Sorting #3 - The Merge of the King

CptS 223 - Fall 2017 - Aaron Crandall

Today’s Agenda

- Announcements
- Thing of the day
- Sorting is fun! - Time for some merge sort

Announcements

- Looks like our midterm #2 will be...
  - Any major objections?
  - Topics include: Hashing, Heaps, Sorting

To answer the “Is heap sort stable” question:

No.

- More specifically, consider sorting:
  - Input list: [21 20a 20b 12 11 8 7]
    It will come out: [7 8 11 12 20b 20a 21]
- There’s also opportunity for much larger heaps to cause unstable behavior if equal elements are in different subtrees with well crafted parents.

Playing with Oracle of Bacon

- All movie performers are really close to Kevin Bacon
  - https://oracleofbacon.org/
- This is a graph traversal problem finding shortest paths
  - Dijkstra's Algorithm
- We'll look at it more after Midterm #2

TotD: Robots doing to-home grocery delivery?

“These six-wheeled robots are about to start delivering food in the US”

Door to door delivery of groceries and small goods. Mostly autonomous, but always learning. Currently being tested in D.C., as it shown to my right. →

Mergesort! - It’s a way to come together

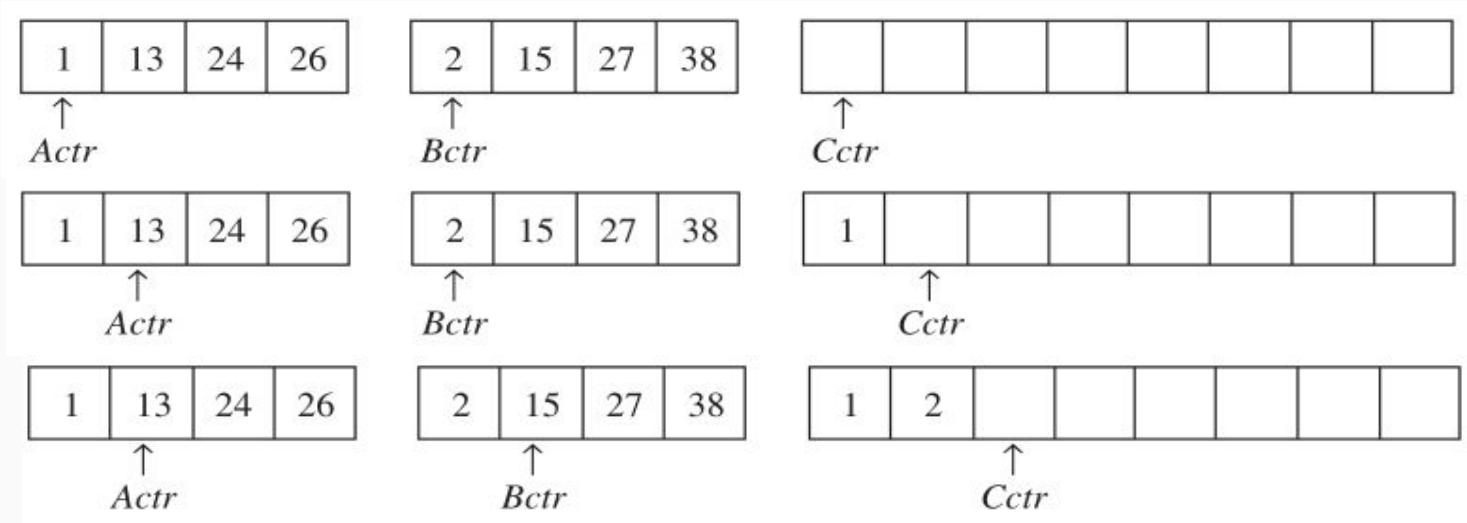
- Runs in O(N log N) time
- Number of comparisons nearly optimal
- Fundamentally, it’s merging two lists together
  - The lists are already sorted (by definition)
  - Results go in a 3rd list - beware space complexity!
    - Can be mitigated by using in-place swaps at cost of time
- FYI: Invented by John von Neumann in 1945
  - Was ranked as a top secret algorithm by the US military at the time

