Assignment 2: Sorting with Divide and Conquer

1 Problem Description

This assignment builds on the previous one – we will take the randomized quicksort and mergesort algorithms that you implemented, and modify them slightly. Note that we won't be using deterministic quicksort anymore (you can probably guess why). We will also limit our inputs to the randomized ones, and not use the sorted lists.

Recall that both algorithms use divide-and-conquer, by dividing the list into k=2 parts and dealing with each part individually. In this assignment, we will see the effects of dividing them into $k = 3$ or $k = 4$ parts in addition to the 2 that we explored, and see how that impacts the running time.

For this code, we would like you to write the recurrence relation that describes each algorithm for $k = 2, 3, 4$, and solve them, comparing the values you get in terms of the asymptotic running time. Then we'd like you to measure the actual running times and comment on the effect of the way the divide-and-conquer tree looks on the running time.

1.1 Logistics

Deadline: Friday September 22, 11:59pm Bloomington time (currently GMT-5).

What to turn in through canvas: your code for the two algorithms for $k = 3, 4$, a plot showing six different functions (three per algorithm – one for value of $k = 2, 3, 4$) and your comments on any choices you have made in writing the code (and difficulties you've had), how you generated and plotted your inputs, comments on how your plots look, and why. Also the recurrence relations for each algorithm for $k = 2, 3, 4$, and their solutions (for simplicity you can assume that the pivots for randomized quicksort divide the list evenly into k sublists), the resulting asymptotic complexities of each algorithm (a total of 6) and comments on how they compare to the actual running times and why. Is there an advantage or disadvantage in dividing into more pieces for divide-and-conquer in this case?

Grading: 20 points for turning in the code; 30 more if the code is working; 30 for the plots, and 20 for the comments, if your code/plots/comments are correct.

Late policy: you will lose 10 points for each late day (including holidays); if you have a good reason for being late, please contact the instructor and the AIs.

Please do NOT use the web. To be clear, in this class using code/answers from the web constitutes plagiarism. You may discuss coding aspects with your classmates (see below for reporting such discussions); you should write the code and do everything yourselves.

2 Doing the Work

2.1 Code and Output

Python3 is our official language for this class, so please make sure your submitted code meets the requirements below:

- be compatible with Python3.
- does not trigger any `TypeError`, `SyntaxError` or `ImportError`
- add your own test cases. The accompanying plain text file contains some test data, which you can turn into your test cases, but you should write more test cases to ensure the correctness of you functions. DO NOT READ test data elsewhere in your test cases.
- do not use the built-in `.sort` method or `sorted` method.
- your submitted files are a `.py` file that contains your implementation and test cases, and `.pdf` file that contains your plot and some descriptions if any.
- If you resort to resources other than the textbook (CLRS 3rd ed), please list your references in your pdf. But DO NOT USE CODE FOUND ON INTERNET. If you discuss with other fellow students in this class, please list their names as well.
- Use the builtin unit test framework or py.test

3 Tasks

Exercise 1

Implement a new merge sort function, `merge_sort_plus`, which expects two arguments. The first is an input array, and the other is k (where $2 \leq k \leq 4$). It returns the sorted input array. When $k = 3$, we split the list into three equal portions, sort each recursively, and merge the three sorted lists. Extending this to $k = 4$ is straightforward.

Exercise 2

Implement a new randomized quick sort function, `random_qs_plus`, which expects two arguments. The first is an input array, and the other is k (where $2 \leq k \leq 4$). It returns the sorted input array. When $k = 3$, instead of picking one random pivot, we pick two random pivots, let's call them p_1 and p_2 such that $p_1 \leq p2$ and split the list into three sublists. The first sublist contains elements $\leq p_1$, the second those are $> p_1$ and $\leq p_2$ and the third those $> p_2$. We then repeat the action on these sublists. Again, extending this to $k = 4$ is straightforward.

Exercise 3

Benchmark the two functions defined above with respect to different combinations of input. For each function , you should draw a line for a k. Given $k = 2, 3, 4$, you will need **three** lines to represent the performance of each function. We would like to see test arrays of at least 10000 elements (more is better). When you're plotting the running time, take your input at 50

equally spaced data points. For instance, if you're going up to an array size of 5,000, plot the running time for array sizes of 100, 200, 300, … 5000.