# Quick Sort

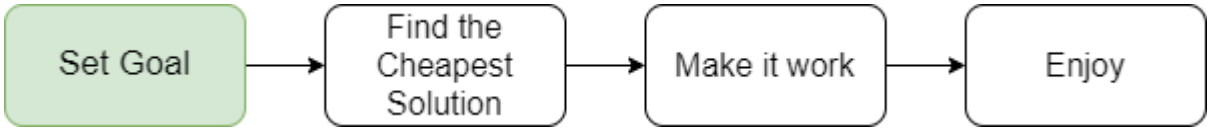
v0.1 by Tree

## 0. Knowing your goal is important

Humans tend to change things according to their will to fulfill their unlimited desires. However, the lazy nature of human beings, always looking for shortcuts and sneaky ways. Surprisingly, these two negative traits have given humans the motivation for technological advancements, making us the greatest creature of the Solar System.

In this tutorial, our goal is sort a bunch of random numbers from small to large.

The goal is set, commander. We accomplished quarter of the mission!




## 1. Find the cheapest solution

As I said, we humans like free stuff. But in this universe, nothing is truly free except chaos. Even thin air, keeping it clean actually costs a lot of money.

So I listed some of the options that I can instantly think of:

1. Just give up and play Minecraft.

    +Sleep till afternoon!

    -You are not going to achieve the goal.

    -You're gonna get fired/failed.

2. Hire a person to do it for you.

    +Brain dead solution.

    -Even minimum wage is too expensive for this job.

    -Costs will increase exponentially as the quantity of number gets larger.

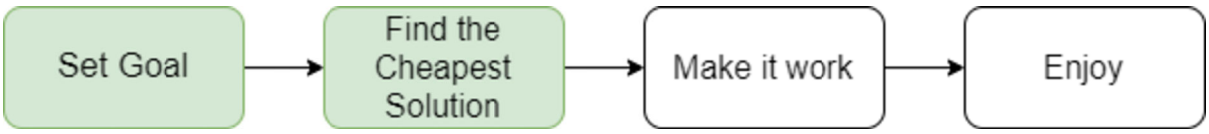
3. Make a machine that sort numbers.

+Minimum wage is 0, probably has a little bit of electric bill.

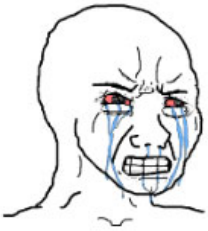
+Easy to scale up.

-Smart enough to make the machine.

Obviously, third option is the cheapest. Let’s choose third option shall we.




Good news! We already have a magical machine called a computer that claims to be able to solve any problem! Now we just tell computer what it need to do… right? However, even we sent our best negotiators, we still failed to communicate. Computer doesn't seem to understand human language.



Sort dis group of numbers plz!!!

What the hell is this dude talking about?

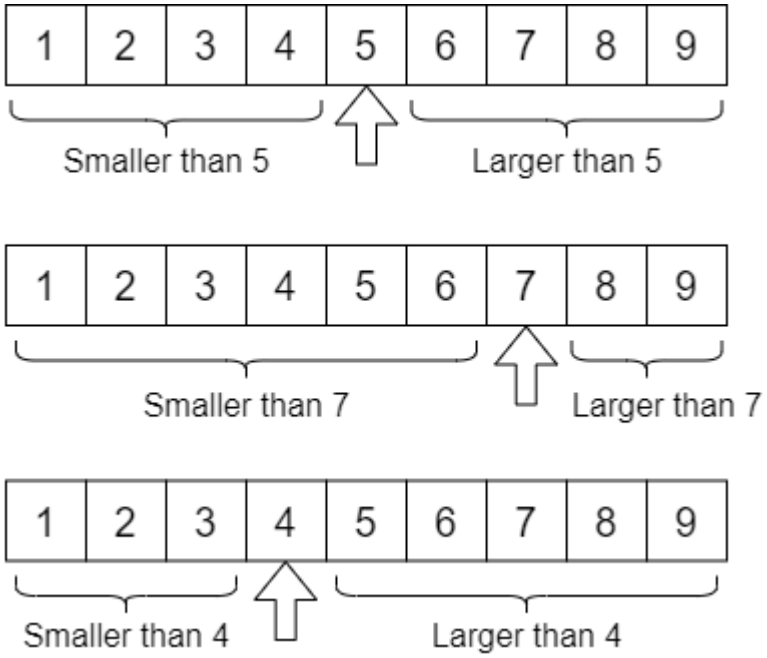
That's because computer can only understand and process some of the simplest commands, like move a number, count, add numbers, etc.

Instead directly input some daily English sentences, we should tell it exactly how to do at each step, similar to teaching urban youth how to cook. In order to write commands that will let computer output the results we want, a plan or idea is need to be set first, we call it **Algorithm**, and also we refer those commands as **code**. Algorithm is the idea behind the code.

