# Understanding Different Types of Algorithms

Anushka Tonapi

In an earlier article published in the November 2024 issue [1], we explored the concept of algorithms and their significance and looked at different ways that we use them in everyday life. In this article, we will explore different types of algorithms and suggest activities to demonstrate these in the classroom.

An algorithm is a list of instructions that help you solve a problem or complete a task. We have discussed that there are multiple algorithms that can be used to solve a problem. But did you know that there are many different *types* of algorithms that use various methods to solve a problem? Some help you make choices, others help you find the fastest path, and some put things in the correct order. In this article, we shall explore the most common types of algorithms in fun and interactive ways.

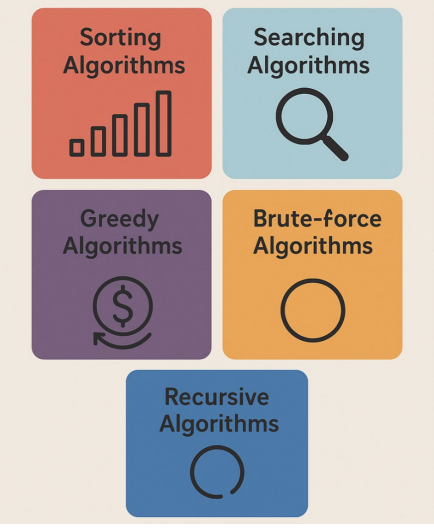


Figure 1 (Source: AI)

## 1. Sorting algorithms: Putting things in order

Let's start with something we all do - organizing! Sorting algorithms help us arrange items in a particular order, such as smallest to largest, or alphabetically.

**Why sorting is important:** When you arrange your books by height, your clothes by colour, or your crayons in a rainbow pattern, you are using sorting. Computers use sorting to arrange emails by date, numbers in a spreadsheet, or high scores in games, for example. In Figure 2, you can see six crayons sorted in the same order of the colours in the rainbow.

Figure 2

*Keywords: Computational Thinking, Processes, Searching, Sorting, Algorithms*

There are many different ways of sorting things, with the following list of sorting algorithms:

- **Bubble Sort:** Compare pairs of items and keep swapping them if they're in the wrong order. The biggest item "bubbles up" to the end in each round. For example, suppose we have the unsorted list: 1, 3, 4, 5, 2; and we want to arrange them in ascending order. Then we consider the first two numbers: 1 and 3. Since they are in order already, we aren't going to swap them. Similarly, we don't swap the next two numbers 3 and 4 either, or even 4 and 5. But 5 and 2 are not in order, so we swap 5 and 2 with each other to get 1, 3, 4, 2, 5. Once again, we swap 4 and 2 (since they are not in order) to get 1, 3, 2, 4, 5. Finally, we swap 3 and 2 to get 1, 2, 3, 4, 5. Notice how the number 2 "bubbles up" towards the left and 5 (the biggest number) "bubbles down" to the end.
- **Selection Sort:** Search the whole list for the smallest item and place it at the beginning. Then search for the next smallest, and so on. We consider our old example but jumbled a bit differently: 5, 3, 4, 1, 2. The smallest item is 1, so we place it in the left-most position 1, 5, 3, 4, 2. Then the next smallest item among the remaining items is 2, which goes next to 1: 1, 2, 5, 3, 4. Similarly, 3 and 4 also get placed in their respective places until we obtain the fully sorted list: 1, 2, 3, 4, 5.
- **Insertion Sort:** Insert each item into its correct place in an already sorted list. Such as putting your cards in order as they are dealt out during a card game. Suppose we have the same numbers as earlier, but now in the order 3, 1, 4, 2, 5.

**Insertion Sort: Putting Mixed Cards in Order!**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Step 1 | 3 | | | | 1 | 4 | 2 | 5 |
| Step 2 | 1 | 3 | | | 4 | 2 | 5 | |
| Step 3 | 1 | 3 | 4 | | 2 | 5 | | |
| Step 4 | 1 | 2 | 3 | 4 | 5 | | | |
| Step 5 | 1 | 2 | 3 | 4 | 5 | | | |

Figure 3: Insertion Sort: Here the green represents the cards in hand and the pink, the cards being picked up, sequentially from left to right.

**Classroom Activity:** Take number cards from 1 to 10 and mix them up. Now try each sorting method. Which one takes fewer steps? Which one feels easier?

Try all the three given sorting algorithms with five balls of different sizes which have to be arranged in order of size from biggest to smallest:

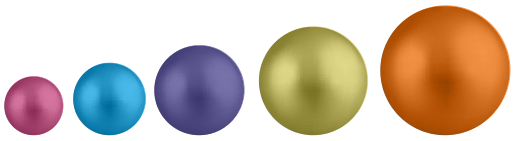

Figure 4: Balls that are in a sorted list from smallest to biggest

Which algorithm is the easiest/fastest?

## 2. Searching algorithms: Finding what you need

Imagine you are looking for your mathematics notebook in your school bag. You go through each item until you find it. That's a Linear Search! Linear Search is when many types of objects are carefully scoured through to find a specific object.

