2. Recurrences
2. Recurrences

- Computing values
- Telescoping
- Types of recurrences
- Mergesort
- Master Theorem
What is a recurrence?

**Def.** A *recurrence* is an equation that recursively defines a sequence.

Familiar example 1: *Fibonacci numbers*

recurrence

\[ F_N = F_{N-1} + F_{N-2} \text{ for } N \geq 2 \text{ with } F_0 = 0 \text{ and } F_1 = 1 \]

sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots

Q. Simple formula for sequence (function of \( N \))?
What is a recurrence?

Recurrences directly model costs in programs.

Familiar example 2: Quicksort (see lecture 1)

recurrence

\[ C_N = N + 1 + \sum_{0 \leq k \leq N-1} \frac{1}{N} (C_k + C_{N-k-1}) \]

for \( N \geq 1 \) with \( C_0 = 0 \)

sequence

0, 2, 5, 8 2/3, 12 5/6, 17 2/5, ...

program

```java
public class Quick {
    private static int partition(Comparable[] a, int lo, int hi) {
        int i = lo, j = hi+1;
        while (true) {
            while (less(a[++i], a[lo])) if (i == hi) break;
            while (less(a[lo], a[--j])) if (j == lo) break;
            if (i >= j) break;
            exch(a, i, j);
        }
        exch(a, lo, j);
        return j;
    }

    private static void sort(Comparable[] a, int lo, int hi) {
        if (hi <= lo) return;
        int j = partition(a, lo, hi);
        sort(a, lo, j-1);
        sort(a, j+1, hi);
    }
}
```
Common-sense rule for solving any recurrence

Use your computer to compute values. \( F_N = F_{N-1} + F_{N-2} \) for \( N \geq 2 \) with \( F_0 = 0 \) and \( F_1 = 1 \)

Use a recursive program?

```java
public static void F(int N) {
    if (N == 0) return 0;
    if (N == 1) return 1;
    return F(N-1) + F(N-2);
}
```

NO, NO, NO: Takes exponential time!

Instead, save all values in an array.

```java
long[] F = new long[51];
F[0] = 0; F[1] = 1;
if (N == 1) return 1;
for (int i = 2; i <= 50; i++)
    F[i] = F[i-1] + F[i-2];
```
Common-sense starting point for solving any recurrence

Use your computer to compute initial values.


http://algs4.cs.princeton.edu
Common-sense starting point for solving any recurrence

Use your computer to compute initial values (modern approach).

Ex. 1: *Fibonacci*  \( F_N = F_{N-1} + F_{N-2} \) with \( F_0 = 0 \) and \( F_1 = 1 \)

```java
public class Fib implements Sequence {
    private final double[] F;

    public Fib(int maxN) {
        F = new double[maxN+1];
        F[0] = 0; F[1] = 1;
        for (int N = 2; N <= maxN; N++)
            F[N] = F[N-1] + F[N-2];
    }

    public double eval(int N) {
        return F[N];
    }

    public static void main(String[] args) {
        int maxN = Integer.parseInt(args[0]);
        Fib F = new Fib(maxN);
        for (int i = 0; i < maxN; i++)
            StdOut.println(F.eval(i));
    }
}
```

 `% java Fib 15
 0.0
 1.0
 1.0
 2.0
 3.0
 5.0
 8.0
 13.0
 21.0
 34.0
 55.0
 89.0
 144.0
 233.0
 377.0`
Common-sense starting point for solving any recurrence

Ex. 2: Quicksort

\[ NC_N = (N + 1)C_{N-1} + 2N \]

**QuickSeq.java**

```java
public class QuickSeq implements Sequence {
    private final double[] c;

    public QuickSeq(int maxN) {
        c = new double[maxN+1];
        c[0] = 0;
        for (int N = 1; N <= maxN; N++)
            c[N] = (N+1)*c[N-1]/N + 2;
    }

    public double eval(int N) {
        return c[N];
    }

    public static void main(String[] args) {
        // Similar to Fib.java.
    }
}
```