John von Neumann
https://goo.gl/ORD7D

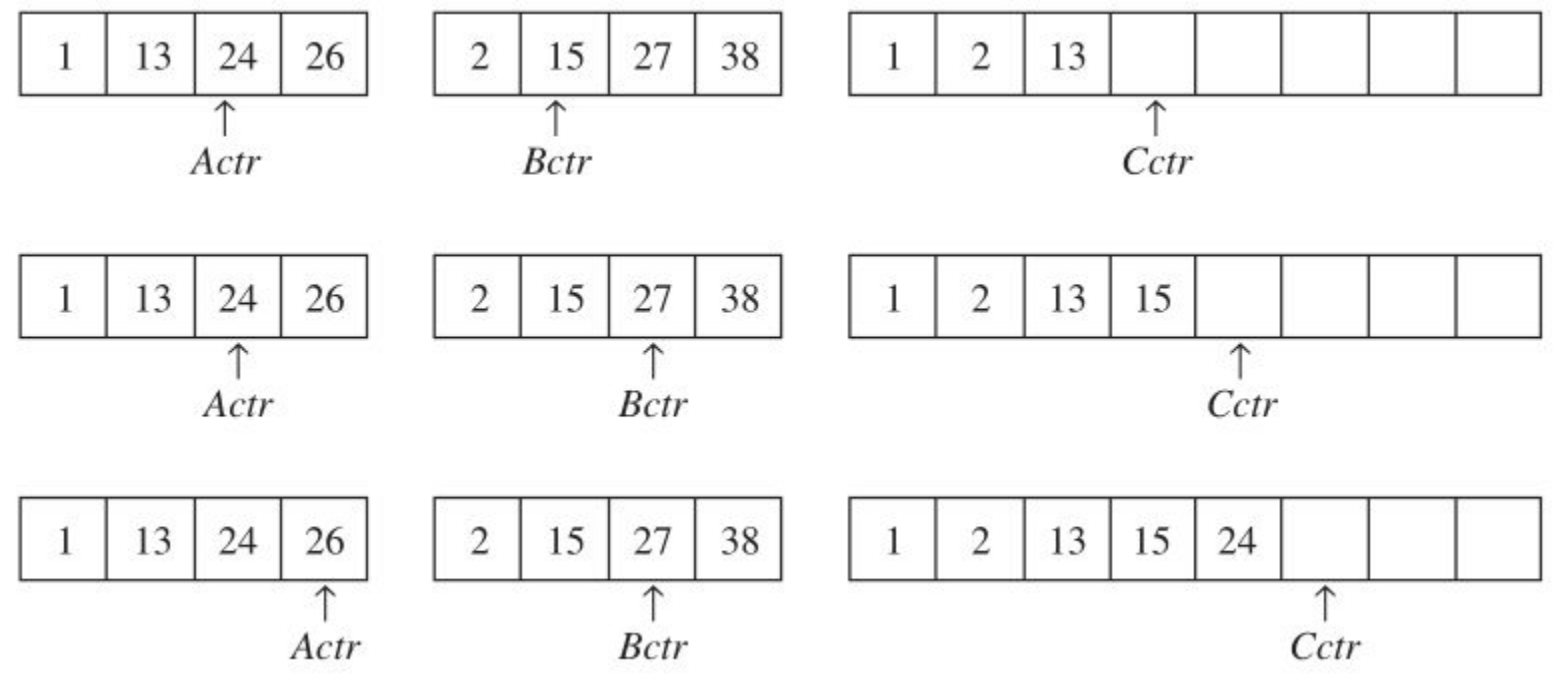
- "the last representative of the great mathematicians" like Euler, Gauss, Poincaré or Hilbert”
- Pioneer of operator theory to quantum mechanics
- Key dev in game theory
- Cellular automata, the universal constructor and the digital computer
- 150 papers: 60 math, 20 physics, 60 applied math - last one from his hospital bed became a book “The Computer and The Brain”
- Analysis of self replication work preceded DNA
- This is just the tip of the iceberg - we owe him so very much

