**Function API Documentation**

The following is documentation for the functions contained within the main and algorithms files regarding their usage and inputs.

|  |  |
| --- | --- |
| time\_sort(file, algorithm, iterations)  : file > string  : algorithm > callable[[list], None]  : iterations > int | Tests a given sorting algorithm with the contents of a specified file loaded into an array. Can execute multiple iterations to average the runtime of the algorithm for more accurate results. |
| quick\_sort(arr)  : arr > list | Takes in a list and sorts it using the quicksort algorithm. |
| quick(arr, low, high)  : arr > list  : low > int  : high > int | Call used to execute quicksort on a portion of a list, low being the lowest value to begin sortin and high to be the highest. |
| partition(arr, low, high)  : arr > list  : low > int  : high > int | Partitions the given list between a low and high value. Used in the quick sort algorithm to organize values. |
| heap\_sort(arr)  : arr > list | Sorts the given list using the heap sort algorithm. |
| generate\_heap(arr)  : arr > list  : n > int  : i > int | Generate’s heaps given the index of the list where to generate and the number of values to include in the heap. |

**Analysis**

Time Complexity

Heap Sort:

* Average: O(n log(n))
* Worst: O(n log(n))

Quick Sort:

* Average: O(n log(n))
* Worst : O(n^2)

Performance Analysis

|  |  |  |
| --- | --- | --- |
| # of elements | Heap Sort Performance (100 runs average) | Quick Sort Performance (100 runs average) |
| 100 | 0.45ms | 0.14ms |
| 1,000 | 6.02ms | 2.49ms |
| 10,000 | 79.39ms | 33.69ms |
| 100,000 | 1,1456.2ms | 421.51ms |

Questions

**Does your calculated run-time complexity match the measured run-time complexity? Why or why not?**

No, though quicksort does have a worse worst case than heapsort, the quick sort algorithm itself tends to perform better on average in practice than heap sort. Though the Big O notation is still correct.

**Under which dataset sizes does each algorithm perform best? Why or why not? Does any algorithm perform best under all cases?**

Both algorithms perform well at all dataset sizes. They both have an average time complexity which handles large sets well and are both relatively performant algorithms.