% java QuickSeq 15
0.000000
2.000000
5.000000
8.666667
12.833333
17.400000
22.300000
27.485714
32.921429
38.579365
44.437302
50.477056
56.683478
63.043745
69.546870
Common-sense starting point for solving any recurrence

Use your computer to **plot** initial values.

```java
public class QuickSeq implements Sequence {
    public static void main(String[] args) {
        int maxN = Integer.parseInt(args[0]);
        QuickSeq q = new QuickSeq(maxN);
        Values.show(q, maxN);
    }
}
```

```java
public class Values {
    public static void show(Sequence f, int maxN) {
        double max = 0;
        for (int N = 0; N < maxN; N++)
            if (f.eval(N) > max) max = f.eval(N);
        for (int N = 0; N < maxN; N++)
            double x = 1.0*N/maxN;
            double y = 1.0*f.eval(N)/max;
            StdDraw.filledCircle(x, y, .002);
        StdDraw.show();
    }
}
```

% java QuickSeq 1000
2. Recurrences

- Computing values
- Telescoping
- Types of recurrences
- Mergesort
- Master Theorem
2. Recurrences

- Computing values
- Telescoping
- Types of recurrences
- Mergesort
- Master Theorem
Telescoping a (linear first-order) recurrence

Linear first-order recurrences *telescope* to a sum.

**Example 1.**

\[ a_n = a_{n-1} + n \quad \text{with} \quad a_0 = 0 \]

Apply equation for \( n-1 \)

\[ = a_{n-2} + (n - 1) + n \]

Do it again

\[ = a_{n-3} + (n - 2) + (n - 1) + n \]

Continue, leaving a sum

\[ = a_0 + \sum_{1 \leq k \leq n} k \]

Evaluate sum

\[ = \frac{(n + 1)n}{2} \]

Check.

\[ \frac{(n + 1)n}{2} = \frac{n(n - 1)}{2} + n \]

Challenge: Need to be able to evaluate the sum.
### Elementary discrete sums

<table>
<thead>
<tr>
<th></th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>geometric series</td>
<td>$\sum_{0 \leq k &lt; n} x^k = \frac{1 - x^n}{1 - x}$</td>
</tr>
<tr>
<td>arithmetic series</td>
<td>$\sum_{0 \leq k &lt; n} k = \frac{n(n - 1)}{2} = \binom{n}{2}$</td>
</tr>
<tr>
<td>binomial (upper)</td>
<td>$\sum_{0 \leq k \leq n} \binom{k}{m} = \binom{n + 1}{m + 1}$</td>
</tr>
<tr>
<td>binomial theorem</td>
<td>$\sum_{0 \leq k \leq n} \binom{n}{k} x^k y^{n-k} = (x + y)^n$</td>
</tr>
<tr>
<td>Harmonic numbers</td>
<td>$\sum_{1 \leq k \leq n} \frac{1}{k} = H_n$</td>
</tr>
<tr>
<td>Vandermonde convolution</td>
<td>$\sum_{0 \leq k \leq n} \binom{n}{k} \binom{m}{t-k} = \binom{n+m}{t}$</td>
</tr>
</tbody>
</table>

see Knuth volume 1 for many more
Telescoping a (linear first-order) recurrence (continued)

When coefficients are not 1, multiply/divide by a summation factor.

Example 2.

\[ a_n = 2a_{n-1} + 2^n \quad \text{with} \quad a_0 = 0 \]

Divide by \( 2^n \)

\[ \frac{a_n}{2^n} = \frac{a_{n-1}}{2^{n-1}} + 1 \]

Telescope to a sum

\[ \frac{a_n}{2^n} = \sum_{1 \leq k \leq n} 1 = n \]

\[ a_n = n2^n \]

Check.

\[ n2^n = 2(n - 1)2^{n-1} + 2^n \]

Challenge: How do we find the summation factor?
Telescoping a (linear first-order) recurrence (continued)

Q. What’s the summation factor for \( a_n = x_n a_{n-1} + \ldots \)?

A. Divide by \( x_n x_{n-1} x_{n-2} \ldots x_1 \)

**Example 3.**

\[
a_n = \left(1 + \frac{1}{n}\right) a_{n-1} + 2 \quad \text{for } n > 0 \text{ with } a_0 = 0
\]

Divide by \( n+1 \)

\[
\frac{a_n}{n+1} = \frac{a_{n-1}}{n} + \frac{2}{n+1}
\]

Telescope

\[
= 2 \sum_{1 \leq k \leq n} \frac{1}{k+1} = 2H_{n+1} - 1
\]

\[
a_n = 2(n+1)(H_{n+1} - 1)
\]

Challenge: Still need to be able to evaluate sums.
In-class exercise 1.