Top level view of merging lists in general

- Lists: A, B, C
- Counters: Actr, Bctr, Cctr

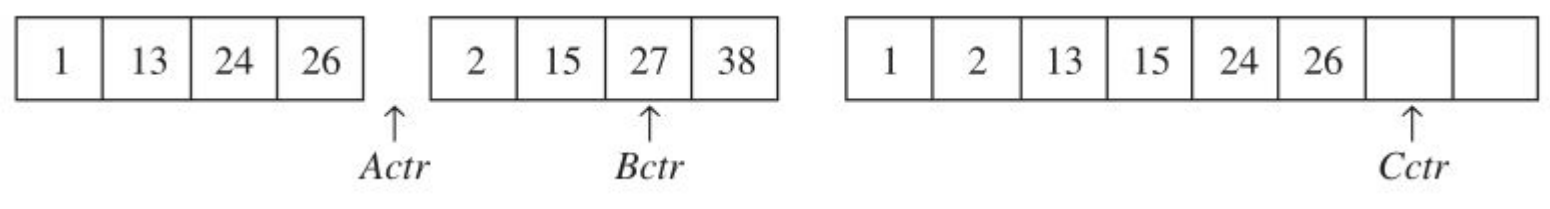


And so on


1 13 24 26
Actr
2 15 27 38
Bctr
1 2 13
Cctr
1 13 24 26
Actr
2 15 27 38
Bctr
1 2 13 15
Cctr
1 13 24 26
Actr
2 15 27 38
Bctr
1 2 13 15 24
Cctr

Short circuit if one of the lists ends early

- Remainder of B gets copied without comparisons into C when A runs out.



Time to merge two lists

- Clearly linear: at most N-1 comparisons are made
- The question is: how to get two sorted lists?

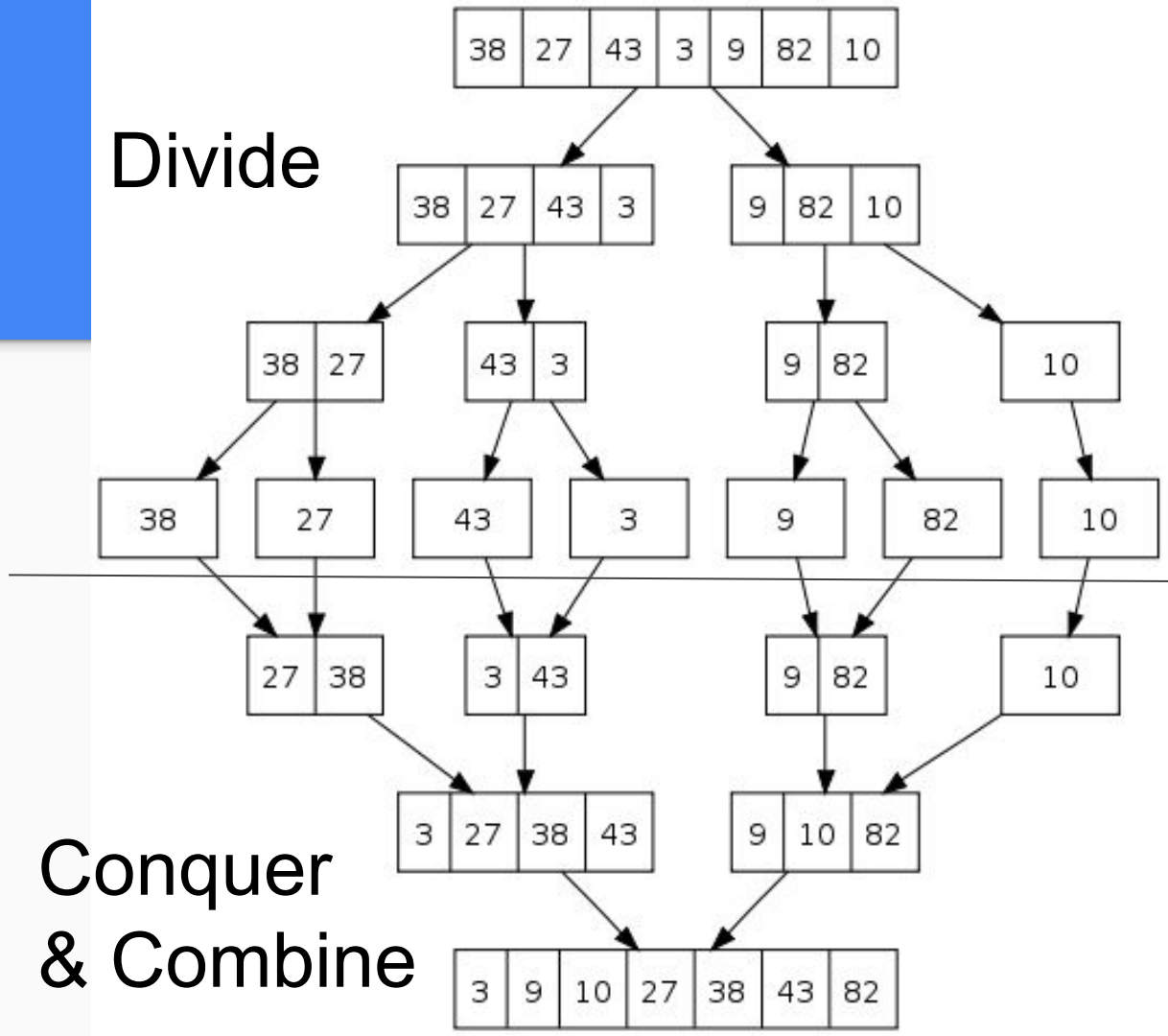
- The answer is:

  Recursively subdivide the original list and sort those subdivided lists

For example

- To sort: [24,13,26,1,2,27,38,15]
- Recursively sort the first 4 and last 4 elements to get:
  [1,13,24,26,2,15,27,38]
- Merge the two halves to get: [1,2,13,15,24,26,27,38]
- This is a classic divide and conquer approach
  - Divide the problem into subproblems to be solved recursively
  - Conquer by patching together the solved pieces
- Is this algorithm a stable sort?

Other example



Analyzing Mergesort

- Recursive analysis for this algorithm
- Assume N is a power of 2 (to make it easy)
- For N == 1, time to merge is O(1)
- Time to mergesort N numbers is time to do two recursive mergesorts of size N/2, plus time to merge, which is linear (That N-1 we saw earlier)
  - T(1) = 1
  - T(N) = 2T(N/2) + N

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

$$\frac{T(N/2)}{N/2} = \frac{T(N/4)}{N/4} + 1$$

$$\frac{T(N/4)}{N/4} = \frac{T(N/8)}{N/8} + 1$$

$$\vdots$$

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

This series sums up log N times and mostly cancels out to:

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

Multiply by N on both sides for:

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

Limits of Mergesort

- Although it's O(N log N) time:
  - Requires linearly O(N) extra space for the merging
  - Has to copy to temp array and back with each layer of recursion
    - Can be avoided by swapping a and tmpArray as you alternate recursive calls
- Heavily dependent upon the wallclock costs of comparing elements and moving elements in the array - This is language implementation dependent
  - Java: comparisons are slow, moves are fast -> mergesort is default sort in the libraries
  - C++: copying can be expensive, comparing fast -> Quicksort default in libraries
  - C++11 has better copying semantics, so this might change, but it hasn't yet

Our Book's Mergesort Implementation

```
template <typename Comparable>
void mergeSort( vector<Comparable> & a,
                vector<Comparable> & tmpArray, int left, int right )
{
    if( left < right )
    {
        int center = ( left + right ) / 2;
        mergeSort( a, tmpArray, left, center );
        mergeSort( a, tmpArray, center + 1, right );
        merge( a, tmpArray, left, center + 1, right );
    }
}
```

```
template <typename Comparable>
void merge( vector<Comparable> & a, vector<Comparable> & tmpArray,
            int leftPos, int rightPos, int rightEnd )
{
    int leftEnd = rightPos - 1;
    int tmpPos = leftPos;
    int numElements = rightEnd - leftPos + 1;

    // Main loop
    while( leftPos <= leftEnd && rightPos <= rightEnd )
        if( a[ leftPos ] <= a[ rightPos ] )
            tmpArray[ tmpPos++ ] = std::move( a[ leftPos++ ] );
        else
            tmpArray[ tmpPos++ ] = std::move( a[ rightPos++ ] );
```

```
    while( leftPos <= leftEnd )    // Copy rest of first half
        tmpArray[ tmpPos++ ] = std::move( a[ leftPos++ ] );

    while( rightPos <= rightEnd )  // Copy rest of right half
        tmpArray[ tmpPos++ ] = std::move( a[ rightPos++ ] );

    // Copy tmpArray back
    for( int i = 0; i < numElements; ++i, --rightEnd )
        a[ rightEnd ] = std::move( tmpArray[ rightEnd ] );
}
```

Mergesort summary:

- Operates in O(N log N) time - DOES NOT go faster for sorted data?
- Is stable
- Can take O(N) extra space unless you do extra work
- Invented by John Von Neumann in 1945
  - This name will come up again in your studies…
  - Was originally classified as Top Secret
  - Invented the Von Neumann computer architecture, which your computers are using now
- The sort used by Java because it’s nearly optimal in comparisons