Kamden Ashmore

CSC212

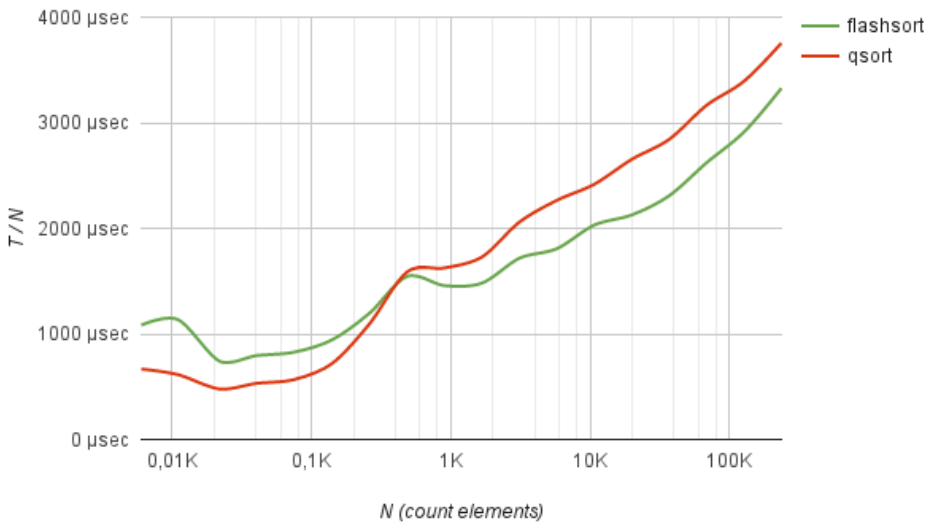
Flashsort

The flashsort algorithm was created by Karl-Dietrich Neubert in 1998. This in place sorting algorithm is very efficient with large, evenly distributed sets of data. It has a time complexity of sorting n elements in O(n) time. Traditionally the memory complexity of this algorithm is O(m) where m is the number of buckets and also m is 0.45 times n. We have altered this portion of the algorithm in our implementation and will go into further detail later in this paper.

The flashsort algorithm is built around the concept of divide and conquer. It accomplishes this by using buckets to separate similar data and sort it within those buckets. The previous steps are similar to any bucket sorting algorithm, but there are two things that separate flashsort and bucket sort. Before we dive into that we must firstly create m buckets where m is 0.45 * n. Then each n element is put into the corresponding m bucket. Then, we must convert bucket counts to prefix sums and then rearrange elements based on prefix sums. Prefix sums are counts that help the algorithm efficiently rearrange the elements in the input array. To calculate the prefix sum let Lb represent the prefix sum for bucket b, where L0 = 0 and Lm = n, with m being the number of buckets. Lb stores the cumulative count of elements in all buckets up to and including bucket b. With this prefix sum we can rearrange the elements. To do this we would iterate through the input array. For each element Ai we determine its bucket b based on its value. Place Ai in the position j such that $L_{b-1} < j \leq L_b$. In other words, place it after all elements in the previous

buckets but before the elements in the current bucket.Lastly we increment Lb to reflect the new position for the next element in the same bucket. By performing this rearrangement, we effectively group all elements belonging to the same bucket together while maintaining their relative order within each bucket. This rearrangement portion of the algorithm is a key optimization from other traditional sorting algorithms.

The pros of flashsort mainly focus on its memory efficient approach that hinges around large data sets. Additionally, the algorithm is in place, meaning that if a data point is in front of another piece of data with the same value it will remain in that same order. Flashsort being an optimization of bucket sort, another divide and conquer algorithm, focuses on sorting large data sets very efficiently. Having an average sort time of O(n), with uniformly distributed data sets, puts it ahead of a lot of other sorting algorithms.



The cons of quicksort revolve around its worst case scenario where data sets are not evenly distributed or have consistent outliers. On top of that flashsort is designed so that you would need to know the range of data you would be getting as input. Without this information

flashsort will not function as intended. Finally, flashsort is a very complicated algorithm to implement so if a program doesn't fully require a flashsort it is not necessary to force a flashsort implementation.

We chose flashsort in hopes that we could further improve this program to not only take in hundreds of data points, but thousands. Essentially, turning this program from a program marketed to only highschools to a program that could handle the biggest college. We are utilizing this algorithm to sort the students in ascending credit order so that we can prioritize higher credit students when picking classes for them. Additionally, the efficiency of this algorithm stuck out to us as a big upside.