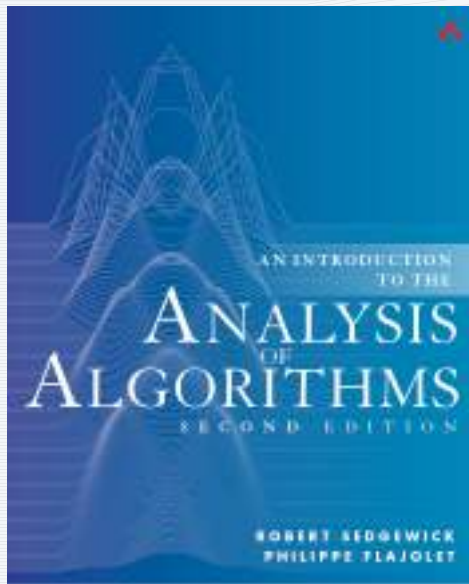


ANALYTIC COMBINATORICS PART ONE

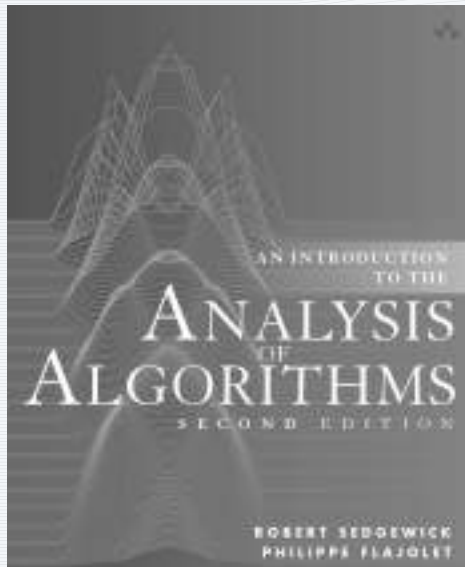


<http://aofa.cs.princeton.edu>

2. Recurrences

ANALYTIC COMBINATORICS

PART ONE



<http://aofa.cs.princeton.edu>

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, ...

MUST specify
for all N with
initial conditions



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 $\frac{2}{3}$, 12 $\frac{5}{6}$, 17 $\frac{2}{5}$, ...

program

```
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.

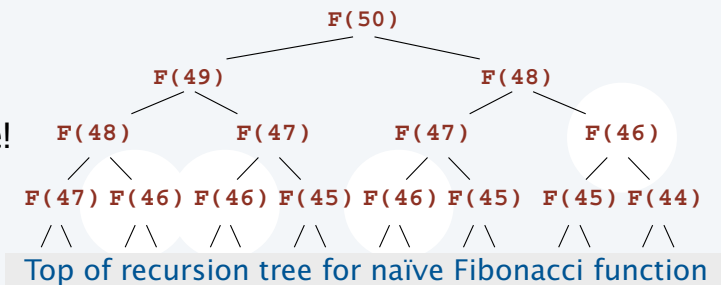
$$F_N = F_{N-1} + F_{N-2} \text{ for } N \geq 2 \text{ with } F_0 = 0 \text{ and } F_1 = 1$$

Use a recursive program?

```
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.