**Types of searching algorithms:**

- **Linear Search:** Check each item one by one. It's slow, but it works for unsorted lists.
- **Binary Search:** This only works for sorted lists. Suppose you are looking for a particular number in a list of numbers, sorted from smallest to biggest. If your number is smaller, search the left half; if it's bigger, search the right half. Repeat until you find it. Note: this algorithm works for more than just numbers! Suppose you want to look up the meaning of the word "Parrot" in your English dictionary. Since the dictionary is very thick and full of thousands of words arranged in alphabetical order, you know it would take a long time to search from the beginning, page by page. So instead, you decide to use a smarter method. You open the dictionary right in the middle and find the word "Lion". You know that the word "Parrot" comes after

"Lion" in alphabetical order, so you ignore the first half of the dictionary and now look only in the second half. Next, you open the middle of this new section and see the word "Tiger". Since "Parrot" comes before "Tiger", you now ignore all the pages after "Tiger" and look only at the words that come between "Lion" and "Tiger". You continue doing this, each time opening the middle of the remaining section and checking which half to keep, until finally, you find the word "Parrot".

**Try This:** Hide a card numbered 27 in an unsorted pile of cards numbered from 1 to 50. Then place the same card in the right place in a sorted pile of cards. Try finding it using both methods. Describe your search. Which pile was easier to search? Which search was faster?

## 3. Greedy algorithms: Choosing the best step now

A greedy algorithm makes the best choice it can at each step, hoping that all these good choices will add up to the best overall solution.

**Example:** Imagine you are collecting peppermints of different sizes. You want to pick the biggest ones but can only pick three. A greedy strategy would pick the biggest visible ones right away, without checking all the comparisons.

Figure 5: Peppermints

**Activity:** Suppose it's your birthday, and you get to choose toys to buy as a gift. Your box has limited space – with space available for only three toys. Make a list of five possible toys and give them a score out of ten based on how much you enjoy playing with each toy. For example, if you love playing with puzzles, give the puzzle box a score of 9 out of 10, and if you don't really enjoy playing with toy cars, give the car a score of 3 out of 10. You use the greedy algorithm to pick the toys with the highest score so that you can enjoy them the most. You may end up with three puzzles! Now, try making another selection without repeating any toys. You will see a different type of greedy solution (since the previous solution just involves repeating the toy with the highest score to maximize enjoyment and make it as high as possible).

Think about any other situation in your life where you can apply greedy algorithms to solve a problem.

## 4. Recursive algorithms: Solving problems by solving smaller ones

Recursive algorithms solve a big problem by solving a smaller version of the same problem. It's like asking a younger sibling to help, and they ask an even younger sibling, and so on.

**Example:** Russian matryoshka dolls are dolls nested within each other. Think of a problem as a set of matryoshka dolls, with smaller problems nested inside of it.

Figure 6: Matryoshka dolls
(source: Macalester College, Russian studies)

**Activity:** Build a tower of cups as shown below.

Figure 7: A tower of cups

Now, try to think of a way to remove the cups. To remove the cup at the bottom left, you must remove the cups on top of it, and to remove those cups, you must remove the topmost cup. This way, the problem of removing the cups gets reduced to a simpler problem using a recursive algorithm, and when you solve the smallest part of the problem, the rest gets solved sequentially; i.e., when you remove the topmost cup you can proceed to remove the next row of cups, and finally the last row of cups.

Problems that are solved using the recursive algorithm are better suited for higher classes, so we are not giving an example here.

## 5. Brute force: Try every possibility

Brute force means checking every possible solution. It's not clever, but it guarantees a solution.

**Example:** Suppose you have a lock on a door, and a set of keys with 10 keys in total. You have no idea which key unlocks the door, so you must try every single key and put it into the door lock to find which key unlocks the door.

**Activity:** Let students think of a number between 1 and 20 and let a partner guess it using a brute force method.

How many guesses did it take?

## Conclusion: Think like an algorithm!

Algorithms are everywhere – they can help us be more organized, more efficient, and more creative. By learning different types of algorithms, you become a better thinker and problem-solver, because you can apply different methods of approaching a problem to try and solve it with these algorithms.

Next time you face a tricky challenge, ask yourself: Can I break this down into steps? Can I try different methods such as greedy, brute force or recursive algorithms?

Because when you think like an algorithm, no problem is too big!

**Note from Editor:** One may wonder if such algorithms should be introduced in primary classes. We would certainly not recommend naming and defining the algorithms for searching; however, you would have noticed that most of the examples used are from the everyday life of primary school children. It would be interesting to design tasks that require searches- if students can employ these different methods and compare them during reflection and discussion after the task, they would be absorbing the ideas of computational thinking and understanding its relevance in a fun and non-threatening manner.

### Reference

1. Tonapi, A. (2024). Introduction to algorithms. *At Right Angles, (20),* 14–19. https://bit.ly/3XJ6E1e

**ANUSHKA TONAPI** is a 11th grade student at Alpine Public School in Bengaluru. She is a Young Member of the New York Academy of Sciences and she is a Spirit of Ramanujan Fellowship awardee. She enjoys solving math problems and is interested in theoretical computer science. Anushka loves playing chess and practises Carnatic music in her spare time. She can be contacted at Anushka.tonapi@gmail.com