Fall 2020 CS313 Prof. Fried

Group 5: Problem no. 5

Muhammad Abbas Ali Sajjad.

When it comes to sorting a list of elements there's always a few things to make sure of. The first being a sorting algorithm which correctly sorts the elements accurately every time, regardless of the size of the array. The second being a time efficient algorithm that computes sorting in a quick manner. The third being a memory competent sorting algorithm capable of completing its task without excessively allocating memory. There are several methods of evaluating these properties but we're going to focus on time and space efficiency.

Common sorting algorithms such as bubble sort, selection sort and insertion sort typically all share time complexities of O(n^2). Then, there are more advanced sorting algorithms like merge sort and quick sort that can be O(n*log(n)). The problem at hand was to incorporate two sorting algorithms at a multilevel approach. We would start with sortX and after a specific threshold of size n=10, implement sortY so as to sort the array by partitioning sections into sub-problems and aim for a time complexity of O(n*log(n)) if not better. The algorithmic approach we decided to base our multilevel sort on was to divide and conquer. A very popular approach to solving large problems by dividing them into smaller problems. Aiming to reduce the size of the array down to n, n arbitrarily being 10 in this case. SortX is a modified algorithm of merge sort and sortY is an insertion sort.

In order to reduce the size of the array without a loop was to use a recursive approach calling the multilevelsort() function repeatedly taking in int* array, starting index and ending index. Utilizing the basis of merge sort, which divides the sorting sequence into two procedures. The first being a dividing portion and the second being a merging portion. The left side of each array is passed into the function recursively until the size of the array is one. One element doesn't require sorting as it is sorted by default. The right side of the re-sized array is then evaluated down to one element. Then the merging portion of the algorithm is called. The merging of two halves creates two dynamically allocated arrays and fills them with the subsequent left and right side. Then the smallest element is added one by one from each respective sub-array until either goes out of bounds and then remaining elements of the other array are then ended.

SortY is this case is an insertion sort. A far simpler sorting algorithm, which evaluates each element and compares it to its predecessor. If the predecessor is smaller than the current value then a swap is initiated. The value is then compared against its new predecessor and the cycle repeats. The key to this algorithm is to evaluate every element after the 1st element since the index 0 element has no predecessor and resetting the index value after evaluating every predecessor. Hence, O(n^2).

Our multilevel sort implements sortX and after subsequent divides and the sub-array has a length of n=10 or less SortY is called inside sortX. The processing time of the multilevel sort was compared to Insertion sort, merge sort using C++11 <Chrono> library. An array with randomly allocated integers was sorted of size 10, 100, 1000, 100,000. Std::chrono::high_resolution_clock::now(); was only called before and after each sort call to ensure accurate processing times.

Average Processing times in microseconds

Array size	Insertion sort	Merge sort	Multilevel sort
10	1 µs	18 µs	12 µs
100	18 µs	90 µs	83 µs
1,000	1,719 µs	971 µs	949 µs
10,000	203,253 µs	9,051 µs	8,115 µs
100,000	19,694,137 µs	79, 806 µs	80,562 µs

Insertion sort had the best processing times when the size of the array was small, which is understandable because there is no recursive call stack to navigate. Insertion sort is also the least taxing on memory comparatively. When the size of our data structure is above 4000 bytes merge sort and the multilevel sort are far faster than insertion sort. The multilevel sort also appears slightly faster than merge sort when the size of the array is small, most probably due the inclusion of an insertion sort like process when the array is smaller than 10 recursively.