SUPER DUPER TAPE SORTING PROGRAM!

By Zachary Owen

**Sort 1: Single tape**

**Implementation:**

Sort 1 is implements a special insert sort that finds the biggest and smallest values between the search indexes. The program also keeps track of the values of the largest and smallest values. Keeping track of the indexes where the biggest and smallest were found the program then places the largest at the max index and smallest at the mindex and replaces the spots they were in in the tape at large with the values at the edges. The set function also needs to deal with the problem of max or min values being found on our edges. However it turns out every possible case can be delt with by placing the edge value of the oposite values index edge ( If the min value was found on any outer edge the max needs to print the value where it was found.). Cases where both are on edges are ignored since the maximum and minimum values are printed on the edges regardless in the correct order.

**Runtime**

This program runs slowly, but faster than the provided example. Its saving grace is its ability to place a single value in the correct place after a single full read of the tape on average. Since we find two values and then set two It works out on average.

This means the whole algorithm should run in about n^2\*(n-1)/8 for tape reads. This is due to a full read needing to be done for each value in the list and for each of those reads we will need to rewind except the last.

**Testing:**

Honestly we just threw a bunch of big number sets at it and each produced fully sorted lists.

**System Requirements:**

The only thing required by this program is memory space for 10 variables and a single tape drive. Also a C compiler.

**Sort 2: Triple Tape**

**Implementation:**

Sort 2 implements a bad binary radix sort. At run time the program generates a mask of 0x01 which is used to sort the lists into their buckets this radix was used due to the spare two tape drives we have avalable. I say it is a bad binary radix since it uses a comarison to determine which tape the value is written to when an array structure would allow us to skip that check. A good thing this program does is after writing the values to our tape bins, it writes the higher order number onto the end of the lower ordered drive. This saves us some time although the worst it would do is add 2 write and rewind cycles . Which is pretty bad at large values.

Another really neat function of this program is after we find the bit width of the largest value we manually do another sort before jumping into the loop. This has 2 features which follow from one another.

1. At the end of the while loop checking the number of iterations we’ve done the original tape drive/ output drive is populated with the numbers as sorted as the can be.

2. Even if an odd number of iterations is nessiary at the point where the numbers are sorted the mask will be on a bit that will be zero in every case. And with our delimited concatenation of the high order drive, The final pass will be almost as fast as a straight write.

**Runtime**

This program should run in a close to O ( log(n)( (26.5 n +1)/8 + 3) ). The program must run log base 2 of n iterations to sort each number by its highest possible bit width. During each iteration. The source tape is read once (N) however each value is written to one of the two other drives(n) at which point the high drive is delimited (+1) rewound ((.5n+1)/8 ) and its values written to the high drive ( n/2 + 1). At which point the high drive is rewound( n/2 +1), the low drive is rewound (n/8) and the source drive is rewound ( n/8). This anylisis relies on the assumption that half the values must be high order and half low order.

**Testing:**

Honestly we just threw a bunch of big number sets at it and each produced fully sorted lists. This program even ran in reasonable time on 100,000,000 values

**System Requirements:**

The only thing required by this program is memory space for 4 variables and three tape drives. Also a C compiler.

I’ve thought of a joke: Yo mama so fat IJVM has to preface her with a 0xC4 opcode, amirite?