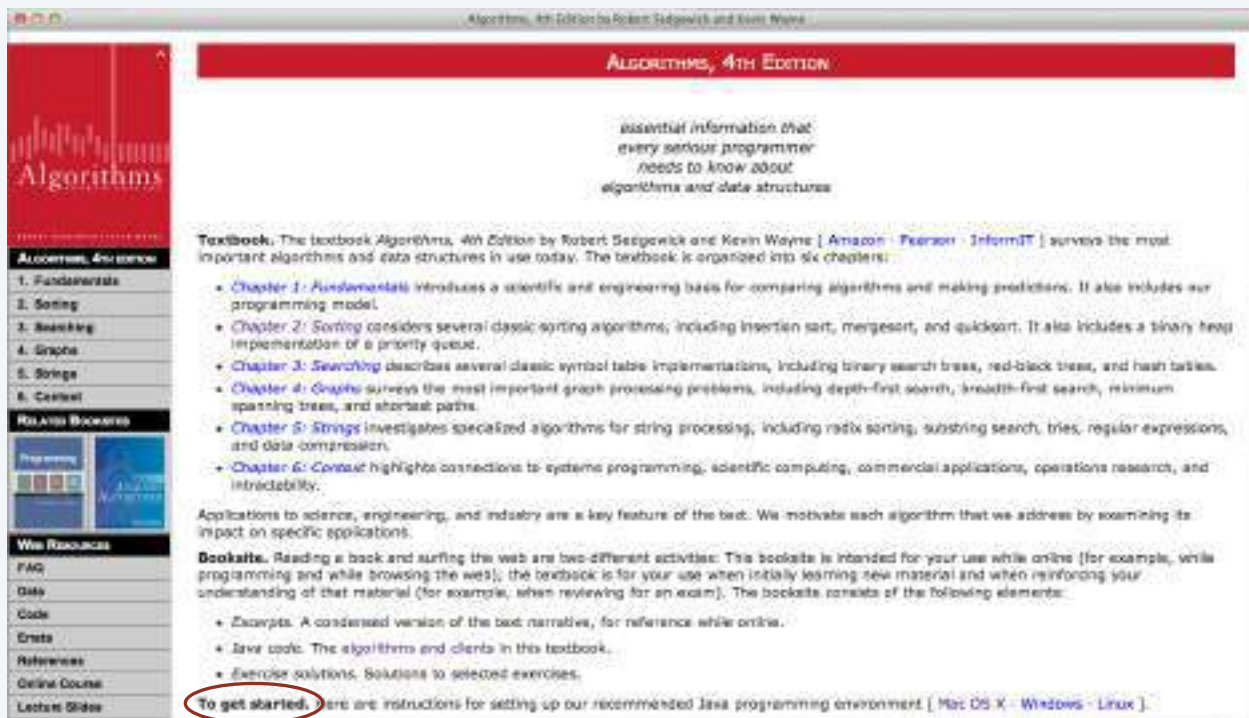
```
long[] F = new long[51];
F[0] = 0; F[1] = 1;
if (N == 1) return 1;
for (int i = 2; i <= 50; i++)
    F[i] = F[i-1] + F[i-2];
```



Common-sense starting point for solving any recurrence

Use your computer to compute initial values.

First step: Download "standard model" from *Algorithms, 4th edition* booksite.



<http://algs4.cs.princeton.edu>

StdIn *Standard Input*

StdOut *Standard Output*

StdDraw *Standard Drawings*

StdRandom *Random Numbers*

... (Several other libraries)

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 $F_1 = 1$

```
Fib.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));
    }
}
```

Compute all values
in the constructor

Sequence.java

```
public interface Sequence
{
    public double eval(int N);
}
```

```
% 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

```
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.

Values.java

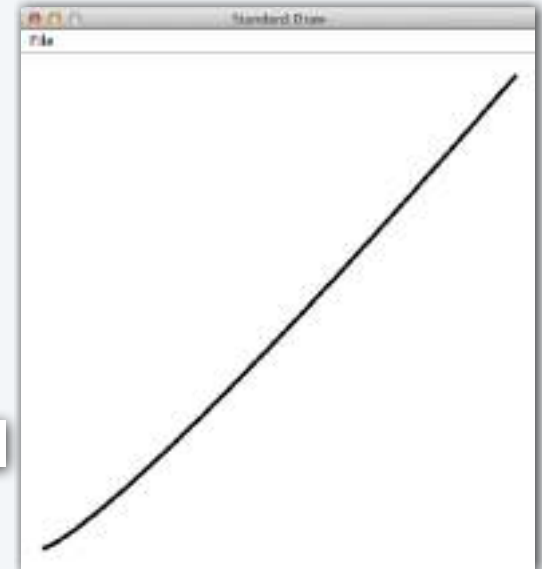
```
public class Values
{
    public static void show(Sequence f, int maxN)
    {
        double max = 0;
        for (int N = 0; N < maxN; N++)
            if (f.eval(N) > max) max = f.eval(N);
        for (int N = 0; N < maxN; N++)
        {
            double x = 1.0*N/maxN;
            double y =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

QuickSeq.java

```
public class QuickSeq implements Sequence
{
    // Implementation as above.

