion Sort Example

CardList [0]	CardList [1]	CardList [2]	CardList [3]	CardList [4]	CardList [5]
1	2	9	10	4	5
Sorted	Sorted	Sorted	Sorted		

↑ temp

Insertion Sort Example

CardList [0]	CardList [1]	CardList [2]	CardList [3]	CardList [4]	CardList [5]
1	2	4	9	10	5
Sorted	Sorted	Sorted	Sorted	Sorted	

↑ temp

Insertion Sort Example

CardList [0]	CardList [1]	CardList [2]	CardList [3]	CardList [4]	CardList [5]
1	2	4	5	9	10
Sorted	Sorted	Sorted	Sorted	Sorted	Sorted

temp

Insertion Sort Task

Handout 1 Task 3

Task 3 – Insertion sort

1. Show the steps that an Insertion Sort would take to put the following list into ascending alphabetical order (from A to Z).

sorted		unsorted					
index	0	1	2	3	4	5	6
value	Sydney	Salt Lake City	Athens	Turin	Beijing	Vancouver	London



Isaac Computing

Link to Insertion Sort Algorithm topic on the Isaac Computer Science website

[Insertion sort link on Isaac](#)



Pseudocode

```

1 PROCEDURE insertion_sort(items)
2
3   // Initialise the variables
4   num_items ← len(items)
5
6   // Repeat for each item in the list, starting at the second item
7   FOR index ← 1 TO num_items - 1
8     // Get the value of the next item to insert
9     item_to_insert ← items[index]
10
11    // Get the current position of the last sorted item
12    position ← index - 1
13
14    // Repeat while there are still items in the list to check
15    // and the current sorted item is greater than the item to insert
16    WHILE position >= 0 AND items[position] > item_to_insert
17
18      // Copy the value of the sorted item up one place
19      items[position + 1] ← items[position]
20
21      // Get the position of the next sorted item
22      position ← position - 1
23    ENDWHILE
24
25    // Copy the value of the item to insert into the correct position
26    items[position + 1] ← item_to_insert
27  NEXT index
28 ENDPROCEDURE

```



Comparisons

Choosing a sorting algorithm depends on:

- efficiency with large datasets
- memory usage.

Each sorting algorithm can organise items in a list either from the lowest to the highest value or vice versa.



Insertion vs Bubble sort

Efficiency:

- Insertion sort usually has fewer comparisons
- Insertion sort is generally faster, particularly for larger datasets.

Memory Usage:

- Both algorithms operate without significant additional memory usage.

Item Manipulation:

- Only the value of a single item is copied when inserting or swapping items.



Merge vs Bubble sort

- Merge sort is significantly faster, especially for large and less ordered lists.

Memory Usage:

- Merge sort consumes additional memory during splitting and merging.
- Bubble sort minimally impacts memory, as swaps occur within the original list.



Comparison task

Handout 1 Task 4

Use the links to see the algorithms in action.

Task 4 – Comparison

Insertion Sort http://tiny.cc/insert-anl	Bubble Sort http://tiny.cc/bubble-anl	Merge Sort http://tiny.cc/merge-anl



Sorting Out Sorting

Which algorithm is best?

Complete the comparison table

use the links in the worksheet to see the algorithms in action.

Task 4 – Comparison

Insertion Sort http://tiny.cc/insert-anl	Bubble Sort http://tiny.cc/bubble-anl	Merge Sort http://tiny.cc/merge-anl
Efficient for smaller data sets, but not on larger lists. Adaptive, reduces the total number of steps if the list is partly sorted. Sorts the array by placing one at a time into the correct place on the (left hand) side of the list. Next item is compared to the sorted array on the left and this is repeated until the first item has been placed. References to left hand side are as a human sees it, but the computer just looks at each value and compares with the lowest values that have already been sorted.	Large values are always sorted first. Only takes one iteration to check the list is sorted. Starts at the beginning of the list, compares the first pair of values and swaps the pair of values if needed. Works its way along the list comparing each pair of values. When all the pairs of values have been compared, that is the end of a pass. Multiple passes are made until the sort is complete. The final pass will be when no swaps are made which means the list is in order.	Separates a list into two smaller lists and then keeps repeating this until each list has only one value in it. Pairs of lists with one value are combined and sorted into order to make a sorted list of two values. This then repeats for a pair of lists of two values to make a sorted list of four values. The process above repeats until all values are in one sorted list. Divide and Conquer algorithm. Breaks down a problem into sub-problems. Solutions to the sub-problems are then combined.



Isaac Computer Science

Complete "Handout 1 Task 5 – Gameboard"

<https://isaacscs.org/assignment/7a4e8d22-8638-4e3c-b3d8-833205bbd057>



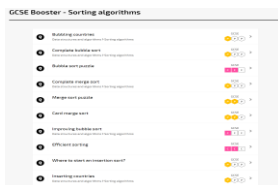
Extension - Questions

- 1) If you have a bubble sort which takes 3 passes to get the numbers in order. How many passes will it take in total?
- 2) When might a bubble sort be a good algorithm to use when sorting data?
- 3) In a bubble sort is has a temp variable. Why is this needed?
- 4) Why is a merge sort a more efficient algorithm to use if you have a large number of items?
- 5) How does having more cores make a merge sort more efficient?



Isaac Gameboard practice

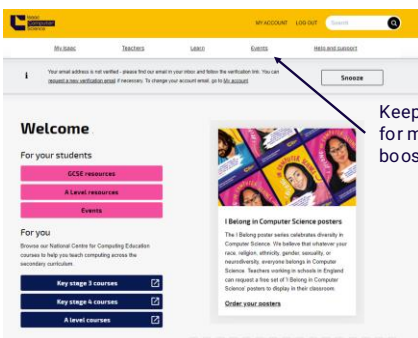
- If you want more sorting algorithms practice, then try this gameboard.
- You will need to sign in to **Isaac Computer Science** or register for a free account if not done already.



 ncce.io/isc-sort 



Check for more ISAAC boosters



Keep an eye out
for more student
booster events



Intended Learning Outcomes

By the end of this session, you will be able to:

- Use the key steps to perform a Bubble Sort
- Use the key steps to perform a Merge Sort
- Use the key steps to perform an Insertion Sort
- Explain the efficiency of the methods.



Isaac Computing

Link to Sorting Algorithms topic on the Isaac Computer Science website

<https://isaacomputerscience.org/topics/sorting?examBoard=all&stage=all>



"Anyone who has never made a mistake has never tried anything new."

Albert Einstein



Thank you