CS 146: Data Structures and Algorithms

July 14 Class Meeting

Department of Computer Science
San Jose State University

Summer 2015

Instructor: Ron Mak

www.cs.sjsu.edu/~mak

Review of Sorting Algorithms

- Insertion sort
- Shellsort
- Heapsort
- Mergesort
- Quicksort

- What is going on with these sorts?

Demo

Analysis of Quicksort

- What is the running time to quicksort a list of N?
- Partition the array into two subarrays (constant cN time).
- A recursive call on each subarray.
- A recurrence relation:

$$T(N) = \begin{cases} 1 & \text{if } N = 0 \text{ or } 1 \\ T(i) + T(N - i - 1) + cN & \text{if } N > 1 \end{cases}$$

  - where i is the number of values in the left partition.

Analysis of Quicksort

- The performance of quicksort is highly dependent on ...
  - ... the quality of the choice of pivot.

Quicksort: Best Case Analysis

$$T(N) = \begin{cases} 1 & \text{if } N = 0 \text{ or } 1 \\ T(i) + T(N - i - 1) + cN & \text{if } N > 1 \end{cases}$$

- The pivot is always the median. Each subarray is the same size.

$$T(N) = 2T(N/2) + cN$$

Divide through by N:

$$\frac{T(N)}{N} = \frac{T(N/2)}{N/2} + c$$

Telescope:

$$\frac{T(N/2)}{N/2} = \frac{T(N/4)}{N/4} + c$$

$$\frac{T(N/4)}{N/4} = \frac{T(N/8)}{N/8} + c$$

$$\vdots$$

$$\frac{T(2)}{2} = \frac{T(1)}{1} + c$$

Add and cancel (there are $\log N$ equations):

$$\frac{T(N)}{N} = \frac{T(1)}{1} + c \log N$$

Therefore:

$$T(N) = N + cN \log N = \theta(N \log N)$$

Quicksort: Average Case Analysis

$$T(N) = \begin{cases} 1 & \text{if } N = 0 \text{ or } 1 \\ T(i) + T(N-i-1) + cN & \text{if } N > 1 \end{cases}$$

- Each size for a subarray after partitioning is equally likely, with probability $1/N$:

$$T(N-i-1) = \frac{1}{N}\sum_{j=0}^{N-1} T(j)$$

Since there are two partitions:

$$T(N) = \frac{2}{N}\left[\sum_{j=0}^{N-1} T(j)\right] + cN$$

Multiply by N:

$$NT(N) = 2\left[\sum_{j=0}^{N-1} T(j)\right] + cN^2 \quad \text{(a)}$$

Substitute N by N-1:

$$(N-1)T(N-1) = 2\left[\sum_{j=0}^{N-2} T(j)\right] + c(N-1)^2 \quad \text{(b)}$$

Subtract (a) – (b):

$$NT(N) - (N-1)T(N-1) = 2T(N-1) + 2cN - c$$

Quicksort: Average Case Analysis, *cont'd*

$$NT(N) - (N-1)T(N-1) = 2T(N-1) + 2cN - c$$

Rearrange and drop the insignificant $-c$:

$$NT(N) = (N+1)T(N-1) + 2cN$$

Divide through by $N(N+1)$:

$$\frac{T(N)}{N+1} = \frac{T(N-1)}{N} + \frac{2c}{N+1}$$

Telescope:

$$\frac{T(N-1)}{N} = \frac{T(N-2)}{N-1} + \frac{2c}{N}$$

$$\frac{T(N-2)}{N-1} = \frac{T(N-3)}{N-2} + \frac{2c}{N-1}$$

$$\vdots$$

$$\frac{T(2)}{3} = \frac{T(1)}{2} + \frac{2c}{3}$$

Add and cancel

Quicksort: Average Case Analysis, *cont'd*

Add and cancel:

$$\frac{T(N)}{N+1} = \frac{T(1)}{2} + 2c\sum_{i=3}^{N+1}\frac{1}{i}$$

Recall the harmonic number: $\sum_{i=3}^{N+1}\frac{1}{i} \approx \log_e N$

And so:

$$\frac{T(N)}{N+1} = O(\log N)$$

Therefore:

$$T(N) = O(N \log N)$$

Quicksort: Worst Case Analysis

$$T(N) = \begin{cases} 1 & \text{if } N = 0 \text{ or } 1 \\ T(i) + T(N - i - 1) + cN & \text{if } N > 1 \end{cases}$$