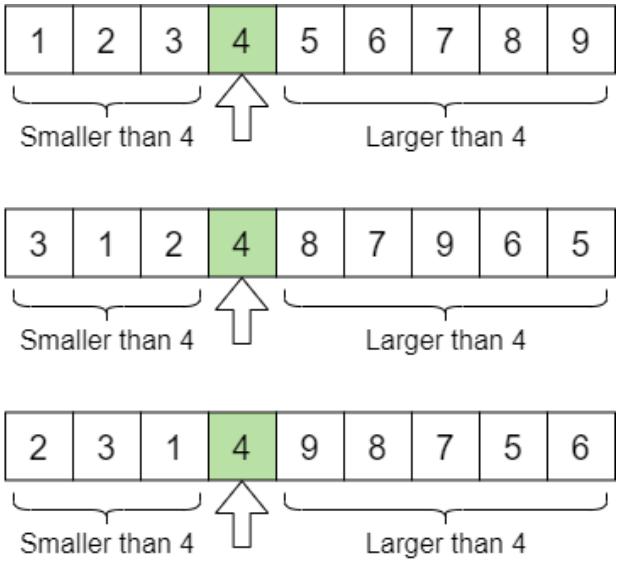
In this tutorial, we are going to learn quick sort, which is one of the most difficult algorithm for beginners.

## 2. Lets Start -- Quick Sort

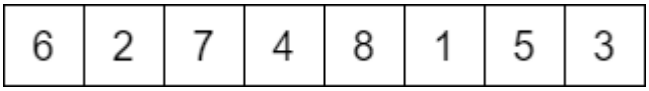
If you look at an array that has been sorted from smallest to largest, you will notice a phenomenon: for any number in this array, all the numbers to the left are smaller than this number, and all the numbers to the right are larger than this number.




Another words, if a number **is larger than** all the numbers to its left side, and **smaller than** all the numbers to its right side, then this number is in the final correct sorted position (You can add “or equal to” behind the **bold words** for more general cases). We can use this idea to design an algorithm that sorts numbers.

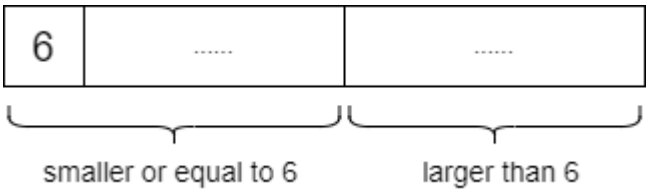


Here is an example with random numbers.




For the convenience, we choose first number of the array as pivot.

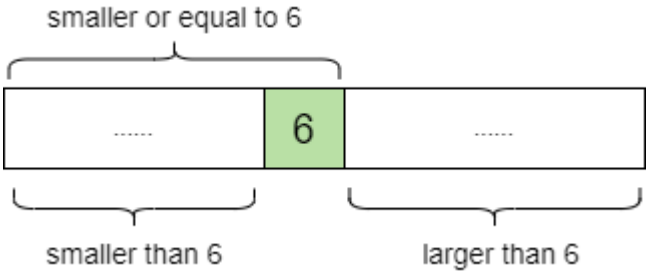
**Divide** those numbers to two parts, numbers that **smaller or equal to 6** (first part) and **larger than 6** (second part).




The largest number of the first part is 6 (otherwise you made a mistake in the previous step). So rightmost position of this part is the final position of the 6.

**Swap** the rightmost number of the first part **and** 6.

Then, 6 is in the final sorted position. 6 is larger than all the numbers to its left side, and smaller than all the numbers to its right side.

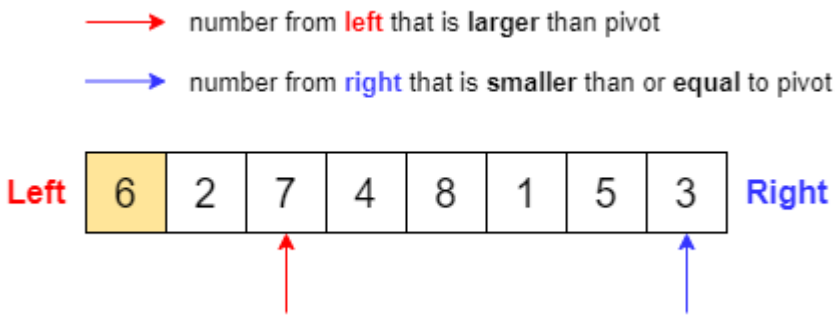



Next, treat these two small wings (smaller than 6 & larger than 6) as independent arrays, do the exact same process until all the numbers are moved to their sorted position.