Verify the solution for Example 3.

Check initial values

\[ a_n = \left(1 + \frac{1}{n}\right)a_{n-1} + 2 \quad \text{for } n > 0 \text{ with } a_0 = 0 \]

\[ a_1 = 2a_0 + 2 = 2 \]
\[ a_2 = \frac{3}{2}a_1 + 2 = 5 \]
\[ a_3 = \frac{4}{3}a_2 + 2 = 26/3 \]

\[ a_n = 2(n + 1)(H_{n+1} - 1) \]
\[ a_1 = 4(H_2 - 1) = 2 \]
\[ a_2 = 6(H_3 - 1) = 5 \]
\[ a_3 = 8(H_4 - 1) \]
\[ = 8(1/2 + 1/3 + 1/4) \]
\[ = 26/3 \]

Proof

\[ (1 + \frac{1}{n})2n(H_n - 1) + 2 = 2(n + 1)(H_n - 1) + 2 \]
\[ = 2(n + 1)(H_{n+1} - 1) \]
In-class exercise 2.

Solve this recurrence:

\[ na_n = (n - 2)a_{n-1} + 2 \quad \text{for } n > 1 \text{ with } a_1 = 1 \]

**Hard way:**

Summation factor:

\[ \frac{n - 2}{n} \cdot \frac{n - 3}{n - 1} \cdot \frac{n - 4}{n - 2} \cdots = \frac{1}{n(n - 1)} \]

**Easy way:**

\[ 2a_2 = 2 \quad \text{so} \quad a_2 = 1 \]

\[ \text{Therefore} \quad a_n = 1 \]

WHY?
Recurrences

- Computing values
- **Telescoping**
- Types of recurrences
- Mergesort
- Master Theorem
Recurrences

- Computing values
- Telescoping
- Types of recurrences
- Mergesort
- Master Theorem
## Types of recurrences

<table>
<thead>
<tr>
<th>Type</th>
<th>Order</th>
<th>Linear Formula</th>
<th>Nonlinear Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>first order</td>
<td>linear</td>
<td>( a_n = na_{n-1} - 1 )</td>
<td>( a_n = 1/(1 + a_{n-1}) )</td>
</tr>
<tr>
<td></td>
<td>nonlinear</td>
<td></td>
<td></td>
</tr>
<tr>
<td>second order</td>
<td>linear</td>
<td>( a_n = a_{n-1} + 2a_{n-2} )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>nonlinear</td>
<td>( a_n = a_{n-1}a_{n-2} + \sqrt{a_{n-2}} )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>variable coefficients</td>
<td>( a_n = na_{n-1} + (n - 1)a_{n-2} + 1 )</td>
<td></td>
</tr>
<tr>
<td>higher order</td>
<td>( a_n = f(a_{n-1}, a_{n-2}, \ldots, a_{n-t}) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>full history</td>
<td>( a_n = n + a_{n-1} + a_{n-2} \ldots + a_1 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>divide-and-conquer</td>
<td>( a_n = a_{\lfloor n/2 \rfloor} + a_{\lceil n/2 \rceil} + n )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Nonlinear first-order recurrences

**Example.** (Newton’s method)

\[ c_N = \frac{1}{2} \left( c_{N-1} + \frac{2}{c_{N-1}} \right) \]

