Eric Ramsey

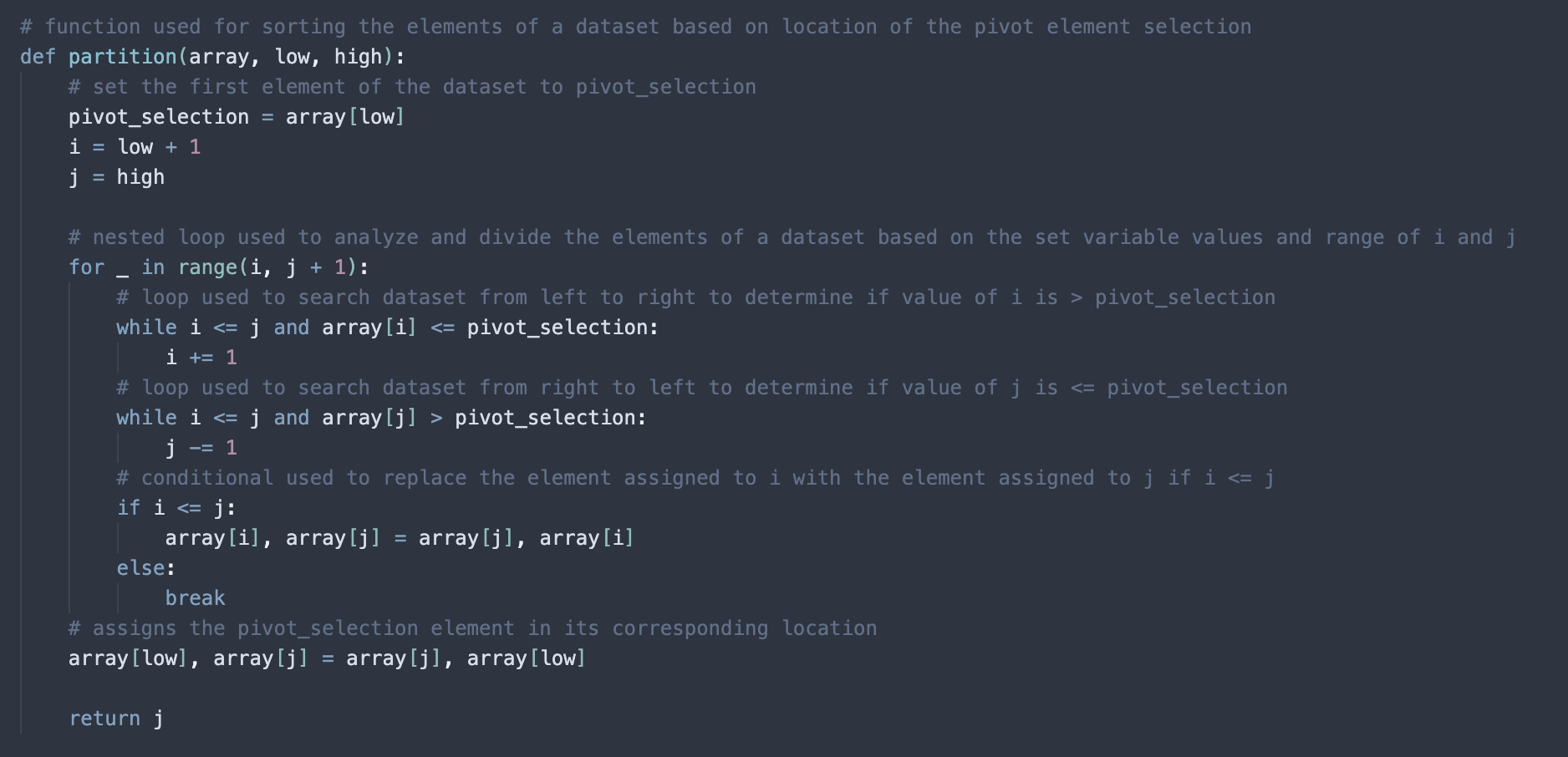
CSCI 3102

Module 3 Assignment

Spring 2024

For this assignment I chose to implement the three Quick Sort techniques using the Python programming language and tested the sorting efficiency by creating a unit test file to verify the accuracy of the created functions. The file ***quicksort.py*** is created to store the functions for the Quick Sort technique implementations of ***quicksort\_pivot\_selection***, ***quicksort\_random\_selection***, and ***quicksort\_median\_of\_three***. This file also contains two partition functions, ***partition*** and ***partition\_median***, used for the sorting process and separating the elements of a given dataset based on the location of the element assigned as the pivot selection. I chose to have two separate partition functions for this file due to errors with determining the median of a given dataset and partitioning the elements after. After researching how to resolve the errors I was encountering, located in the resources below, separating the partitioning process into two separate functions was the best approach to the errors. The file ***test\_quicksort.py*** is created to store unit test cases and functions to determine the efficiency and accuracy of the previously mentioned functions.