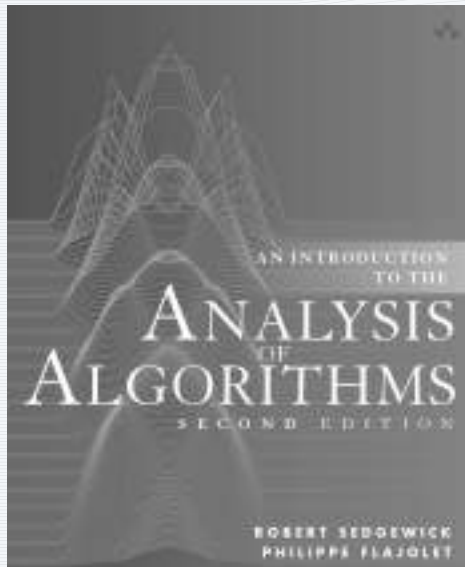
    public static void main(String[] args)
    {
        int maxN = Integer.parseInt(args[0]);
        QuickSeq q = new QuickSeq(maxN);
        Values.show(q, maxN);
    }
}
```

```
% java QuickSeq 1000
```



ANALYTIC COMBINATORICS

PART ONE



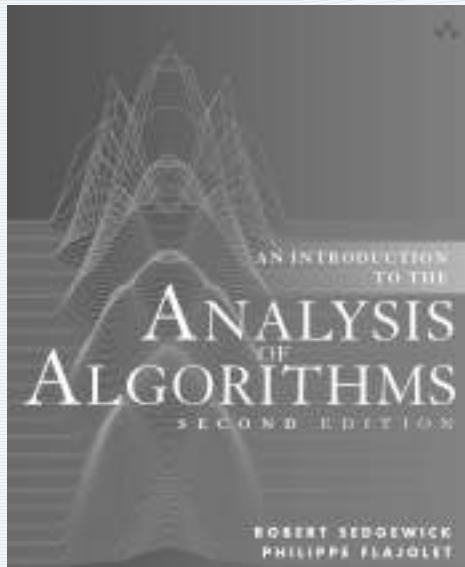
<http://aofa.cs.princeton.edu>

2. Recurrences

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

ANALYTIC COMBINATORICS

PART ONE



<http://aofa.cs.princeton.edu>

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.

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

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

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

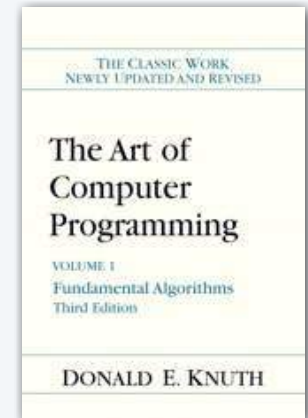
Check.

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

Challenge: Need to be able to evaluate the sum.

Elementary discrete sums

geometric series	$\sum_{0 \leq k < n} x^k = \frac{1 - x^n}{1 - x}$
arithmetic series	$\sum_{0 \leq k < n} k = \frac{n(n-1)}{2} = \binom{n}{2}$
binomial (upper)	$\sum_{0 \leq k \leq n} \binom{k}{m} = \binom{n+1}{m+1}$
binomial theorem	$\sum_{0 \leq k \leq n} \binom{n}{k} x^k y^{n-k} = (x + y)^n$
Harmonic numbers	$\sum_{1 \leq k \leq n} \frac{1}{k} = H_n$
Vandermonde convolution	$\sum_{0 \leq k \leq n} \binom{n}{k} \binom{m}{t-k} = \binom{n+m}{t}$



see Knuth volume I
for many more

Telescoping a (linear first-order) recurrence (continued)

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

Example 2.

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

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 $a_n = x_n a_{n-1} + \dots$?

A. Divide by $x_n x_{n-1} x_{n-2} \dots x_1$

Example 3.

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

summation factor:

$$\frac{n+1}{n} \frac{n}{n-1} \frac{n-1}{n-2} \dots \frac{3}{2} \frac{2}{1} = n+1$$

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

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

$$a_1 = 2a_0 + 2 = 2$$

$$a_2 = \frac{3}{2}a_1 + 2 = 5$$

$$a_3 = \frac{4}{3}a_2 + 2 = 26/3$$

$$a_n = 2(n+1)(H_{n+1} - 1)$$

$$a_1 = 4(H_2 - 1) = 2$$

$$a_2 = 6(H_3 - 1) = 5$$

$$\begin{aligned} a_3 &= 8(H_4 - 1) \\ &= 8(1/2 + 1/3 + 1/4) \\ &= 26/3 \end{aligned}$$

Proof

$$\begin{aligned} \left(1 + \frac{1}{n}\right) \overbrace{2n(H_n - 1)}^{a_{n-1}} + 2 &= 2(n+1)(H_n - 1) + 2 \\ &= 2(n+1) \underbrace{(H_{n+1} - 1)}_{a_n} \end{aligned}$$

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: $\frac{n-2}{n} \frac{n-3}{n-1} \frac{n-4}{n-2} \cdots = \frac{1}{n(n-1)}$

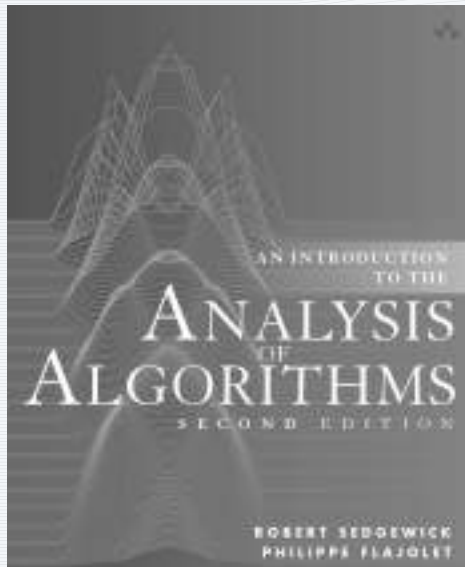
Easy way: $2a_2 = 2$ so $a_2 = 1$

therefore $a_n = 1$

↑
WHY?