- The pivot is always the smallest value of the partition, and so $i = 0$.

$$T(N) = T(N-1) + cN$$

Telescope:

$$T(N-1) = T(N-2) + c(N-1)$$

$$T(N-2) = T(N-3) + c(N-2)$$

$$\vdots$$

$$T(2) = T(1) + c(2)$$

Add and cancel:

$$T(N) = T(1) + c\sum_{i=2}^{N} i = \theta(N^2)$$

Quicksort: Worst Case Analysis, *cont'd*

$$T(N) = \begin{cases} 1 & \text{if } N = 0 \text{ or } 1 \\ T(i) + T(N - i - 1) + cN & \text{if } N > 1 \end{cases}$$

- The pivot is always the smallest value of the partition, and so $i = 0$.

$$T(N) = T(1) + c\sum_{i=2}^{N} i = \theta(N^2)$$

- How does this explain the very bad behavior of quicksort when the data is already sorted?

Quicksort: Worst Case Analysis, *cont'd*

N = 100,000

ALGORITHM	MOVES	COMPARES	MILLISECONDS
Insertion sort	0	99,999	0
Shellsort suboptimal	0	1,500,006	4
Shellsort Knuth	0	967,146	4
Heap sort	1,900,851	3,882,389	12
Merge sort array	3,337,856	853,904	17
Merge sort linked list	1,115,021	815,024	29
Quicksort suboptimal	400,000	5,000,150,000	4,857
Quicksort optimal	400,000	1,968,946	6

Assignment #5

- Add heapsort, mergesort, and quicksort to your work for Assignment #4.
- Do two versions of mergesort:
  - Sort an array.
  - Sort a linked list.
- Do two versions of quicksort:
  - Suboptimal first element as the pivot choice.
  - Median-of-three pivot choice.

Assignment #5, *cont'd*

- Total sorts:
  - Insertion sort
  - Shellsort (two versions, optimal and suboptimal h sequences)
  - Heapsort
  - Mergesort (two versions, array and linked list)
  - Quicksort (two versions, optimal and suboptimal pivot choices)

Assignment #5, *cont'd*

- For each sort, your program should output:
  - How much time it took.
  - Count comparisons it made between two values.
  - Count moves it made of the values.
- Verify that your arrays are properly sorted!
- You should output these results
  in a single table for easy comparison.

Assignment #5, *cont'd*

- You may choose a partner to work with you on this assignment.
  - Both of you will receive the same score.
- Email your answers to ron.mak@sjsu.edu
  - Subject line:
    `CS 146 Assignment #5:` *Your Name(s)*
  - CC your partner when you email your solution.
- Due Friday, July 24 at 11:59 PM.

Break

A General Lower Bound for Sorting

- Any sorting algorithm that uses only comparisons requires $\Omega(N \log N)$ comparisons in the worst case.
- Prove: Any sorting algorithm that uses only comparisons requires $\lceil \log(N!) \rceil$ comparisons in the worst case and $\log(N!)$ comparisons on average.

$$\log(N!) = \Omega(N \log N)$$

A General Lower Bound for Sorting, *cont'd*

- Every sorting algorithm that uses only comparisons can be represented by a decision tree.
  - The number of comparisons is equal to the depth of the deepest leaf.

How many possible combinations for 3 elements?

3!

