Quick Sort: LEXISORT - Easy Sorting

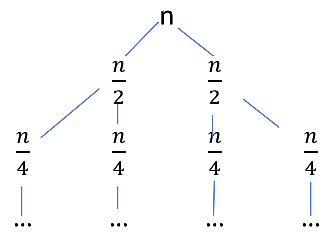
How it works:

For the Lexicographical challenge I used a quick sort algorithm to implement. The quick sort algorithm is split into two different functions, partition and quick_sort. The quick_sort function is called first and takes in the original array, the index of the first element (start), and the index of the last element (end). The first thing quick_sort does is check to see of the start variable is less than the end variable. This check prevents the recursive calls later from running on forever, that is if the start pointer passes the end pointer then the recursive loop breaks. The next step is to call the partition function. The partition function first finds the pivot point to search. The partition function then loops over the range of the entire array passed in. During the loop if the element at the iterating index is less than the element at the end of the array passed in, then the function swaps the two elements. That is, if the element at the end is found to be less than any of the searched elements we know that the element at the end is not the highest value. The loop continues and finally breaks. After breaking the loop a final swap is committed, putting another value at the end, and returning the index for pivot. Returning to the quick_sort function, the program then recursively calls itself to sort through the values to the right of the pivot point, and runs quick_sort again for the left side f the pivot point. This function keeps narrowing things down until the array is of size 1, meaning it is already sorted. Then the array attempts to swap values until all values are put into ascending order.

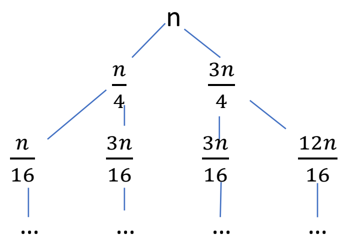
Time Complexity:

The worst-case time complexity of this algorithm is $T(n) = O(n^2)$. The worst-case scenario only occurs when the partition happens to fall on the greatest value, this causes the pivot point to always must switch to the next pointer, looping over the array '$\frac{1}{2}n^2$' times.

The best case for this algorithm is also $T(n) = O(nlog_2(n))$ this occurs when calculating the sum of the algorithm substituting 'n' for 'n+1'. As seen in the partition function, the array will run n times. The next part of the function recursively calls quick_sort $\frac{n}{2}$ times. The quick_sort function is then called repeatedly until n of quick_sort reaches a length of 1. Considering the tree below, each level considers $\log(n)$ time complexity, where n is the sum of the branches.



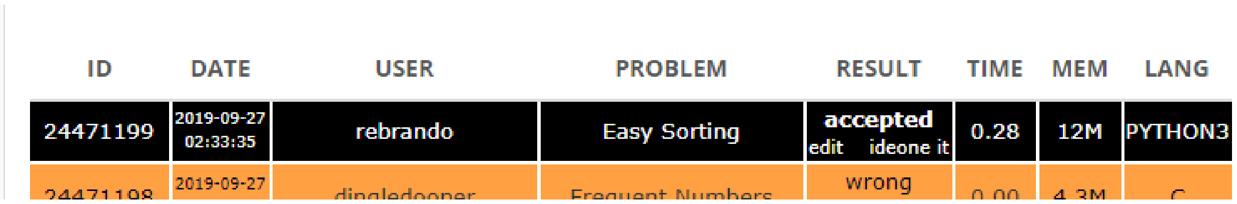
The average case scenario is similar to the best case scenario the only difference is that instead of the array being parted down the middle ($\frac{n}{2}$), or the whole array being parted, the array is parted into a 25%, 75% scenario. So one half of the array has $\frac{3}{4}n$ while the other half has $\frac{1}{4}n$. The using a similar tree to the best case, we see that the time complexity is $T(n) = O(nlog_{\frac{3}{4}}(n))$.



```
def quick_sort(array, start, end):
   if start < end:                                   - O(1)
      part = partition(array, start, end)            - O(1)
      quick_sort(array, part + 1, end)               - O(n/2)
      quick_sort(array, start, part - 1)             - O(n/2)
return array


def partition(array, start, end):                    - Total: T(n)
   pivotPoint = array[end]                           - O(1)
   index = start – 1                                 - O(1)
   loopIndex = start                                 - O(1)
   while loopIndex in range(start, end):             - O(n)
      if array[loopIndex] < pivotPoint:              - O((1)
         index = index + 1                           - O(1)
         array[index], array[loopIndex] = array[loopIndex], array[index]          - O(1)
      loopIndex = loopIndex + 1                      - O(1)
      nextPointer = index + 1                        - O(1)
   array[nextPointer], array[end] = array[end],array[nextPointer]                 - O(1)
   return nextPointer                                - O(1)
```

SPOJ Info:



ID	DATE	USER	PROBLEM	RESULT	TIME	MEM	LANG
24471199	2019-09-27 02:33:35	rebrando	Easy Sorting	**accepted** edit ideone it	0.28	12M	PYTHON3
24471198	2019-09-27	dingledooper	Frequent Numbers	wrong	0.00	4.3M	C