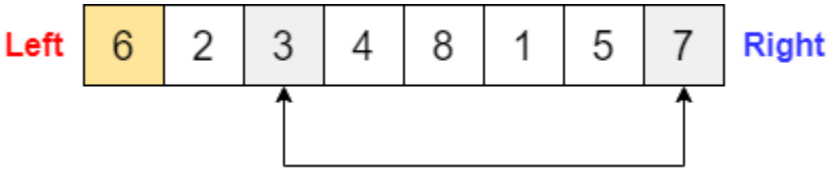
You may ask how to categorize these numbers with their size and divide them into two groups, computer would ask the same thing. It is still too complicated to the machine.

Luckily someone found the solution many years ago.

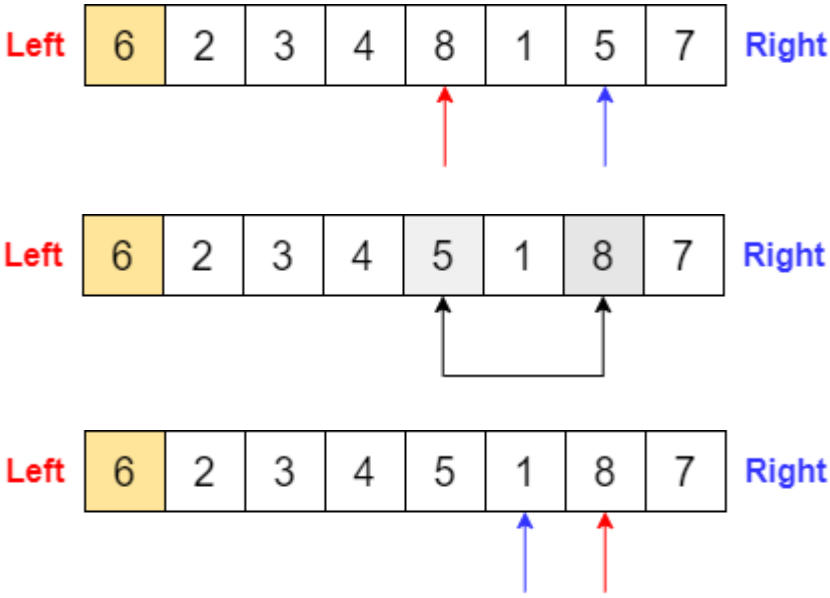
**Step 1**, find the first number from left that is larger than pivot 6, and the first number from right that is smaller than or equal to pivot 6.