ANALYTIC COMBINATORICS

PART ONE



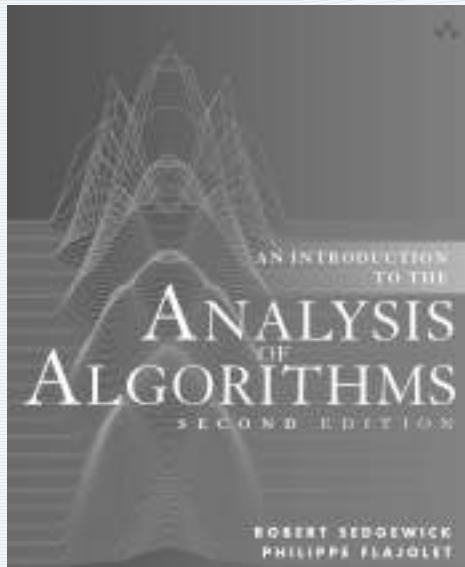
<http://aofa.cs.princeton.edu>

Recurrences

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

ANALYTIC COMBINATORICS

PART ONE



<http://aofa.cs.princeton.edu>

Recurrences

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

Types of recurrences

first order	<i>linear</i>	$a_n = na_{n-1} - 1$
	<i>nonlinear</i>	$a_n = 1/(1 + a_{n-1})$
second order	<i>linear</i>	$a_n = a_{n-1} + 2a_{n-2}$
	<i>nonlinear</i>	$a_n = a_{n-1}a_{n-2} + \sqrt{a_{n-2}}$
	<i>variable coefficients</i>	$a_n = na_{n-1} + (n-1)a_{n-2} + 1$
higher order		$a_n = f(a_{n-1}, a_{n-2}, \dots, a_{n-t})$
full history		$a_n = n + a_{n-1} + a_{n-2} \dots + a_1$
divide-and-conquer		$a_n = a_{\lfloor n/2 \rfloor} + a_{\lceil n/2 \rceil} + n$

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]

SqrtTwo.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.

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

Postulate that $a_n = x^n$

$$x^n = 5x^{n-1} - 6x^{n-2}$$

Divide by x^{n-2}

$$x^2 - 5x + 6 = 0$$

Factor

$$(x - 2)(x - 3) = 0$$

Form of solution must be

$$a_n = c_0 3^n + c_1 2^n$$

Use initial conditions to
solve for coefficients

$$a_0 = 0 = c_0 + c_1$$

$$a_1 = 1 = 3c_0 + 2c_1$$

Note dependence
on initial conditions

Solution is $c_0 = 1$ and $c_1 = -1$

$$a_n = 3^n - 2^n$$

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 $a_n = x^n$

$$x^n = x^{n-1} + x^{n-2}$$

Divide by x^{n-2}

$$x^2 - x - 1 = 0$$

Factor

$$(x - \phi)(x - \hat{\phi}) = 0$$

Form of solution must be

$$a_n = c_0 \phi^n + c_1 \hat{\phi}^n$$

Use initial conditions to
solve for coefficients

$$a_0 = 0 = c_0 + c_1$$

$$a_1 = 1 = \phi c_0 + \hat{\phi} c_1$$

Solution

$$a_n = \frac{\phi^n}{\sqrt{5}} - \frac{\hat{\phi}^n}{\sqrt{5}}$$

$$\phi = \frac{1 + \sqrt{5}}{2}$$
$$\hat{\phi} = \frac{1 - \sqrt{5}}{2}$$

Note dependence
on initial conditions

Higher-order linear recurrences (continued)

Procedure amounts to an *algorithm*.

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

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 $a_n = x^n$:

$$x^n = x^{n-1} + x^{n-2}$$

Divide by x^{n-2} :

$$x^2 - x - 1 = 0$$

Factor

$$(x - \phi)(x - \bar{\phi}) = 0$$

Form of solution must be

$$a_n = c_0 \phi^n + c_1 \bar{\phi}^n$$

Use initial conditions to solve for coefficients:

$$a_0 = 0 = c_0 + c_1$$

$$a_1 = 1 = \phi c_0 + \bar{\phi} c_1$$

Solution

$$a_n = \frac{\phi^n}{\sqrt{5}} - \frac{\bar{\phi}^n}{\sqrt{5}}$$

$$\phi = \frac{1 + \sqrt{5}}{2}$$
$$\bar{\phi} = \frac{1 - \sqrt{5}}{2}$$

Note dependence on initial conditions

Need to compute roots? Use symbolic math package.