[Typical in scientific computing]

```java
public class SqrtTwo implements Sequence {
    private final double[] c;
    public SqrtTwo(int maxN) {
        c = new double[maxN+1];
        c[0] = 1;
        for (int N = 1; N <= maxN; N++)
            c[N] = (c[N-1] + 2/c[N-1])/2;
    }
    public double eval(int N) {
        return c[N];
    }
    public static void main(String[] args) {
        int maxN = Integer.parseInt(args[0]);
        SqrtTwo test = new SqrtTwo(maxN);
        for (int i = 0; i < maxN; i++)
            StdOut.println(test.eval(i));
    }
}
```

quadratic convergence: number of significant digits doubles for each iteration

% java SqrtTwo 10
1.0
1.5
1.4166666666666665
1.4142156862745097
1.4142135623746899
1.414213562373095
1.414213562373095
1.414213562373095
1.414213562373095
1.414213562373095
Higher-order linear recurrences

[ Stay tuned for systematic solution using generating functions (next lecture) ]

### Example 4.

\[ a_n = 5a_{n-1} - 6a_{n-2} \quad \text{for } n \geq 2 \text{ with } a_0 = 0 \text{ and } a_1 = 1 \]

<table>
<thead>
<tr>
<th>Step</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Postulate that ( a_n = x^n )</td>
<td>( x^n = 5x^{n-1} - 6x^{n-2} )</td>
</tr>
<tr>
<td>Divide by ( x^{n-2} )</td>
<td>( x^2 - 5x + 6 = 0 )</td>
</tr>
<tr>
<td>Factor</td>
<td>( (x - 2)(x - 3) = 0 )</td>
</tr>
<tr>
<td>Form of solution must be</td>
<td>( a_n = c_03^n + c_12^n )</td>
</tr>
<tr>
<td>Use initial conditions to solve for coefficients</td>
<td>( a_0 = 0 = c_0 + c_1 )  ( a_1 = 1 = 3c_0 + 2c_1 )</td>
</tr>
<tr>
<td>Solution is ( c_0 = 1 ) and ( c_1 = -1 )</td>
<td>( a_n = 3^n - 2^n )</td>
</tr>
</tbody>
</table>

Note dependence on initial conditions
Higher-order linear recurrences

[ Stay tuned for systematic solution using generating functions (next lecture) ]

Example 5. Fibonacci numbers

\[ a_n = a_{n-1} + a_{n-2} \quad \text{for } n \geq 2 \text{ with } a_0 = 0 \text{ and } a_1 = 1 \]

Postulate that \( a_n = x^n \)

Divide by \( x^{n-2} \)

Factor

Form of solution must be

Use initial conditions to solve for coefficients

Solution

\[ \phi = \frac{1 + \sqrt{5}}{2} \]

\[ \hat{\phi} = \frac{1 - \sqrt{5}}{2} \]

Note dependence on initial conditions
Higher-order linear recurrences (continued)

Procedure amounts to an \textit{algorithm}.

\begin{example}
Fibonacci numbers
\begin{align*}
a_n &= a_{n-1} + a_{n-2} \quad \text{for } n \geq 2 \text{ with } a_0 = 0 \text{ and } a_1 = 1
\end{align*}
\end{example}

Postulate that $a_n = x^n$

Divide by $x^{n-2}$

Factor

Form of solution must be

Use initial conditions to solve for coefficients

Solution

\begin{align*}
\phi &= \frac{1 + \sqrt{5}}{2} \\
\hat{\phi} &= \frac{1 - \sqrt{5}}{2}
\end{align*}

Multiple roots? Add $n \alpha^n$ terms (see text)

Need to compute roots? Use symbolic math package.

\begin{verbatim}
sage: realpoly.<z> = PolynomialRing(CC)
sage: factor(z^2-z-1)
(z - 1.61803398874989) * (z + 0.618033988749895)
\end{verbatim}

Complex roots? Stay tuned for systematic solution using GFs (next lecture)
Divide-and-conquer recurrences

*Divide and conquer* is an effective technique in algorithm design.

Recursive programs map directly to recurrences.