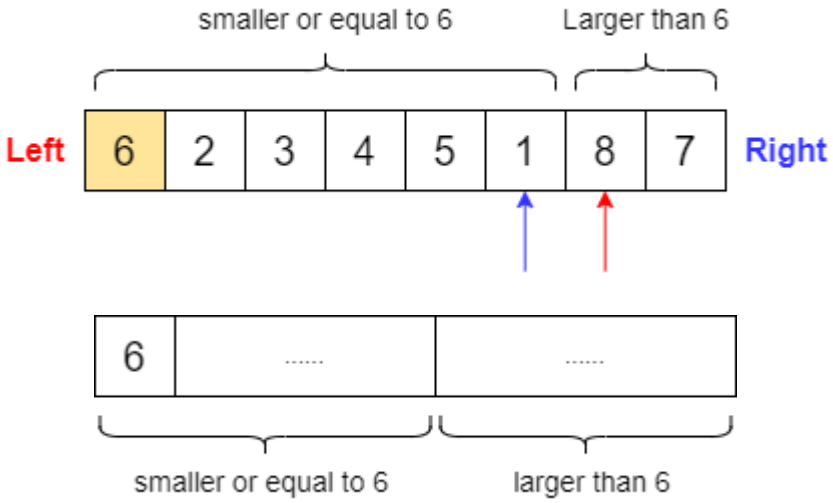

**Step 2**, **Swap** these two number



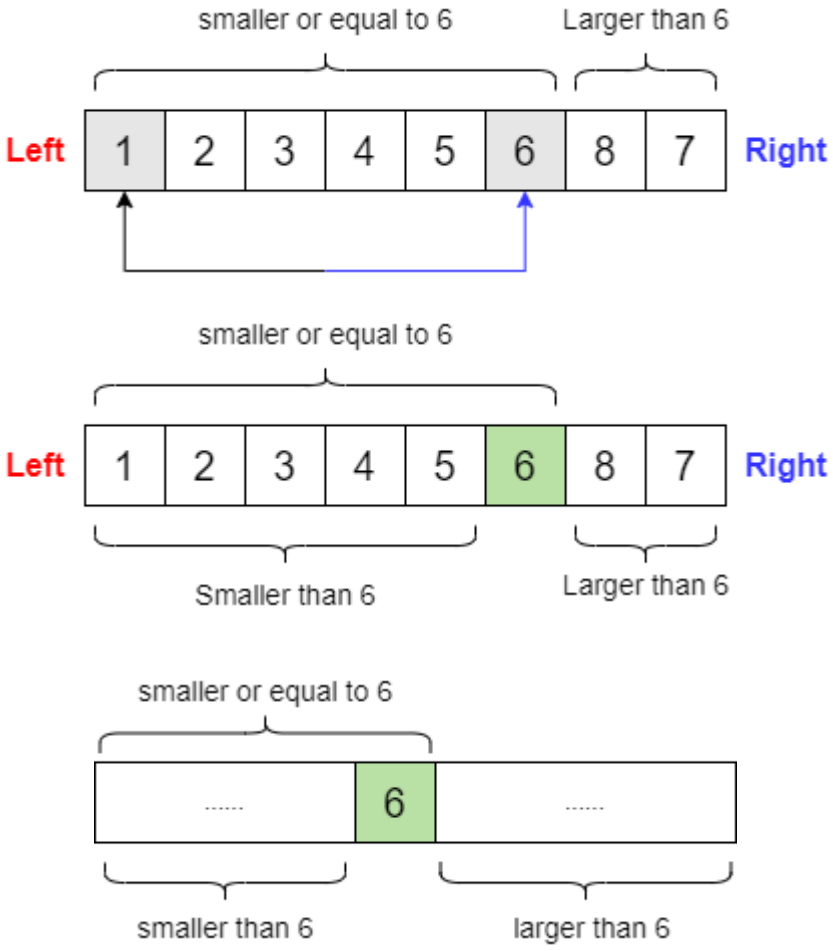

**Step 3**, repeat previous two steps **until** red arrow went right side of the blue arrow.



Don’t swap! Stop right here. Have you noticed something?