```
sage: realpoly.<z> = PolynomialRing(CC)
sage: factor(z^2-z-1)
(z - 1.61803398874989) * (z + 0.618033988749895)
```

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.

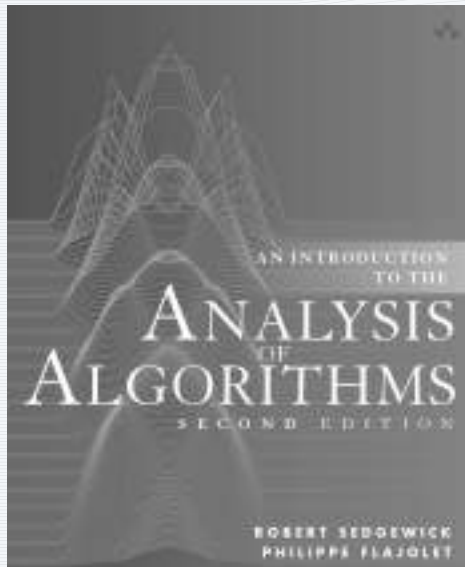
Recursive programs map directly to recurrences.

Classic examples:

- Binary search
- Mergesort
- Batcher network
- Karatsuba multiplication
- Strassen matrix multiplication

ANALYTIC COMBINATORICS

PART ONE



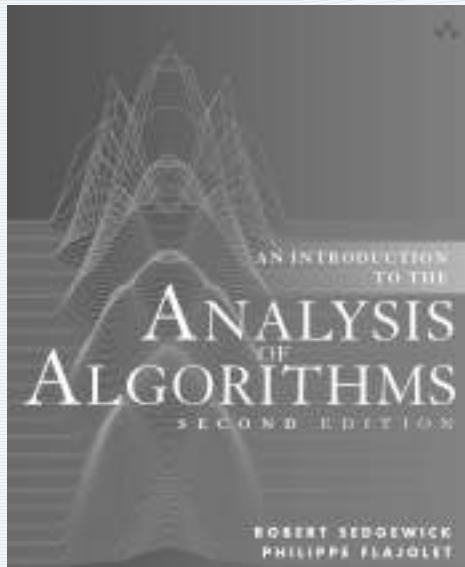
<http://aofa.cs.princeton.edu>

Recurrences

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

ANALYTIC COMBINATORICS

PART ONE



<http://aofa.cs.princeton.edu>

Recurrences

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

Warmup: binary search

Everyone's first divide-and-conquer algorithm

```
// 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)

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

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

Example.

1101011	110101	1
107	53	
N	$\lfloor N/2 \rfloor$	

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

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

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

$$\text{or } n \leq \lg N < n + 1 \implies n = \lfloor \lg N \rfloor$$

N	1	2	3	4	5	6	7	8	9
binary	1	10	11	100	101	110	111	1000	1001
$\lg N$	0	1.0	1.58...	2.0	2.32...	2.58...	2.80...	3	3.16...
$\lfloor \lg N \rfloor$	0	1	1	2	2	2	2	3	3
$\lfloor \lg N \rfloor + 1$	1	2	2	3	3	3	3	4	4