Classic examples:
- Binary search
- Mergesort
- Batcher network
- Karatsuba multiplication
- Strassen matrix multiplication
Recurrences

- Computing values
- Telescoping
- Types of recurrences
- Mergesort
- Master Theorem
Recurrences

- Computing values
- Telescoping
- Types of recurrences
- Mergesort
- Master Theorem
Warmup: binary search

Everyone’s first divide-and-conquer algorithm

```java
// Precondition: array a[] is sorted.
public static int rank(int key, int[] a)
{
    int lo = 0;
    int hi = a.length - 1;
    while (lo <= hi)
    {
        // Key is in a[lo..hi] or not present.
        int mid = lo + (hi - lo) / 2;
        if (key < a[mid]) hi = mid - 1;
        else if (key > a[mid]) lo = mid + 1;
        else return mid;
    }
    return -1;
}
```

Number of compares in the worst case

$$B_N = B_{\lfloor N/2 \rfloor} + 1 \quad \text{for } N > 1 \text{ with } B_1 = 1$$
Analysis of binary search (easy case)

\[ B_N = B_{\lceil N/2 \rceil} + 1 \quad \text{for } N > 1 \text{ with } B_1 = 1 \]

Exact solution for \( N = 2^n \).

\[ a_n \equiv B_{2^n} \]

\[ a_n = a_{n-1} + 1 \quad \text{for } n > 0 \text{ with } a_0 = 1 \]

Telescope to a sum

\[ a_n = \sum_{1 \leq k \leq n} 1 = n \]

\[ B_N = \lg N \quad \text{when } N \text{ is a power of } 2 \]

Check. \( \lg N = \lg(N/2) + 1 \)
Analysis of binary search (general case)

Easy by correspondence with binary numbers

Define $B_N$ to be the number of bits in the binary representation of $N$.
- $B_1 = 1$.
- Removing the rightmost bit of $N$ gives $\lfloor N/2 \rfloor$.

Therefore $B_N = B_{\lfloor N/2 \rfloor} + 1$ for $N > 1$ with $B_1 = 1$

same recurrence as for binary search

**Theorem.** $B_N = \lfloor \lg N \rfloor + 1$

**Proof.** Immediate by definition of $\lfloor \rfloor$.

<table>
<thead>
<tr>
<th>$N$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary</td>
<td>1</td>
<td>10</td>
<td>11</td>
<td>100</td>
<td>101</td>
<td>110</td>
<td>111</td>
<td>1000</td>
<td>1001</td>
</tr>
<tr>
<td>$\lg N$</td>
<td>0</td>
<td>1.0</td>
<td>1.58...</td>
<td>2.0</td>
<td>2.32...</td>
<td>2.58...</td>
<td>2.80...</td>
<td>3</td>
<td>3.16...</td>
</tr>
<tr>
<td>$\lfloor \lg N \rfloor$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$\lfloor \lg N \rfloor + 1$</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Example.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$\lfloor N/2 \rfloor$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1101011</td>
<td>110101</td>
</tr>
<tr>
<td>107</td>
<td>53</td>
</tr>
</tbody>
</table>

$B_N = n + 1$ for $2^n \leq N < 2^{n+1}$

or $n \leq \lg N < n + 1$ $\implies$ $n = \lfloor \lg N \rfloor$
Mergesort

Everyone’s *second* divide-and-conquer algorithm

```java
public class Merge {
    ...
    private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
        if (hi <= lo) return;
        int mid = lo + (hi - lo) / 2;
        sort(a, aux, lo, mid);
        sort(a, aux, mid + 1, hi);
        merge(a, aux, lo, mid, hi);
    }
    ...
}
```

For simplicity, assume merge implementation uses *N* compares

**Number of compares for sort:**

\[
C_N = C_{\lfloor N/2 \rfloor} + C_{\lfloor N/2 \rfloor} + N \quad \text{for } N > 1 \text{ with } C_1 = 1
\]
Analysis of mergesort (easy case)

Number of compares for sort: \( C_N = C_{[N/2]} + C_{[N/2]} + N \) for \( N > 1 \) with \( C_1 = 1 \)

Already solved for \( N = 2^n \)

\[ a_n = 2a_{n-1} + 2^n \quad \text{with} \quad a_0 = 0 \]

Divide by \( 2^n \)

\[ \frac{a_n}{2^n} = \frac{a_{n-1}}{2^{n-1}} + 1 \]