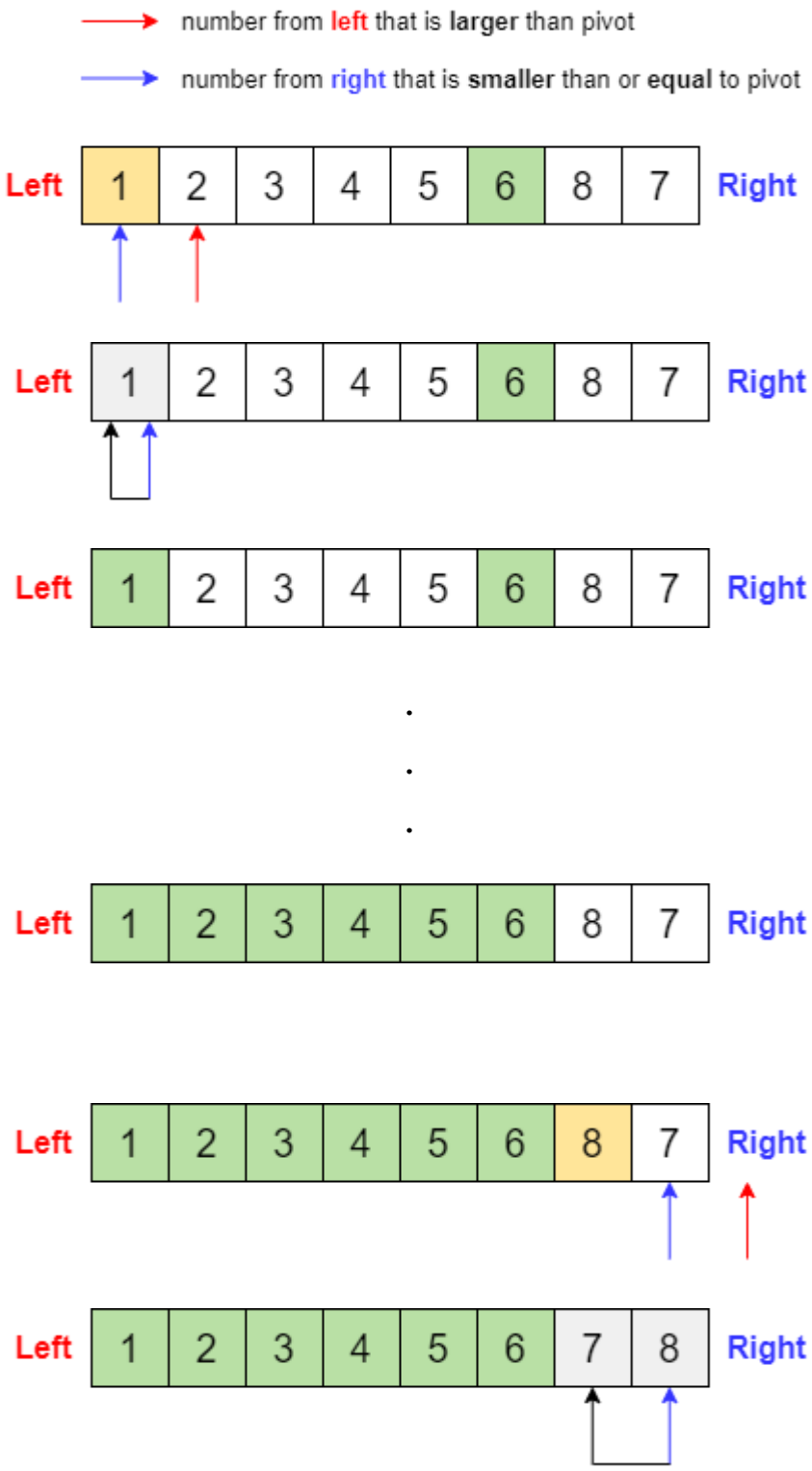


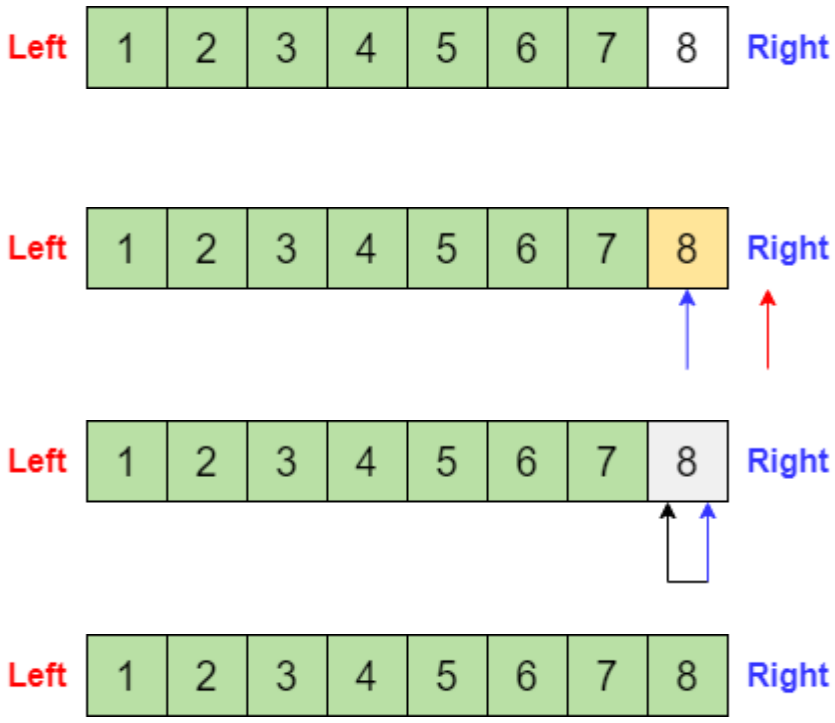

**Step 4**, you guessed it, **Swap** the rightmost number of the first part and 6. That is, S**wap** the 6 with the number indicated by the blue arrow.



Hooray!! We did it! Number 6 is now officially in final sorted position.

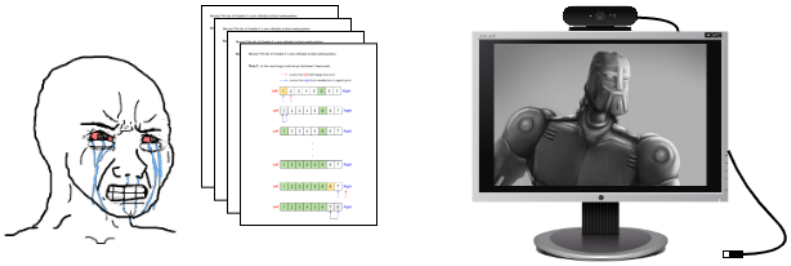
**Step 5**, do the same thing to mini arrays that haven't been sorted.






Nice! That is all about the quick sort algorithm.

But wait! How about the code?




Before using legitimate programming language to write this sorting function, we can use pseudocode. Pseudocode has no strict syntax formatting requirements, as long as it is easy for you to understand, and at the same time, easy to translate into computer code. Literally if you want, you can use Middle English and write it like poetry from the Canterbury Tales (but for God's sake, don't).