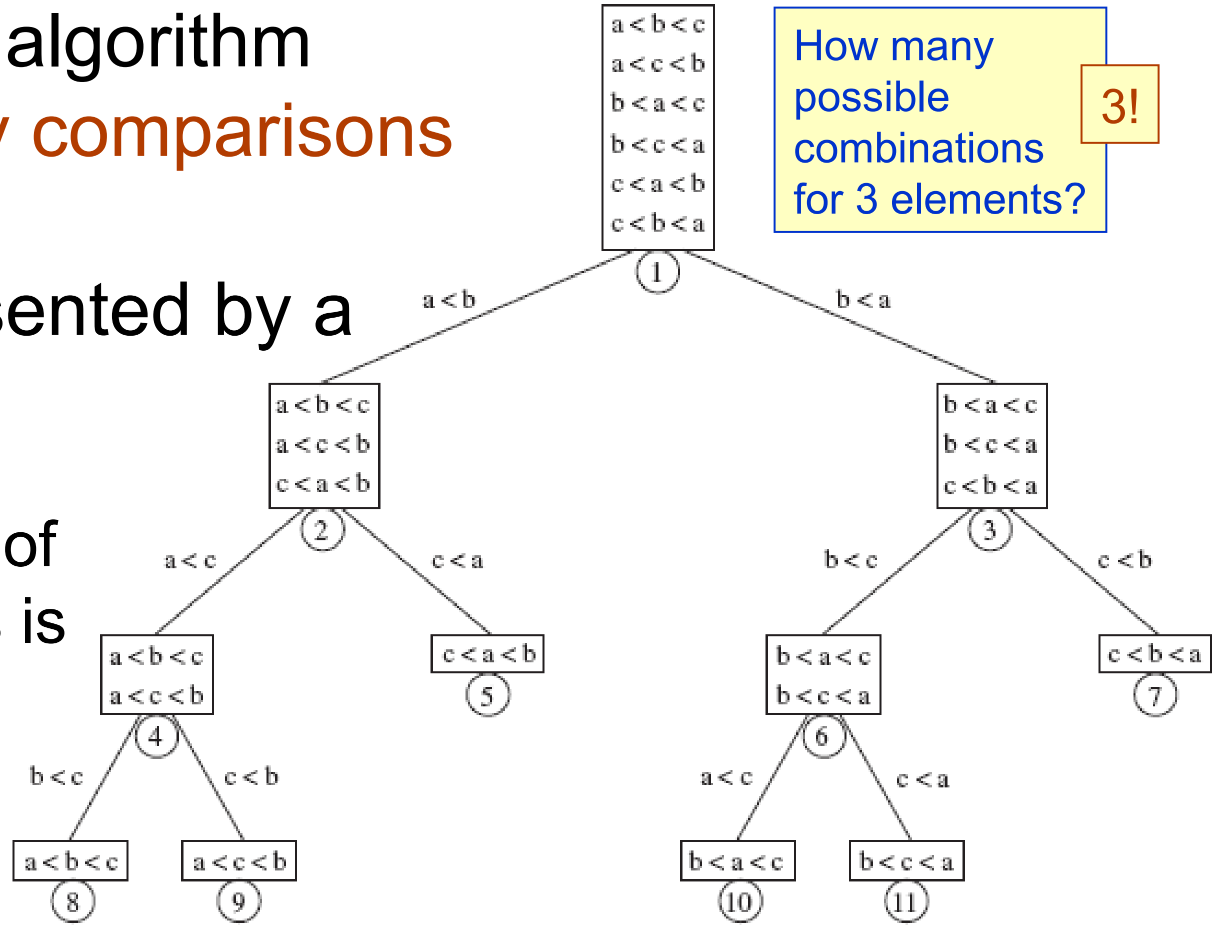


Figure 7.18 A decision tree for three-element sort

Some Decision Tree Properties

- A binary tree of depth d has at most 2^d leaves.
- A binary tree with L leaves must have depth at least $\lceil \log L \rceil$.
- Any sorting algorithm that uses only comparisons between elements requires at least $\lceil \log(N!) \rceil$ comparisons in the worst case.
  - A decision tree to sort N elements must have $N!$ leaves.

A General Lower Bound for Sorting, *cont'd*

- Prove: Any sorting algorithm that uses only comparisons between elements requires $\Omega(N \log N)$ comparisons.

