**CS469 Data Structures and Algorithms**

**HOS04A: Quick sort and Merge Sort**

04/19/2023 Review by Christopher Sharp

01/22/2024 Updated by Anh Nguyen

09/29/2024 Reviewed by Shahid Khan

School of Technology and Computing (STC) @City University of Seattle (CityU)

**Before You Start**

* The document’s examples are written in Python. Please finish the Python tutorial in Module0 folder before you start the assignment.
* Some steps are not explained in the tutorial**.** If you are not sure what to do:
  1. Consult the resources listed below.
  2. If you cannot solve the problem after a few tries, ask a TA for help.

**Learning Outcomes**

Students will be able to:

* Understand Quicksort and Merge-Sort
* Implement Quicksort and Merge-Sort by using Python

**Resources**

* Bhargava, Y, A. (2016). Grokking algorithms GitHub repository. Retrieved from: <https://github.com/egonSchiele/grokking_algorithms>
* Data Structures and Algorithms in Python:

<https://ebookcentral.proquest.com/lib/cityuseattle/reader.action?docID=4946360>

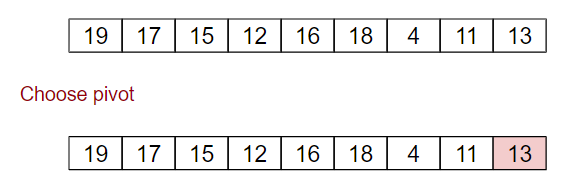
* Grokking algorithms : an illustrated guide for programmers and other curious people: <https://learning.oreilly.com/library/view/grokking-algorithms/9781617292231/OEBPS/Text/kindle_split_001.html>

# 1. Quicksort

The name "Quick Sort" comes from the fact that quick sort can sort a list of data elements significantly **faster** than any of the common sorting algorithms. It is one of the most efficient sorting algorithms and is based on the Divide and Conquer approach.

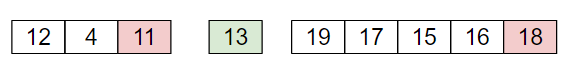
The basic idea of quicksort is the splitting of an array (partition) into smaller ones and swapping (exchange) based on the comparison with the 'pivot' element selected.

**Step 1**: Pick an element from the sequence and call it "pivot".



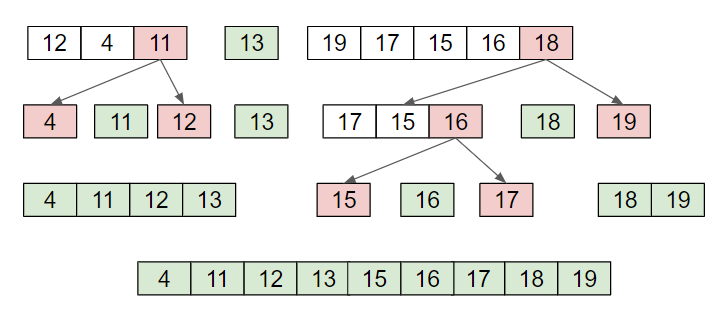
**Step 2**: Re-order the sequence.

* All elements smaller than the pivot’s value are placed on the left side the pivot,
* All elements larger than the pivot’s value are placed on the right side of the pivot
* The same number can go to either side.
* After the partition exits, the pivot is in the middle of the sequence. This is called a partition operation.



**Step 3**: Recursively sort the sub-sequences of

* Elements smaller than the pivot’s value
* Elements larger than the pivot’s value.



## Time complexity

To sort an array of n distinct elements:

* On average, quicksort takes O() time in expectation.
* In the best-case scenario, when each time we perform a partition, we divide the list into two nearly equal pieces, quicksort takes O() time.
* In the worst-case scenario, when one of the sublists returned by partitioning is of size , quicksort takes O().