Mergesort

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

```
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: $C_N = C_{\lfloor N/2 \rfloor} + C_{\lceil N/2 \rceil} + N$ for $N > 1$ with $C_1 = 1$

Analysis of mergesort (easy case)

Number of compares for sort: $C_N = C_{\lfloor N/2 \rfloor} + C_{\lceil N/2 \rceil} + N$ for $N > 1$ with $C_1 = 1$

Already solved for $N = 2^n$

Example 2.

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

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: $C_N = C_{\lfloor N/2 \rfloor} + C_{\lceil N/2 \rceil} + N$ 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 α ?

A. NO !

Coefficient of the linear term for mergesort

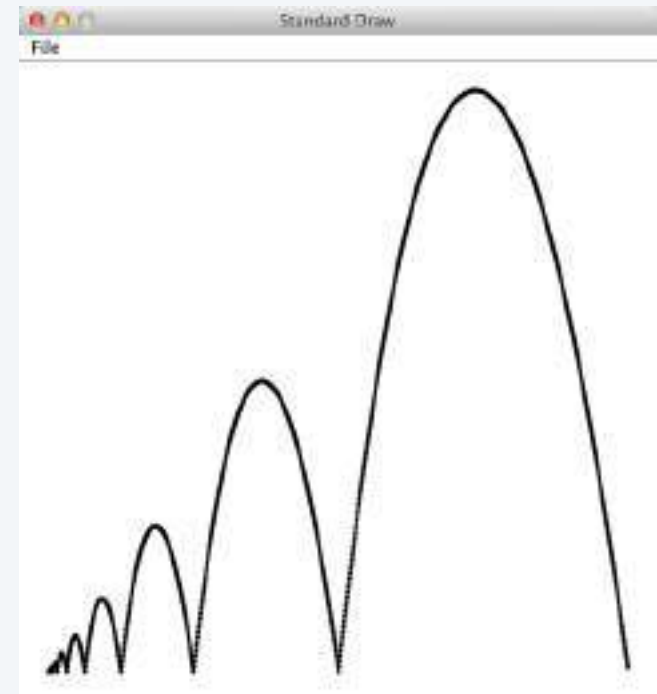
```
public class MergeLinearTerm implements Sequence
{
    private final double[] c;