**Partition Functions**



The first partition function accepts three arguments (***array, low, high***) and is used to arrange the elements of a given dataset in respect to the location of the element assigned to the variable ***pivot\_selection***. The element that is identified as the first index of the dataset is assigned to ***low*** and the element that is identified as the last element of the given dataset is assigned to ***high***. The next step of the of the functions process is to assign the values of ***low*** and ***high*** to the variables ***i*** and ***j***, with the addition of incrementing ***i*** by one to account for shifting to the second element in the given dataset. The next step of this functions process is to iterate through the dataset (from left to right) with an implemented ***for loop***, that contains nested ***while loops*** and conditional statements used for analysis of the temporary variables, ***i*** and ***j***, and set the final position of the ***pivot\_selection*** prior to beginning the sorting process. The Quick Sort technique functions that use this partition function are ***quicksort\_pivot\_selection*** and ***quicksort\_random\_selection***.

A screenshot of a computer program

Description automatically generated

The second partition function, contained in the screenshot above, was implemented to resolve issues occurring with the function used to implement the Median of Three technique. This function, ***partition\_median***, is used to partition the elements in the function ***quicksort\_median\_of\_three***. This partition function has the same requirements and process as the previously mentioned partition function, except for the partition selection is assigned to the median of the three elements, which are the first, middle, and last elements of the given dataset. This is indicated in the first lines of the ***partition\_median*** function. The function begins its process by determining the median of the three required elements. The next step in this function’s process is to determine the corresponding location of the assigned array elements ***low***, ***median***, ***high***, and adjust the elements in ascending order (from left to right). After completing these steps, the function then is required to implement the partitioning steps that are identical to the previously mentioned partition function which finalized the ending position of the ***median*** variable used as the pivot selection of this technique. The main difference of these two partition functions is the pivot selection element, the first function simply selects the first element of the dataset while the ***partition\_median*** selects the median of three. This resolved the issues I was running into while enhancing the performance of this function and the Quick Sort technique.

**Quick Sort Functions**

A screen shot of a computer program

Description automatically generated

Similar to the previously mentioned partition functions, the three Quick Sort technique functions all have the arguments or parameters ***array***, ***low***, and ***high***. The function ***quicksort\_pivot\_selection*** is used to implement the original Quick Sort technique and uses recursive calls to sort the elements of a given dataset after first selection the element that is identified at the first index position of the dataset. Prior to initializing the ***pivot\_selection*** variable, the function is required to determine if the value of ***low*** is less than the ***high*** which is used to indicate that there are at least two elements in the dataset before continuing. Once the conditional statement requirement has been met and the pivot selection element has been initialized, the function then recursively calls itself with the implementation of the ***pivot\_selection – 1*** argument to analyze elements left of the pivot selection. The second recursive call has the parameter implementation ***pivot\_selection + 1*** to analyze elements right of the pivot selection.

The second Quick Sort Technique function implemented in this file is the ***quicksort\_random\_selection*** function. This function implements the Quick Sort algorithm but instead of selecting the first element as the pivot selection, it will select a random element in the given dataset and assign it as the pivot selection. The random pivot selection is then assigned to the first element of the dataset, swapping positions with the former first element of the dataset. After calling the partition function to complete the process, the function then completes two recursive calls, and just as the ***quicksort\_pivot\_selection***, one recursive call is assigned to elements to the left of the pivot selection and one recursive call is assigned to the elements to the right of the pivot selection.

The final Quick Sort technique used in this file is the ***quicksort\_median\_of\_three*** functions. This function is used to implement the Median of Three variation of the Quick Sort algorithm and unlike the previously mentioned functions, uses the median of the ***low***, ***middle***, and ***high*** dataset elements as the pivot selection for the sorting process. This function, like the other two Quick Sort functions, starts its process with a conditional statement used to verify that there are at least two elements in the given dataset. Once the requirement of the conditional statement has been met, the functions next step is to call the ***partition\_median*** function used to partition the elements of the dataset in their corresponding positions based on the newly found median pivot selection element. Similar to the previously mentioned Quick Sort functions, this function will also use two recursive calls with one assigned to the elements left of the pivot selection and one assigned to the elements to the right of the pivot selection.

**Unit Test Functions**

A computer screen shot of code

Description automatically generated

The unit test function located in the ***test\_quicksort.py*** file are each designated to their corresponding functions in the ***quicksort.py*** file. To test each of these Quick Sort technique functions, I chose the unsorted array I originally used in the previous module assignments. To test the running time of each of these function’s sorting processes, I imported the ***time*** class contained in the Python frameworks, initiated a start time variable, an end time variable, and assigned the difference of these two values to determine the total running time. The final output of these tests can be found in the screenshot below, all of which passes however I noticed that the ***quicksort\_pivot\_selection*** was significantly slower than the other two. This gave me the assumption that the random pivot selection and Median of Three techniques clearly enhance the efficiency of the Quick Sort algorithm. This also indicated to me that using the first element of the array is clearly a slower process, at least for this dataset. I’d like to continue testing these algorithms with larger dataset values and possibly alter my algorithms to see how I could increase the efficiency of each function.

A computer screen shot of numbers

Description automatically generated

**Resources:**

* <https://www.javatpoint.com/quicksort-in-python#:~:text=The%20Quicksort%20algorithm%20works%20by,equivalent%20to%20the%20pivot%20element>.
* <https://hackernoon.com/essential-algorithms-the-quick-sort-mr1q32wr>
* <https://medium.com/geekculture/quicksort-in-python-abf3881b3cca>
* <https://www.youtube.com/watch?v=CB_NCoxzQnk>
* <https://www.youtube.com/watch?v=Vtckgz38QHs>