Telescope to a sum

\[ \frac{a_n}{2^n} = \sum_{1 \leq k \leq n} 1 = n \]

\[ a_n = n2^n \]

Solution: \( C_N = N\lg N \) when \( N \) is a power of 2
Analysis of mergesort (general case)

Number of compares for sort: \( C_N = C_{[N/2]} + C_{[N/2]} + N \) for \( N > 1 \) with \( C_1 = 1 \)

Solution: \( C_N = N\lg N \) when \( N \) is a power of 2

Q. For quicksort, the number of compares is \( \sim 2N\ln N - 2(1 - \gamma)N \)

Is the number of compares for mergesort \( \sim N\lg N + \alpha N \) for some constant \( \alpha \)?

A. NO!
public class MergeLinearTerm implements Sequence
{
    private final double[] c;

    public MergeLinear(int maxN)
    {
        c = new double[maxN+1];
        c[0] = 0;
        for (int N = 1; N <= maxN; N++)
            c[N] = N + c[N/2] + c[N-(N/2)];
        for (int N = 1; N <= maxN; N++)
            c[N] -= N*Math.log(N)/Math.log(2) + N;
    }

    public double eval(int N)
    {
        return c[N];
    }

    public static void main(String[] args)
    {
        int maxN = Integer.parseInt(args[0]);
        MergeLinearTerm M = new MergeLinearTerm(maxN);
        Values.show(M, maxN);
    }

    }
Analysis of mergesort (general case)

Number of compares for sort:

\[ C_N = C_{\lfloor N/2 \rfloor} + C_{\lceil N/2 \rceil} + N \quad \text{for } N > 1 \]  
with \( C_1 = 1 \)

Same formula for \( N+1 \).

\[ C_{N+1} = C_{\lfloor (N+1)/2 \rfloor} + C_{\lceil (N+1)/2 \rceil} + N + 1 \]

\[ = C_{\lfloor N/2 \rfloor} + C_{\lfloor N/2 \rfloor + 1} + N + 1 \]

Subtract.

\[ C_{N+1} - C_N = C_{\lfloor N/2 \rfloor + 1} - C_{\lfloor N/2 \rfloor} + 1 \]

Define \( D_N = C_{N+1} - C_N \).

\[ D_N = D_{\lfloor N/2 \rfloor} + 1 \]

Solve as for binary search.

\[ D_N = \lceil \lg N \rceil + 2 \]

Telescope.

\[ C_N = N - 1 + \sum_{1 \leq k < N} (\lfloor \lg k \rfloor + 1) \]

Theorem. \( C_N = N - 1 + \text{number of bits in binary representation of numbers } < N \)
## Combinatorial correspondence

\( S_N = \text{number of bits in the binary rep. of all numbers < } N \)

<table>
<thead>
<tr>
<th>( N / 2 )</th>
<th>( S_{[N/2]} )</th>
<th>( S_{[N/2]} )</th>
<th>( N - 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>101</td>
<td>101</td>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>110</td>
<td>110</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>111</td>
<td>111</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>1001</td>
<td>1001</td>
<td>1001</td>
<td>1001</td>
</tr>
<tr>
<td>1010</td>
<td>1010</td>
<td>1010</td>
<td>1010</td>
</tr>
<tr>
<td>1011</td>
<td>1011</td>
<td>1011</td>
<td>1011</td>
</tr>
<tr>
<td>1100</td>
<td>1100</td>
<td>1100</td>
<td>1100</td>
</tr>
<tr>
<td>1101</td>
<td>1101</td>
<td>1101</td>
<td>1101</td>
</tr>
<tr>
<td>1110</td>
<td>1110</td>
<td>1110</td>
<td>1110</td>
</tr>
</tbody>
</table>

\[ S_N = S_{[N/2]} + S_{[N/2]} + N - 1 \]