    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);
    }
}
```

```
% java MergeLinearTerm 512
```



Analysis of mergesort (general case)

Number of compares for sort:

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

Same formula for $N+1$.

$$\begin{aligned} C_{N+1} &= C_{\lfloor (N+1)/2 \rfloor} + C_{\lceil (N+1)/2 \rceil} + N + 1 \\ &= C_{\lfloor N/2 \rfloor} + C_{\lfloor N/2 \rfloor + 1} + N + 1 \end{aligned}$$

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.

$$D_N = \lfloor \lg N \rfloor + 2$$

different
initial
value

Telescope.

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

$\lceil N/2 \rceil = \lfloor (N+1)/2 \rfloor$

1	0	1	1	1
2	1	1	1	2
3	1	2	2	2
4	2	2	2	3
5	2	3	3	3
6	3	3	3	4
7	3	4	4	4
8	4	4	4	5
9	4	5	5	5

$\lfloor N/2 \rfloor + 1 = \lceil (N+1)/2 \rceil$

Theorem. $C_N = N - 1 + \text{number of bits in binary representation of numbers } < N$

Combinatorial correspondence

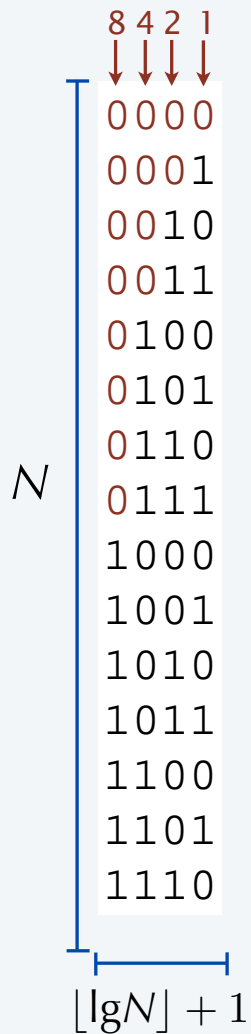
S_N = number of bits in the binary rep. of all numbers $< N$

		$S_{\lfloor N/2 \rfloor}$		$S_{\lceil N/2 \rceil}$		$N - 1$
1		1		1		1
10		10		10		10
11		11		11		11
100		100		100		100
101		101		101		101
110		110		110		110
111		111		111		111
1000	=	1000	+	1000	+	1000
1001		1001		1001		1001
1010		1010		1010		1010
1011		1011		1011		1011
1100		1100		1100		1100
1101		1101		1101		1101
1110		1110		1110		1110

$$S_N = S_{\lfloor N/2 \rfloor} + S_{\lceil N/2 \rceil} + N - 1$$

Same recurrence as mergesort (except for -1): $C_N = S_N + N - 1$

Number of bits in all numbers < N (alternate view)



bits are in an
N by $\lfloor \lg N \rfloor + 1$ box

subtract off red 0s
column by column

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

$$= N\lfloor \lg N \rfloor - 2^{\lfloor \lg N \rfloor + 1} + N + 1$$

$$C_N = S_N + N - 1$$

$$= N\lfloor \lg N \rfloor - 2^{\lfloor \lg N \rfloor + 1} + 2N$$

Theorem. Number of compares for mergesort is $N\lfloor \lg N \rfloor - 2^{\lfloor \lg N \rfloor + 1} + 2N$

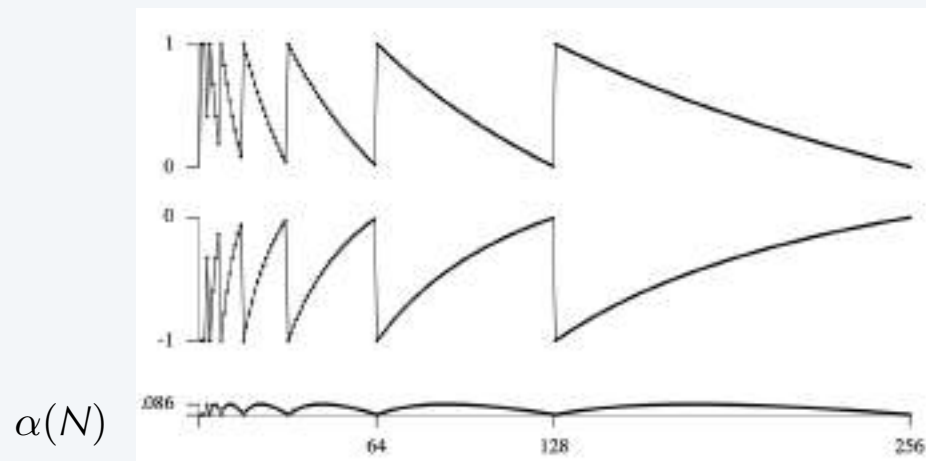
Analysis of mergesort (summary)

Number of compares for sort: $C_N = C_{\lfloor N/2 \rfloor} + C_{\lceil N/2 \rceil} + N$ for $N > 1$ with $C_1 = 1$

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

Theorem. Number of compares for mergesort is $N \lfloor \lg N \rfloor - 2^{\lfloor \lg N \rfloor + 1} + 2N$

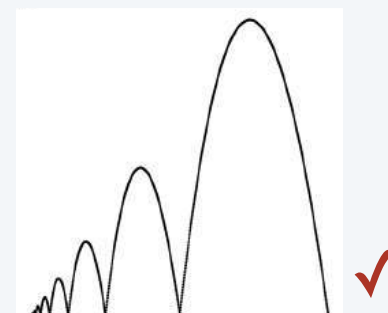
Alternate formulation (Knuth). $C_N = N \lg N + N \alpha(N)$



Notation: $\lfloor \lg N \rfloor = \lg N - \{\lg N\}$

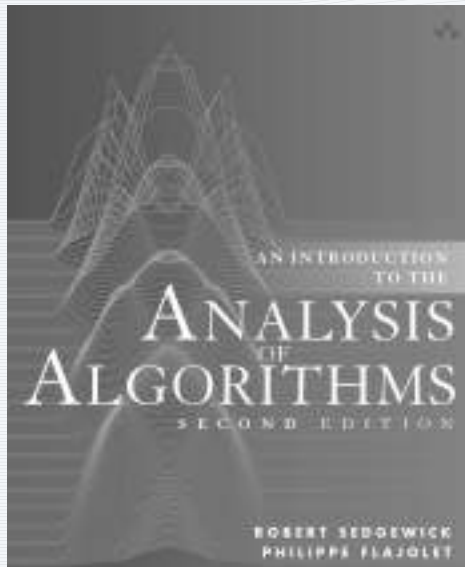
$$\begin{aligned}
 &1 - \{\lg N\} \\
 &+ \\
 &1 - 2^{1 - \{\lg N\}} \\
 &= \\
 &2 - \{\lg N\} - 2^{1 - \{\lg N\}}