A General Lower Bound for Sorting, *cont'd*

$$\log(N!) = \log(1 \bullet 2 \bullet 3 \bullet \cdots \bullet N) = \log(1) + \log(2) + \log(3) + \cdots + \log(N)$$

Delete the first half of the terms:

$$\geq \log\left(\frac{N}{2}\right) + \log\left(\frac{N}{2} + 1\right) + \log\left(\frac{N}{2} + 2\right) + \cdots + \log N$$

Replace each remaining term by the smallest one, $\log(N/2)$:

$$\geq \log\left(\frac{N}{2}\right) + \log\left(\frac{N}{2}\right) + \log\left(\frac{N}{2}\right) + \cdots + \log\left(\frac{N}{2}\right)$$

There are $N/2$ of these $\log(N/2)$ terms:

$$= \frac{N}{2}\log\left(\frac{N}{2}\right) = \frac{N}{2}\log\left(N \bullet 2^{-1}\right) = \frac{N}{2}\left[(\log N) - 1\right] = \frac{N}{2}\log N - \frac{N}{2}$$

Therefore:

$$\log(N!) = \Omega(N \log N)$$

A General Lower Bound for Sorting, *cont'd*

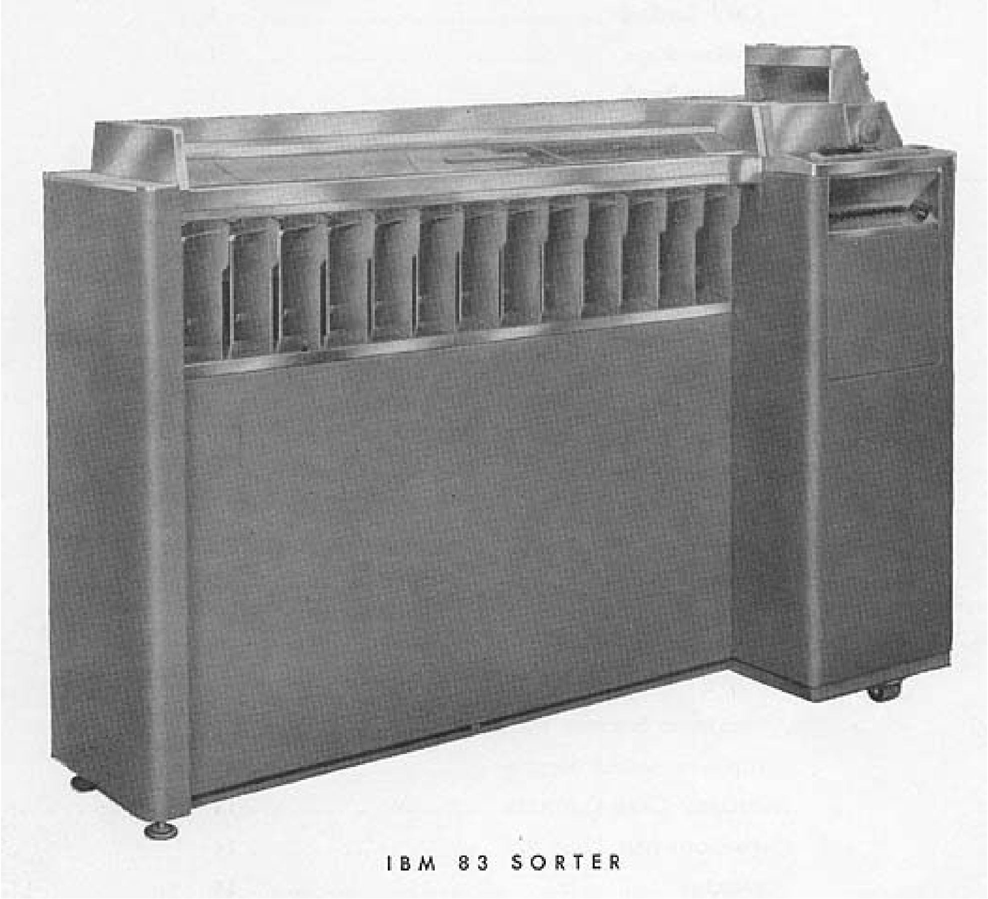
$$\log(N!) = \Omega(N \log N)$$

- Therefore, you cannot devise a sorting algorithm based on comparing elements that will be faster than $\Omega(N \log N)$ in the worst case.

Bucket Sort and Radix Sort

- Bucket sorting relies on using a number of bins, or buckets, into which the values to be sorted are entered.
- Sorting time is linear rather than $O(N \log N)$.
  - Does <u>not</u> rely on comparisons.
- A form of bucket sort is the radix sort.
  - Used to sort values each of which has a limited number of characters.
    - Example: 3-digit numbers.
  - Radix sort was used by the old electromechanical IBM card sorters to sort punched cards.