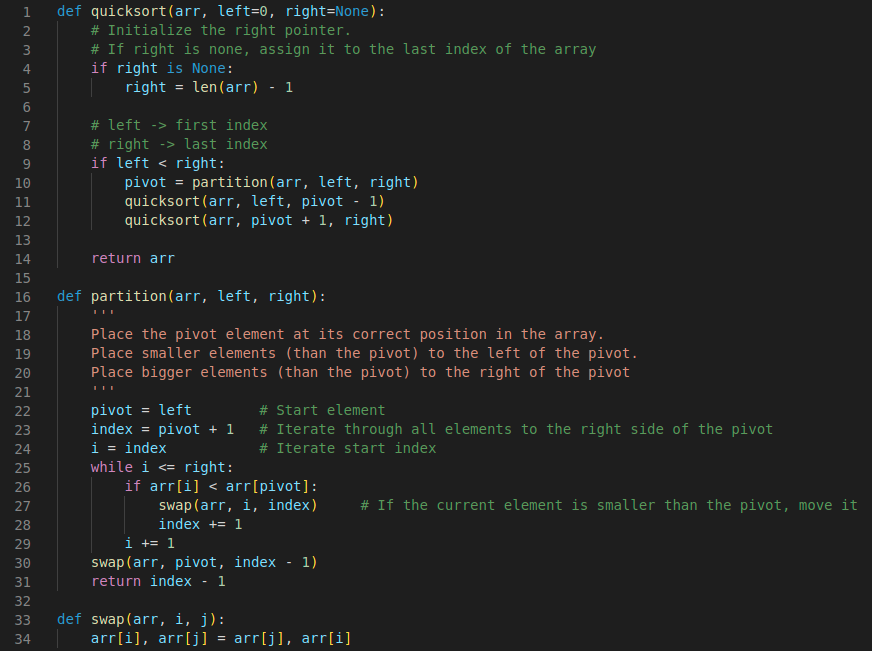
## Choice of pivot

There are multiple ways to choose a pivot:

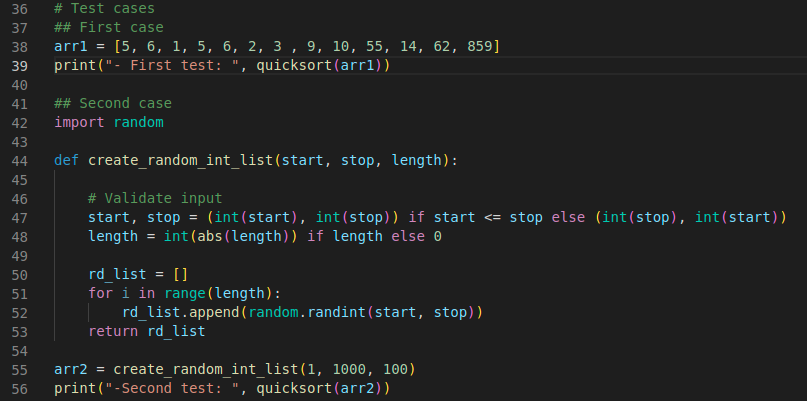
* Always pick the last element as a pivot. (illustrated above)
* Always pick the first element as a pivot. (implemented below)
* Pick a random element as a pivot.
* Pick the middle as the pivot.

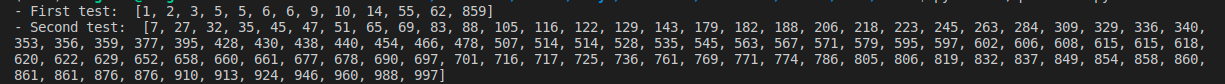
## Implement Quicksort in python

In the cloned repository folder, create a file called **quicksort.py**



In the same file **quicksort.py**, add the following code to create test cases:



Run the code to check whether the arrays have been sorted. The output should look like this:*Note*: The result in the second test case will be different from the picture since we randomly create the list.

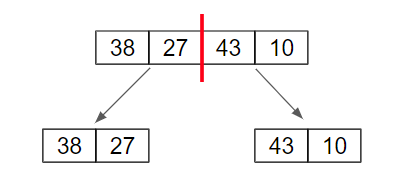
# 2. Merge Sort

Merge sort is also one of the most efficient sorting algorithms.

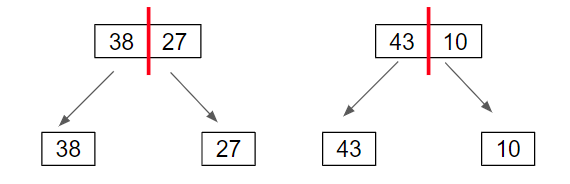
Like quicksort, merge sort also works on the principle of Divide and Conquer.