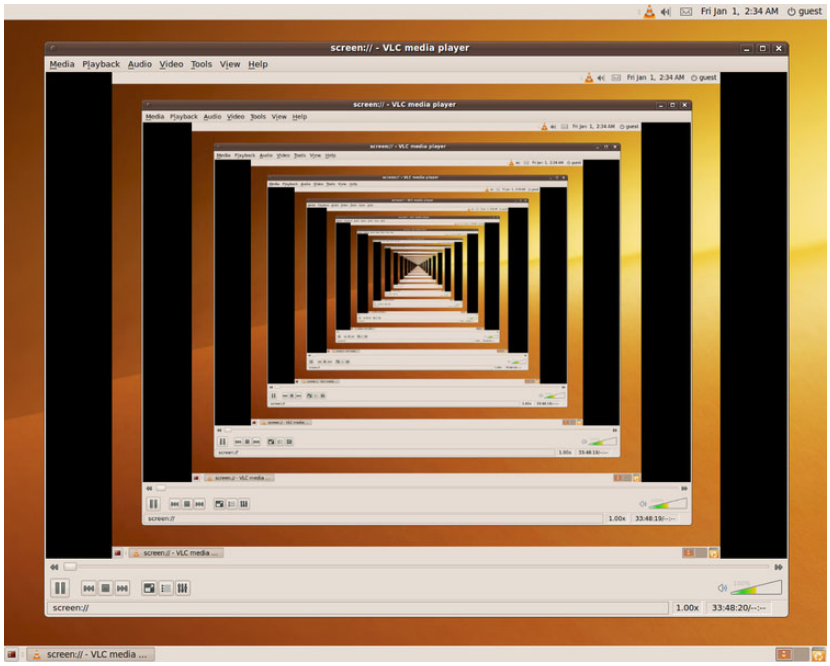
Pseudocode

```
Function partition(array[ ], leftMostIndex, rightMostIndex)
{
        set pivot = leftMostIndex;
        set redArrow = leftMostIndex + 1;
        set blueArrow =  rightMostIndex;

        while loop(redArrow <= blueArrow)
        {
                while loop(array[redArrow] <= array[pivot] AND redArrow <= blueArrow)
                {
                        add 1 to redArrow;
                }

                while loop(array[blueArrow] > array[pivot] AND redArrow <= blueArrow)
                {
                        subtract 1 from blueArrow;
                }

                if(redArrow <= blueArrow)
                {
                        SWAP array[redArrow] and array[blueArrow];
                }
        }

        SWAP array[pivot] and array[blueArrow];
        return blueArrow;
}


Function quicksort(array[ ], leftMostIndex, rightMostIndex)
{
        if (leftMostIndex < rightMostIndex)
        {
                set greenBlockIndex = partition(array, leftMostIndex, rightMostIndex);
                quicksort(array, leftMostIndex, greenBlockIndex – 1);
                quicksort(array, greenBlockIndex + 1, rightMostIndex);
        }
}
```

If you feel dizzy while trying to understand and simulate the code in your head, STOP before damaging your brain cells. You might confuse why the function quicksort is defining itself. Is that even legal?

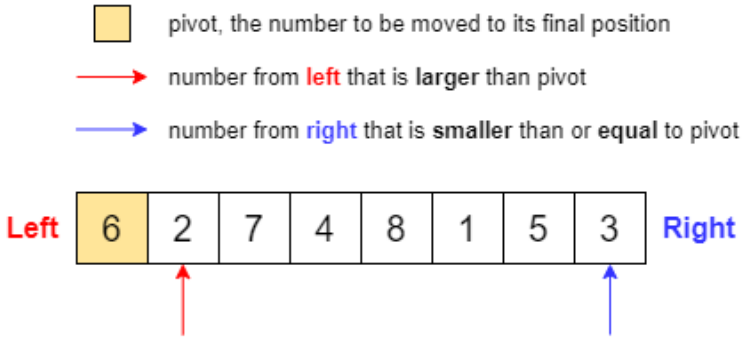
Introducing Recursion. Tales of its misdeeds are told from Ireland to Cathay.  Many programmers hate recursion due to its complexity and unreliability, but somehow loved by a group of bald headed mathematicians. Nonetheless, it's still a great tool to use especially in this case.

In the real world, recursion occurs when you place two mirrors in parallel, or point the camera at the screen during a zoom meeting.




In this Pseudocode, the purpose of function partition is to execute **step 1** to **step 4** and return the location of sorted pivot, which is the index number of the green block shows in the picture under **step 4**.

First, initialize the location of two arrows and the pivot.