Same recurrence as mergesort (except for \(-1\)): \( C_N = S_N + N - 1 \)
Number of bits in all numbers < N (alternate view)

\[ S_N = N(\lfloor \lg N \rfloor + 1) - \sum_{0 \leq k \leq \lfloor \lg N \rfloor} 2^k \]
\[ = N \lfloor \lg N \rfloor - 2^{\lfloor \lg N \rfloor + 1} + N + 1 \]

\[ C_N = S_N + N - 1 \]
\[ = N \lfloor \lg N \rfloor - 2^{\lfloor \lg N \rfloor + 1} + 2N \]

**Theorem.** Number of compares for mergesort is \( N \lfloor \lg N \rfloor - 2^{\lfloor \lg N \rfloor + 1} + 2N \)
Analysis of mergesort (summary)

Number of compares for sort: \[ C_N = C_{\lfloor N/2 \rfloor} + C_{\lfloor N/2 \rfloor} + N \quad \text{for } N > 1 \quad \text{with} \quad C_1 = 1 \]

Solution: \[ C_N = N \lfloor \log_2 N \rfloor \quad \text{when } N \text{ is a power of 2} \]

Theorem. Number of compares for mergesort is \[ N \lfloor \log_2 N \rfloor - 2^{\lfloor \log_2 N \rfloor + 1} + 2N \]

Alternate formulation (Knuth). \[ C_N = N \log_2 N + N \alpha(N) \]

Notation: \( \lfloor \log_2 N \rfloor = \log_2 N - \{ \log_2 N \} \)

\[
\begin{align*}
1 - \{ \log_2 N \} \\
+ \\
1 - 2^{1 - \{ \log_2 N \}} \\
= \\
2 - \{ \log_2 N \} - 2^{1 - \{ \log_2 N \}}
\end{align*}
\]

\( \alpha(N) \)

\( N\alpha(N) \)
Recurrences

- Computing values
- Telescoping
- Types of recurrences
- Mergesort
- Master Theorem
Recurrences

- Computing values
- Telescoping
- Types of recurrences
- Mergesort
- Master Theorem
Divide-and-conquer algorithms

Suppose that an algorithm attacks a problem of size \( N \) by

- Dividing into \( \alpha \) parts of size about \( N/\beta \).
- Solving recursively.
- Combining solutions with extra cost \( \Theta(N^\gamma (\log N)^\delta) \)

**Example 1** (mergesort): \( \alpha = 2, \beta = 2, \gamma = 1, \delta = 0 \)

\[
C_N = 2C_{N/2} + N
\]

**Example 2** (Batcher network): \( \alpha = 2, \beta = 2, \gamma = 1, \delta = 1 \)

\[
C_N = 2C_{N/2} + N\lg N
\]

**Example 3** (Karatsuba multiplication): \( \alpha = 3, \beta = 2, \gamma = 1, \delta = 0 \)

\[
C_N = 3C_{N/2} + N
\]

**Example 4** (Strassen matrix multiply): \( \alpha = 7, \beta = 2, \gamma = 1, \delta = 0 \)

\[
C_N = 7C_{N/2} + N
\]

only valid when \( N \) is a power of 2
"Master Theorem" for divide-and-conquer algorithms

Suppose that an algorithm attacks a problem of size $n$ by dividing into $\alpha$ parts of size about $n/\beta$ with extra cost $\Theta(n^\gamma (\log n)^\delta)$

**Theorem.** The solution to the recurrence

$$a_n = a_{n/\beta} + O(1) + a_{n/\beta} + O(1) + \ldots + a_{n/\beta} + O(1) + \Theta(n^{\gamma} (\log n)^{\delta})$$

is given by

- $a_n = \Theta(n^{\gamma} (\log n)^{\delta})$ when $\gamma < \log_\beta \alpha$
- $a_n = \Theta(n^{\gamma} (\log n)^{\delta+1})$ when $\gamma = \log_\beta \alpha$
- $a_n = \Theta(n^{\log_\beta \alpha})$ when $\gamma > \log_\beta \alpha$

**Example:** $\alpha = 3$

- $\beta = 2$
- $\beta = 3$
- $\beta = 4$
### Typical “Master Theorem” applications

Suppose that an algorithm attacks a problem of size $N$ by

- Dividing into $\alpha$ parts of size about $N/\beta$.
- Solving recursively.
- Combining solutions with extra cost $\Theta(N^\gamma \log N^\delta)$

