Mergesort

```
MERGE-SORT ( A )
    if len(A) < 2
        return copy of A
    else if len(A) = 2
        if A[0] ≤ A[1] return A[0], A[1]
        else return A[1], A[0]
    else
        L ← 1st half of A      } divide     O(n)   (A is linked list)
        R ← rest of A          }
        L ← MERGE-SORT(L)      } conquer
        R ← MERGE-SORT(R)      }
        return MERGE(L, R)     combine      O(n)   (A is array or linked list)
```

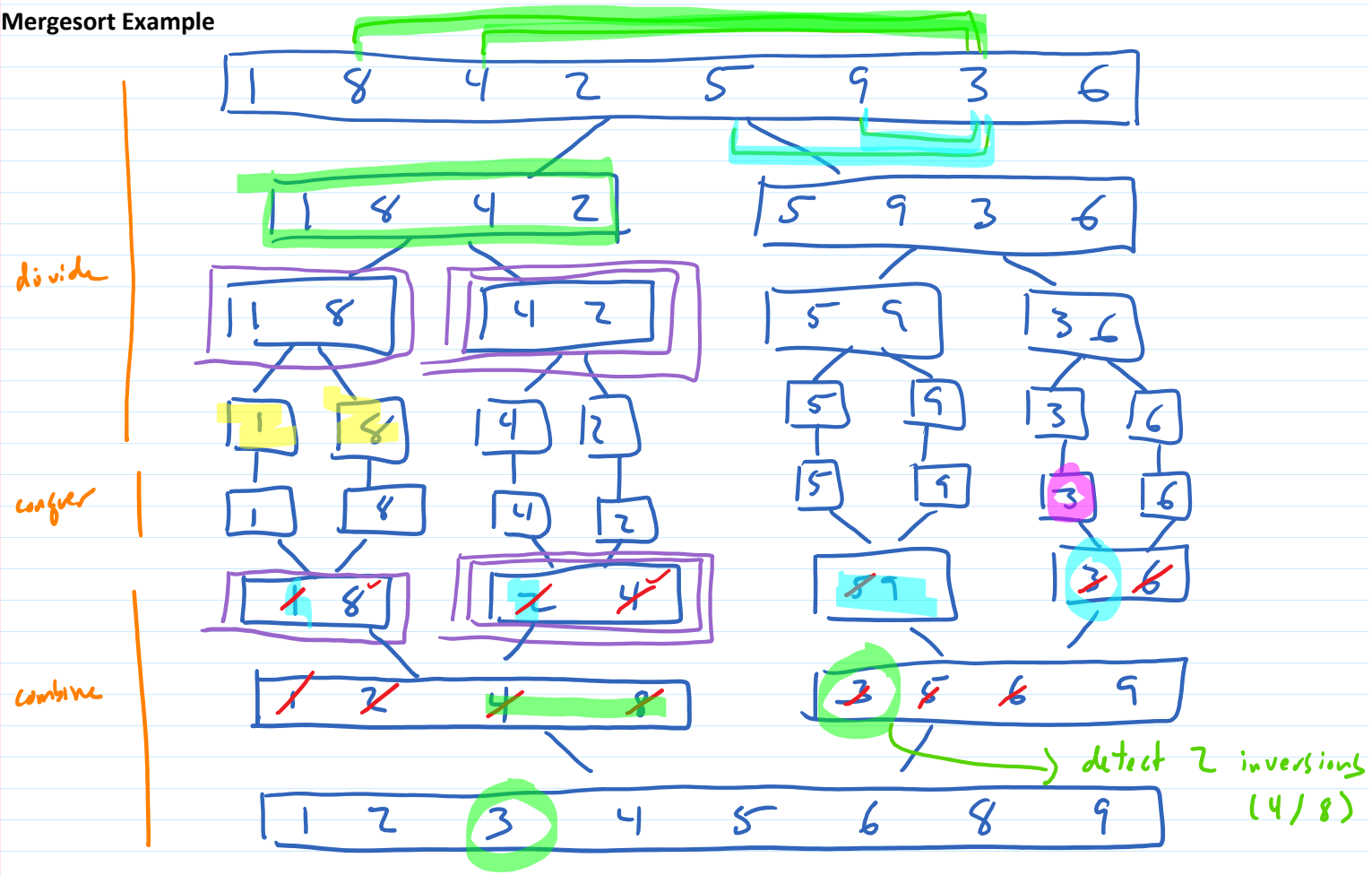
```
MERGE(L, R)

    A ← empty list                          to hold merged list
    i ← 0
    j ← 0
    while i < len(L) and j < len(R)         until one list empty
        if L[i] ≤ R[j]                      which is smaller?
            append L[i] to A                — copy smaller to result
            i ← i+1
        else
            append R[j] to A
            j ← j+1

    append L[i ... len(L)-1] to A
    append R[j ... len(R)-1] to A

    return A
```

