# Real-world Question

- Consider 16 GB integers or more to sort, given 4GB memory or less. (A large portion of data must be on disks)
  1. How would you do the sorting?
  2. Any previous sorting algorithm works for this problem?
  3. Even more data to sort: 1 Petabytes? (data scattered across different storage machines)

# Previous Sorting Algorithms

- Previous sorting schemes were all internal sorting algorithms:
  - *required direct access to list elements*
    - not possible for sequential files
  - *made many passes through the list*
    - not practical for files
- Assume ALL DATA IN MEMORY!

# Mergesort

Dr. Yingwu Zhu

# Mergesort

- Sorting schemes are either …
  - **internal** -- designed for data items stored in main memory
  - **external** -- designed for data items stored in secondary memory, e.g., disks

# Mergesort

- **Mergesort** can be used both as an internal and an external sort.
- Algorithm design: Divide-and-Conquer
- Basic operation in mergesort is **merging**,
  - combining two lists that have previously been sorted
  - resulting list is also sorted.

# MergeSort

- Given a single file


  - Split into two files
  - Apply the Mergesort algorithm on the resulting two files independently and recursively, until it is simple enough (e.g., quicksort)
  - Merge them in a bottom-up fashion

# Mergesort: recursive pseudo-code

```
void mergesort(int F[], int size) {
        if (size can fit in memory or simple enough) {
                sort it using other internal sorting algs
                return
        }
        (equally) split F into F1, F2 //write it into two smaller files
        mergesort(F1, size1);
        mergesort(F2, size2);
        merge F1 and F2 into F; //merge operation
}
```

# Merge Algorithm

1. Open `File1` and `File2` for input, `File3` for output
2. Read first element `x` from `File1` and
   first element `y` from `File2`
3. While neither eof `File1` or eof `File2`
   If `x < y` then
      a. Write `x` to `File3`
      b. Read a new `x` value from `File1`
   Otherwise
      a. Write `y` to `File3`
      b. Read a new `y` from `File2`
   End while
4. If eof `File1` encountered copy rest of of `File2` into `File3`.
   If eof `File2` encountered, copy rest of `File1` into `File3`

# Merge Algorithm

- void merge(int F[ ], int& n, int F1[ ], int n1, int F2[ ], int n2);
  //F1 and F2 are sorted, the resulting F is sorted

# Mergesort

- T(n) = ?

# Mergesort

- $T(n) = O(n \log_2 n)$, why?