IBM 083 Card Sorter



- 1950s vacuum-tube and mechanical technology.
  - Sorted up to 1000 cards per minute.

Punched Cards

- A punched card had up to 12 punches per column, numbered 0-9 and 11 and 12.
  - The card sorter had 12 bins (plus a reject bin).

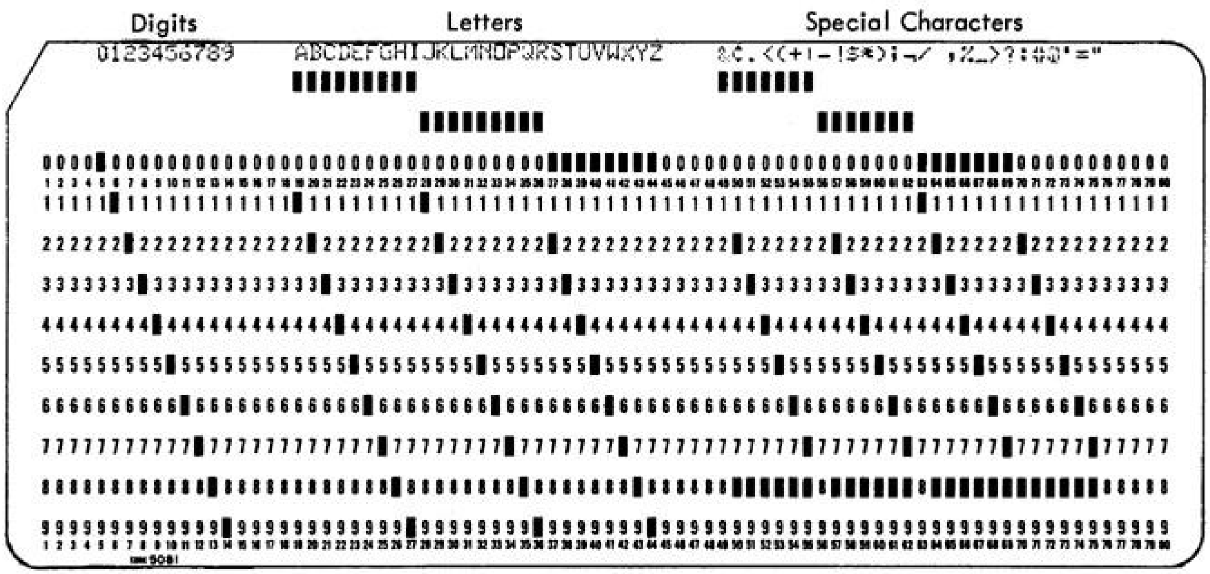
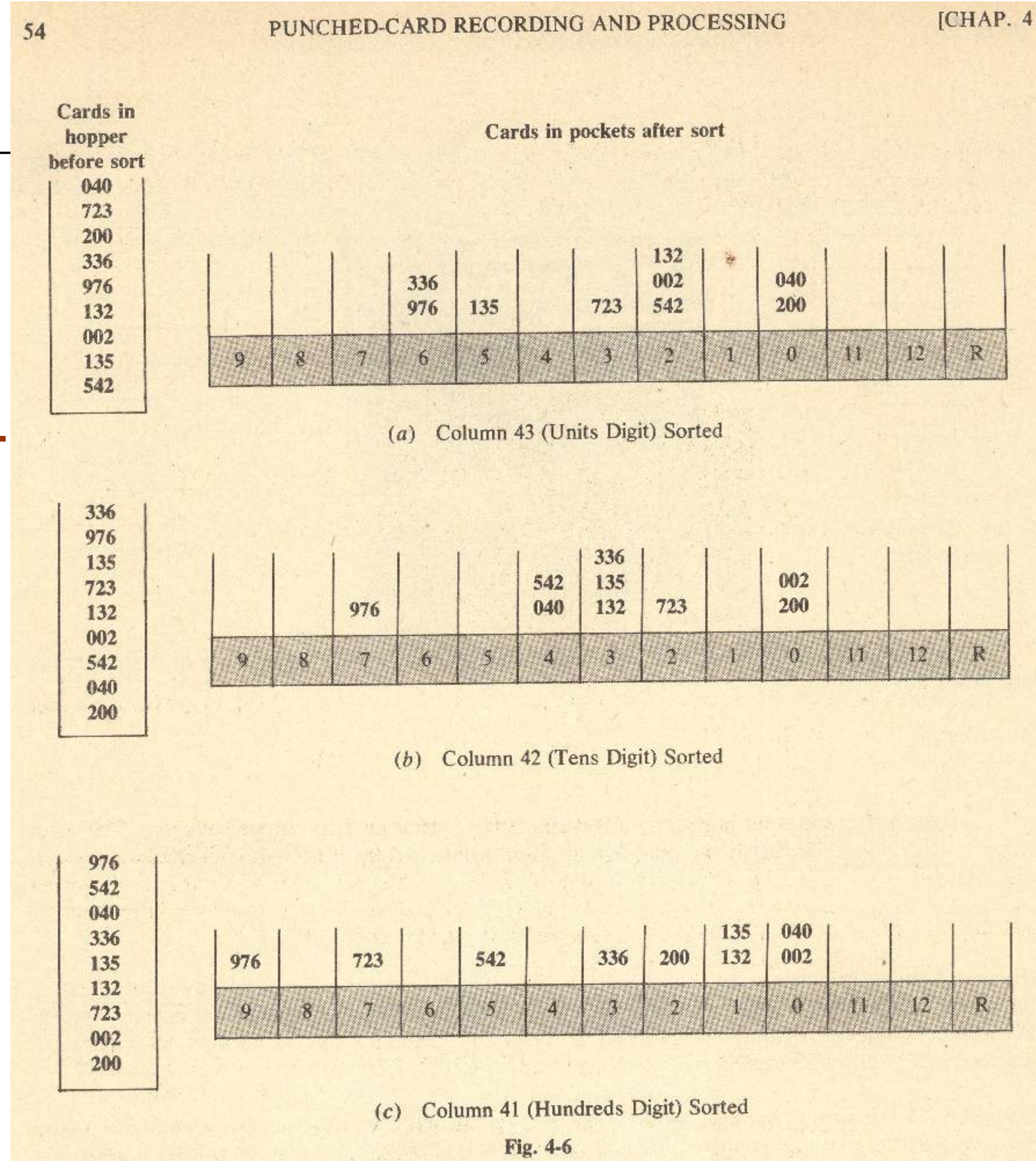