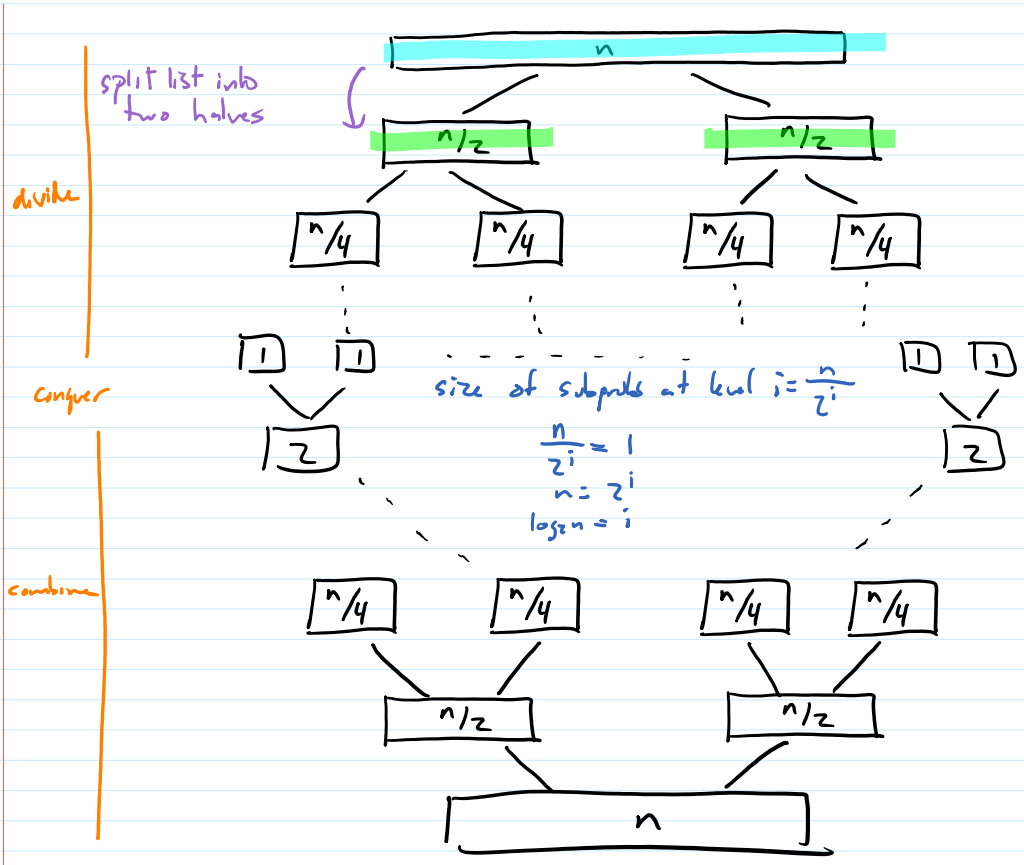
O(n) for array or linked list

Divide-and-Conquer

divide
conquer
combine

Mergesort Example





$\leq cn$

$\leq 2 \cdot c \cdot \frac{n}{2} = cn$

$\leq 4\left(c \cdot \frac{n}{4}\right) = cn$

$\vdots$

$\sum_{i=1}^{\log_2 n} cn = \log_2 n \cdot cn$

$= O(n \log n)$

$T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + c \cdot n$

$\leq 2 \cdot c \frac{n}{2} \log \frac{n}{2} + cn$

$= cn(\log n - 1) + cn$

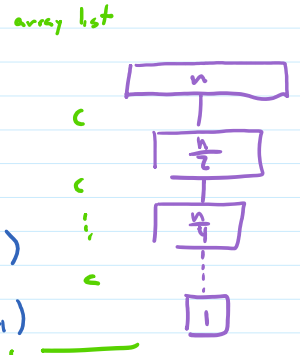
$= cn \log n$ ~~$- cn + cn$~~

Binary Search on Linked List (and why that's pointless)