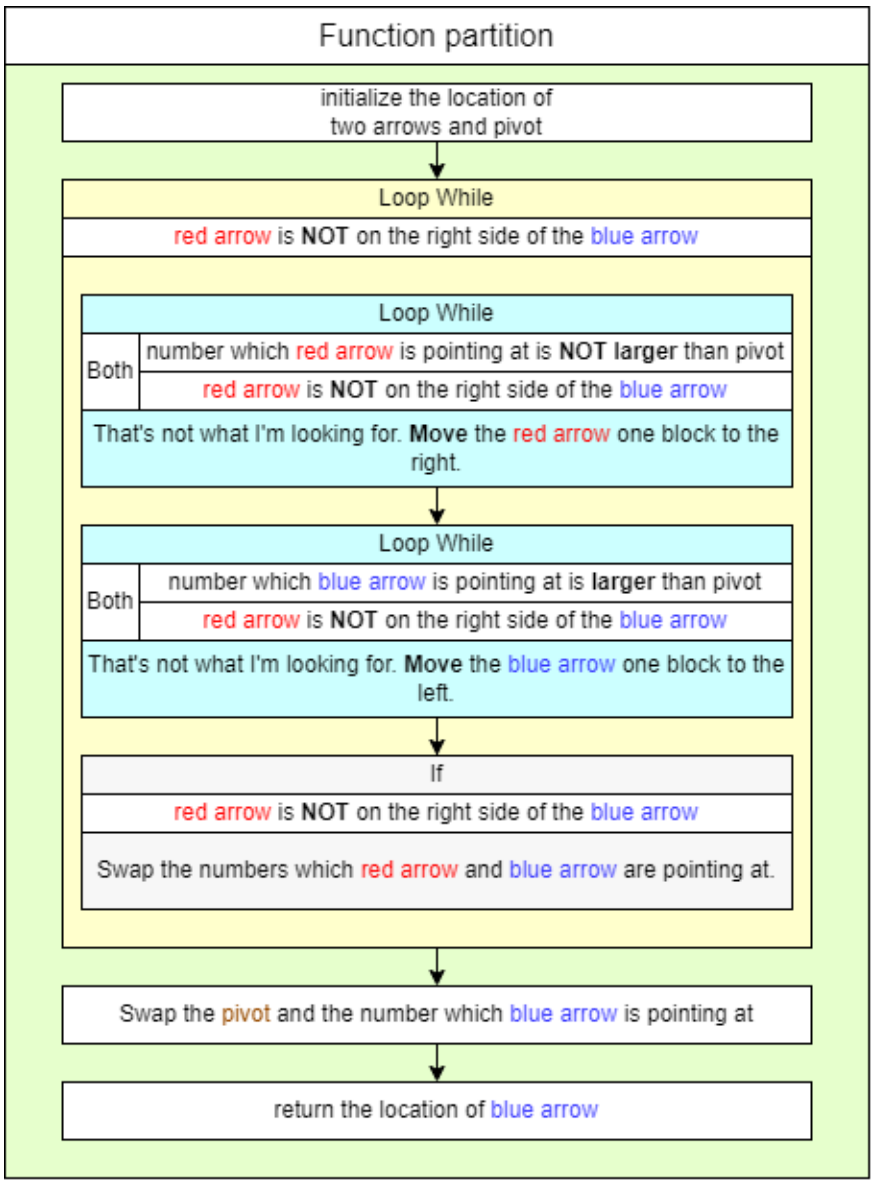


There are 3 While loops. Two inner while loops are inside of one outer while loop.

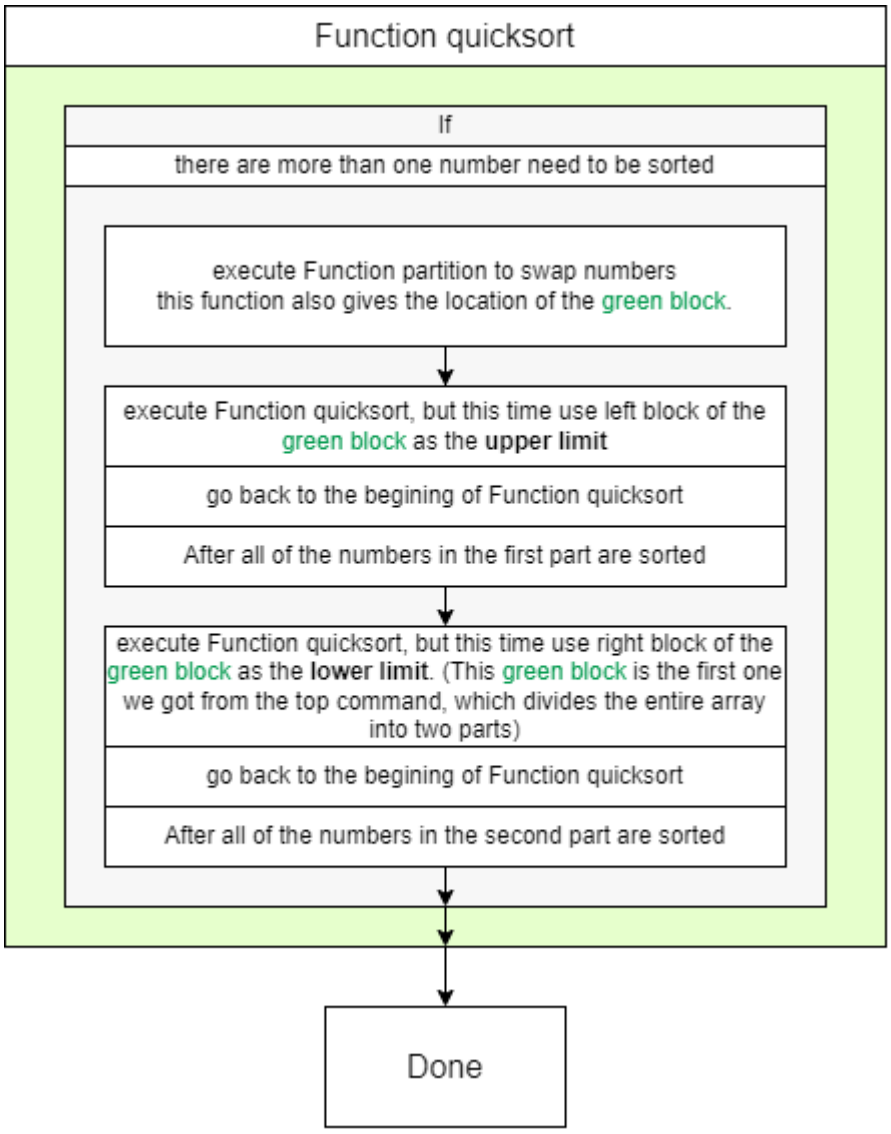
The purpuse of outer while loop is to execute the **step 1** and **step 2**.