Figure 4. Card Codes and Graphics for 64-Character Set

Sorting Punched Cards

- How to sort cards punched with 3-digit numbers (in the same columns):
  - First sort on the units digit.
    - Each card drops into the appropriate bin based on the units digit.
    - Carefully remove the cards from the bins, keeping them in order.
  - Next sort on the tens digit.
    - Each card drops into the appropriate bin based on the 10s digit.
    - Carefully remove the cards from the bins, keeping them in order.
  - Finally sort on the hundreds digit.
    - Each card drops into the appropriate bin based on the 100s digit.
    - Carefully remove the cards from the bins, keeping them in order.

Radix sorting with an old electromechanical punched card sorter.



Magnetic Tape Sorting

- Most magnetic tapes can be read and written in one direction only.
  - You can also rewind a tape.

Magnetic Tape Sorting, *cont'd*

Magnetic Tape Sorting, *cont'd*

- Suppose you have data you want to sort.
- The unsorted data records initially all reside on one magnetic tape.

- You have 4 tape drives and 3 blank tapes.
- The computer's memory can hold and sort only 3 data records at a time.

- Perform an external merge sort.

Magnetic Tape Merge Sort

T1	81 94 11 96 12 35 17 99 28 58 41 75 15
T2	
T3	
T4	

T1	
T2	
T3	11 81 94 17 28 99 15
T4	12 35 96 41 58 75

T1	11 12 35 81 94 96 15
T2	17 28 41 58 75 99
T3	
T4	

Can you follow what's happening? (These slides are animated.)

Magnetic Tape Merge Sort, *cont'd*

T1	11 12 35 81 94 96 15
T2	17 28 41 58 75 99
T3	
T4	

T1	
T2	
T3	11 12 17 28 35 41 58 75 81 94 96 99
T4	15

T1	11 12 15 17 28 35 41 58 75 81 94 96 99
T2	
T3	
T4	