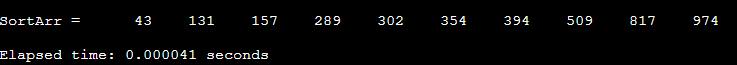
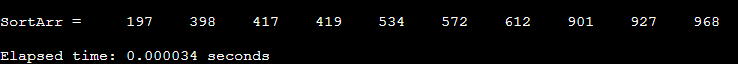
Radix sort, often referred to as bucket sort, is a linear-time sorting algorithm that organizes numbers by processing their digits individually, starting from the least significant digit and moving to the most significant digit. This method ensures that the array is progressively sorted at each digit level, ultimately resulting in a fully sorted list.

This C language implementation of radix sort sorts an array of randomly generated integers ranging from 0 to 999 using linked lists and recursion. The algorithm separates the numbers into "buckets" based on the value of their digits and recursively processes these buckets to achieve the final sorted order.

The RandomInt() function ensures a uniform distribution of the generated numbers. This is crucial for performance analysis, as the characteristics of the input can influence the time complexity of the algorithm. Having a fixed set of numbers will provide consistent time complexity which could fail in data validity.







If exact same set of data [43,131,157, …] were executed 10000 times the average elapsed time of the algorithm will be fixed at 0.000041 seconds. However, if random set of data were executed 10000 times, the average elapsed time of the algorithm will hold a different average. Thus, uniform distribution is a key element in sorting algorithm, for precise and accurate data.

The sorting process utilizes linked lists (struct node) to temporarily hold numbers in the appropriate buckets, which are organized by the value of the current digit being processed. This is done in the Recursive\_BucketSort() function, which groups numbers according to the current digit's value (ones place, tens place, etc.), and then recursively sorts them by the next more significant digit. The recursion terminates when the algorithm has processed all digits of the largest number in the array, as determined by the MaxCipher() function, which calculates the number of digits in the largest number in the array. Use of recursion in the implementation simplifies code complexity, however, it could lead to stack overflow issues when dealing with larger datasets.

An important aspect of radix sort is that its performance remains stable as long as the input data is uniformly distributed. Predefined or highly repetitive input arrays might result in suboptimal performance due to consistent data patterns. The random number generator helps mitigate this by providing a diverse set of input values for each run of the algorithm.

The algorithm's execution time is measured using the time() function. This provides an estimate of the time taken by the sorting process, which can be useful for evaluating the efficiency of the algorithm. The result is printed in seconds with a precision of six decimal places, giving a clear view of the algorithm’s performance on different inputs.

StackExchange User. (2021, December 7). Bucket sort has uniformly distributed data. Computer Science Stack Exchange. https://cs.stackexchange.com/questions/138691/bucket-sort-has-uniformly-distributed-data

Simplilearn. (2023, March 8). Radix sort: How radix sort algorithm works in data structure? Simplilearn. <https://www.simplilearn.com/tutorials/data-structure-tutorial/radix-sort>

Educative. (n.d.). *Comparison of linear-time sorting algorithms*. Educative. Retrieved September 17, 2024, from <https://www.educative.io/answers/comparison-of-linear-time-sorting-algorithms>