The idea of merge sort is breaking down the array into several smaller arrays, until each of these arrays contain a single element, then merging these sub-arrays into a sorted array.

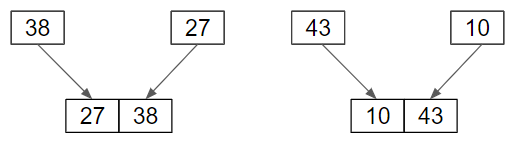
Step 1: Initially, divide the array into two equal halves.



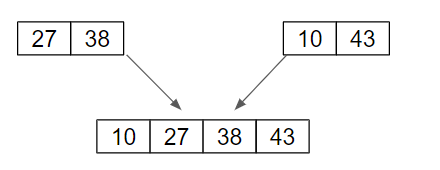
Step 2: Repeat Step 1 to sub-arrays, until they reach a length of 1 – unit length (can no longer be split).



Step 3: Merge the unit length arrays to create sorted subarrays.



Step 4: Repeat Step 3 until all subarrays have been merged, and the original array length is reached.



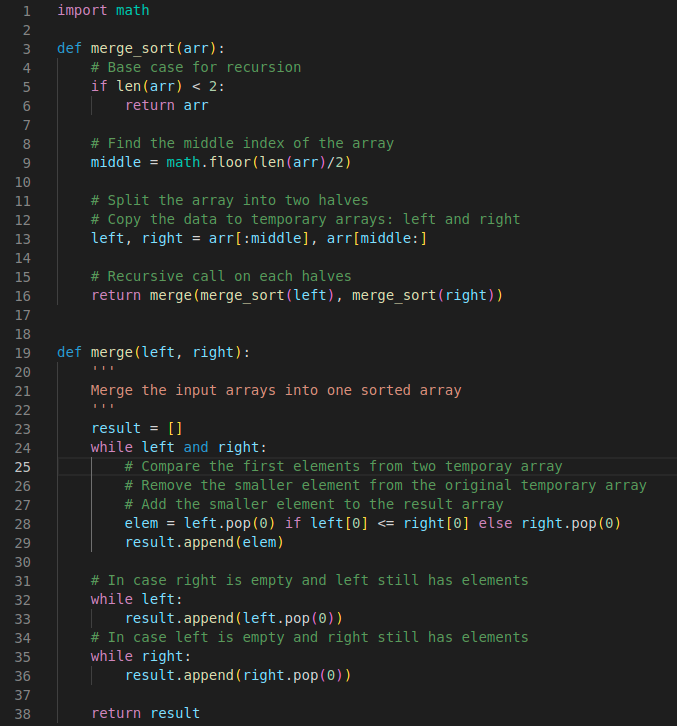
## Time complexity

To sort an array of n distinct elements:

* On average and in the worst-case scenario, merge sort takes O() time in expectation.

## Implement Quicksort in python

In the cloned repository, create a file called **mergeSort.py**



In the same file **mergeSort.py**, add the following code:

文本

描述已自动生成

You can use the test case created in Quicksort to test the Merge Sort algorithm.

Run the code to check whether the output results have been sorted.

To visualize the searching processes on the two test cases, copy the entire code into the python tutor website:

<https://pythontutor.com/visualize.html>

**Q:** **What is the core difference between quick sort and merge sort? Explain those processes and save your explanation in a new file with the name:**

**Submission*-YourName.pdf***

**Optional Resource**:

If you want to learn more about the Quick Sort or Merge Sort and solve some code challenges. You can try some Leetcode Coding challenges: <https://leetcode.com/problems/sort-an-array/>

Because this part is optional, so the TA won’t be responsible for answering questions for the challenges on Leetcode. Instead, you can visit the question’s discussion board to find hints and solutions.

**Push Your Work to Github**

Open terminal and make sure you’re in the repository folder. (i.e: hos04\_courseName\_GitHubUserName)

**Add all code .py files and your answer document.**

**Type the following command to upload your work**:

|  |
| --- |
| >>>> git add .  >>>> git commit -m “Submission for HOS04 - *YourName*”  >>>> git push origin master |