#### Master Theorem

<table>
<thead>
<tr>
<th>Case</th>
<th>Equation</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_n = \Theta(n^\gamma \log n)^\delta$</td>
<td>when $\gamma &lt; \log \beta \alpha$</td>
<td></td>
</tr>
<tr>
<td>$a_n = \Theta(n^\gamma \log n)^{\delta+1}$</td>
<td>when $\gamma = \log \beta \alpha$</td>
<td></td>
</tr>
<tr>
<td>$a_n = \Theta(n^{\log \beta \alpha})$</td>
<td>when $\gamma &gt; \log \beta \alpha$</td>
<td></td>
</tr>
</tbody>
</table>

#### Asymptotic growth rate

<table>
<thead>
<tr>
<th>Example</th>
<th>$\alpha$, $\beta$, $\gamma$, $\delta$</th>
<th>Growth Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 1 (mergesort):</td>
<td>$\alpha = 2$, $\beta = 2$, $\gamma = 1$, $\delta = 0$</td>
<td>$\Theta(N \log N)$</td>
</tr>
<tr>
<td>Example 2 (Batcher network):</td>
<td>$\alpha = 2$, $\beta = 2$, $\gamma = 1$, $\delta = 1$</td>
<td>$\Theta(N \log_2 N^2) = \Theta(N^2.807...)$</td>
</tr>
<tr>
<td>Example 3 (Karatsuba multiplication):</td>
<td>$\alpha = 3$, $\beta = 2$, $\gamma = 1$, $\delta = 0$</td>
<td>$\Theta(N^{\log_2 3}) = \Theta(N^{1.585...})$</td>
</tr>
<tr>
<td>Example 4 (Strassen matrix multiply):</td>
<td>$\alpha = 7$, $\beta = 2$, $\gamma = 1$, $\delta = 0$</td>
<td>$\Theta(N^{\log_2 7}) = \Theta(N^{2.807...})$</td>
</tr>
</tbody>
</table>
Versions of the “Master Theorem”

Suppose that an algorithm attacks a problem of size $N$ by

- Dividing into $\alpha$ parts of size about $N/\beta$.
- Solving recursively.
- Combining solutions with extra cost $\Theta(N^{\gamma}(\log N)^{\delta})$

1. **Precise** results are available for certain applications in the analysis of algorithms.

2. **General** results are available for proofs in the theory of algorithms.

3. **A full solution** using analytic combinatorics was derived in 2011 by Szpankowski and Drmota.

see “A Master Theorem for Divide-and-Conquer Recurrences” by Szpankowski and Drmota (SODA 2011).
Recurrences

- Computing values
- Telescoping
- Types of recurrences
- Mergesort
- Master Theorem
Exercise 2.17

Percentage of three nodes at the bottom level of a 2-3 tree?

Exercise 2.17 [Yao] (“Fringe analysis of 2–3 trees”) Solve the recurrence

\[ A_N = A_{N-1} - \frac{2A_{N-1}}{N} + 2\left(1 - \frac{2A_{N-1}}{N}\right) \quad \text{for } N > 0 \text{ with } A_0 = 0. \]

This recurrence describes the following random process: A set of \( N \) elements collect into “2-nodes” and “3-nodes.” At each step each 2-node is likely to turn into a 3-node with probability \( 2/N \) and each 3-node is likely to turn into two 2-nodes with probability \( 3/N \). What is the average number of 2-nodes after \( N \) steps?
Exercise 2.69

Details of divide-by-three and conquer?

Exercise 2.69 Plot the periodic part of the solution to the recurrence

\[ a_N = 3a_{\lfloor N/3 \rfloor} + N \quad \text{for } N > 3 \text{ with } a_1 = a_2 = a_3 = 1 \]

for \( 1 \leq N \leq 972. \)
Assignments for next lecture

1. Read pages 41-86 in text.

2. Write up solution to Ex. 2.17.

3. Set up StdDraw from Algs book site

4. Do Exercise 2.69.
2. Recurrences