LINKED-BINARY-SEARCH (A, key)

```
if A is empty
    return FALSE
else
    mid = size(A) / 2
    if A[mid] = key                    O(n) if A is linked list
        return TRUE                    O(1) if A is array list
    else if key < A[mid]
        return LINKED-BINARY-SEARCH( A0...mid-1 )
    else                    (combine (only 1 subproblem, so trivial))
        return LINKED-BINARY-SEARCH( Amid+1...length(A)-1 )
```

divide



$\sum_{i=0}^{\log_2 n} c = c \log_2 n$

$O(\log n)$

$\sum_{i=0}^{\log_2 n} \frac{cn}{2^i}$

$= cn \sum_{i=0}^{\log_2 n} \left(\frac{1}{2}\right)^i$

$\leq cn \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i$

$= 2cn$ $O(n)$

no better than sequential search

Counting Inversions

Rank the following

	Orioles	Mariners	Tigers	Red Sox	Yankees
Jim	1	2	3	4	5
Larissa	3	4	5	2	1
Sheldon	3	2	4	1	5
Tucker	3	4	1	2	5

for 2 orderings, count # inversions

pair of entries out of order

J / L: 3 4 5 2 1 — 7

T / L: 5 2 3 4 1 — 7

Larissa should watch games with Sheldon rather than Tucker or Jim

S / L: 2 4 3 5 1 — 5

Brute force: check each pair $O(n^2)$

```
MERGE_AND_COUNT(L, R)
    count ← 0
    A ← empty list
    i ← 0
    j ← 0
    while i < len(L) and j < len(R)
        if L[i] ≤ R[j]
            append L[i] to A
            i ← i+1
        else count ← len(L) − i
            append R[j] to A        inversion between R[j] and L[i], L[i+1], ... L[len(L)-1]
            j ← j+1
    append L[i ... len(L)-1] to A
    append R[j ... len(R)-1] to A
    return A, count
```

COUNT-AND-SORT (A)

$L \leftarrow$ 1st half of A
$R \leftarrow$ 2nd half of A

$O(n \log n)$ just like mergesort

L, countL ← COUNT_AND_SORT (L)
R, countR ← C_A_S (R)

M, countM ← M_A_C (L, R)

return M, countL + countR + countM

Integer Multiplication

862 341 · 979 468

x_1 = 862, x_0 = 341, y_1 = 979, y_0 = 468

```
      8 6 2 3 4 1
      9 7 9 4 6 8
    -------------
    6 8 9 8 7 2 8
  5 1 7 4 0 4 6
3 4 4 9 3 6 4
        ⋮
-------------
```

$O(n^2)$

$x = x_1 \cdot 1000 + x_0$

$y = y_1 \cdot 1000 + y_0$

$x \cdot y = (x_1 \cdot 1000 + x_0) \cdot (y_1 \cdot 1000 + y_0)$

$= x_1 \cdot y_1 \cdot 10^6 + x_1 \cdot y_0 \cdot 10^3 + y_1 \cdot x_0 \cdot 10^3 + x_0 \cdot y_0$

4 multiplications of $\frac{n}{2}$ digit nums

3 additions, 3 shifts] $O(n)$

$xy = x_1 \cdot y_1 \cdot 10^6 + x_1 \cdot y_1 \cdot 10^3 - 10^3 (x_1 - x_0)(y_1 - y_0) + 10^3 \cdot x_0 \cdot y_0 + x_0 \cdot y_0$

3 multiplications of $\frac{n}{2}$ digit nums

3 additions, 3 subtractions, 4 shifts] $O(n)$

Java implementation

pre: both same # of digits (0-pad if not)

```
BigInteger karatsuba(String s1, String s2)
{
  int m = s1.length() / 2;

  if (m == 1)
    return new BigInteger(s1).multiply(new BigInteger(s2));

  String x1 = s1.substring(0, m);
  String x0 = s1.substring(m);
  String y1 = s2.substring(0, m);
  String y0 = s2.substring(m);

  BigInteger tenm = (new BigInteger("10")).pow(s1.length() - m);
  BigInteger ten2m = tenm.multiply(tenm);
  BigInteger z0 = multiply(x0, y0);
  BigInteger z2 = multiply(x1, y1);
  BigInteger z1 = multiply((new BigInteger(x1)).subtract(new BigInteger(x0)).toString(),
                     (new BigInteger(y1)).subtract(new BigInteger(y0)).toString());

  return z2.multiply(ten2m).add(z2.multiply(tenm))
                           .subtract(z1.multiply(tenm))
                           .add(z0.multiply(tenm))
                           .add(z0);
}
```