The purpose of the first inner while loop is for the red arrow to find number larger than the pivot, and the purpose of the second inner while loop is for the blue arrow to find number smaller than or equal to the pivot. After both missons are accomplished, if red arrow is still NOT on the right side of the blue arrow, swap the numbers which red arrow and blue arrow are pointing at.

If red arrow is on the right side of the blue arrow in any moment, exit the outer loop. Swap the pivot and the number which blue arrow is pointing at. Then return the location of blue arrow. That's because we need to know which position divides the array into two parts.



Please read this diagram with previous pages side by side. You probably need some time to understand the pseudocode.



## 3. Code

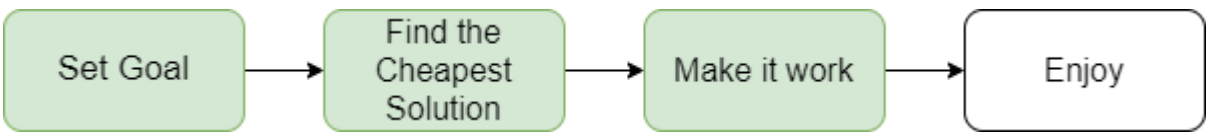
Finally, we simply translate the pseudocode into the real programming language, then feed it into the computer and pray for no errors or warnings pop up.

I've implemented this algorithm in C.

```c
# include < stdio.h >

void swap(int *p1, int *p2)
{
  int temp;
  temp = *p1;
  *p1 = *p2;
  *p2 = temp;
}


int partition(int array[], int leftMostIndex, int rightMostIndex)
{
  int pivot = array[leftMostIndex];
  int redArrow = leftMostIndex + 1;
  int blueArrow = rightMostIndex;

  while(redArrow <= blueArrow)
  {
    while((array[redArrow] <= pivot) && (redArrow <= blueArrow))
    {
      redArrow++;
    }
    while((array[blueArrow] > pivot) && (redArrow <= blueArrow))
    {
      blueArrow--;
    }
    if(redArrow <= blueArrow)
    {
      swap(&array[redArrow], &array[blueArrow]);
    }
  }

  swap(&array[leftMostIndex], &array[blueArrow]);
  return blueArrow;
}
```

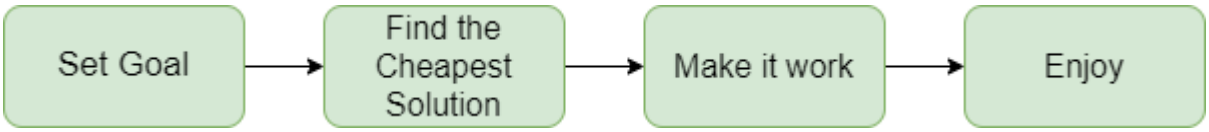
```c
void quickSort(int array[], int leftMostIndex, int rightMostIndex)
{
  if(leftMostIndex < rightMostIndex)
  {
    int greenBlockIndex = partition(array, leftMostIndex, rightMostIndex);
    quickSort(array, leftMostIndex, greenBlockIndex - 1);
    quickSort(array, greenBlockIndex + 1, rightMostIndex);
  }
}


void printArray(int array[], int size)
{
  for(int i = 0; i < size; ++i)
  {
    printf("%d, ", array[i]);
  }
  printf("\n");
}


int main(void)
{
  int array[] = {6, 6, 7, 10, 8, 7, 5, 3};
  int size = sizeof(array)/sizeof(int);

  printf("Original Array:\n");
  printArray(array, size);

  quickSort(array, 0, size - 1);

  printf("Sorted Array:\n");
  printArray(array, size);
}
```

No problem so far.



## 4. Enjoy

It is time for me to enjoy a good sleep. Good night.




---

This tutorial is still far from perfect. It may contain grammatical errors and typos, and even some incomprehensible sentences. If you find such cases, or some other problems, please report them on this GitHub page.

https://github.com/Tree512/Algorithm-Tutorials

Reference:

https://youtu.be/Hoixgm4-P4M

Highly recommend to watch this video and his other algorithm tutorials.