 \end{aligned}$$

$N \alpha(N)$



ANALYTIC COMBINATORICS

PART ONE



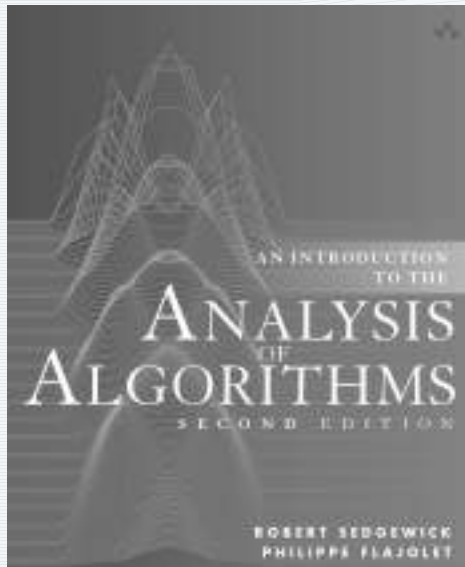
<http://aofa.cs.princeton.edu>

Recurrences

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

ANALYTIC COMBINATORICS

PART ONE



<http://aofa.cs.princeton.edu>

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

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

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

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

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

only valid when
 N is a power of 2



$$C_N = 2C_{N/2} + N$$

$$C_N = 2C_{N/2} + N \lg N$$

$$C_N = 3C_{N/2} + N$$

$$C_N = 7C_{N/2} + N$$

“Master Theorem” for divide-and-conquer algorithms

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

Theorem. The solution to the recurrence

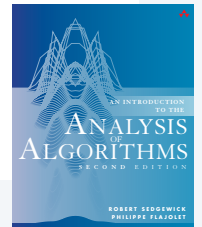
$$a_n = \underbrace{a_{n/\beta+O(1)} + a_{n/\beta+O(1)} + \dots + a_{n/\beta+O(1)}}_{\alpha \text{ terms}} + \Theta(n^\gamma(\log n)^\delta)$$

is given by

$$a_n = \Theta(n^\gamma(\log n)^\delta) \quad \text{when } \gamma < \log_\beta \alpha$$

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

$$a_n = \Theta(n^{\log_\beta \alpha}) \quad \text{when } \gamma > \log_\beta \alpha$$

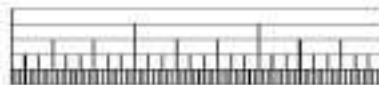


Example: $\alpha = 3$

$\beta = 2$



$\beta = 3$



$\beta = 4$



Typical “Master Theorem” applications

Suppose that an algorithm attacks a problem of size N by

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

Master Theorem

$$\begin{array}{ll} a_n = \Theta(n^\gamma (\log n)^\delta) & \text{when } \gamma < \log_\beta \alpha \\ a_n = \Theta(n^\gamma (\log n)^{\delta+1}) & \text{when } \gamma = \log_\beta \alpha \\ a_n = \Theta(n^{\log_\beta \alpha}) & \text{when } \gamma > \log_\beta \alpha \end{array}$$

Asymptotic growth rate



$$\Theta(N \log N)$$

$$\Theta(N(\log N)^2)$$

$$\Theta(N^{\log_2 3}) = \Theta(N^{1.585\dots})$$

$$\Theta(N^{\log_2 7}) = \Theta(N^{2.807\dots})$$

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

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

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

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

Versions of the “Master Theorem”

Suppose that an algorithm attacks a problem of size N by

- Dividing into α parts of size about N/β .
- 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.

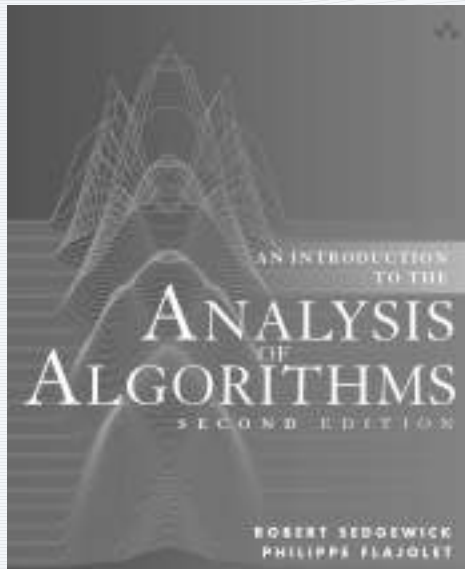


3. **A full solution** using analytic combinatorics was derived in 2011 by Szpankowski and Drmota.

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

ANALYTIC COMBINATORICS

PART ONE



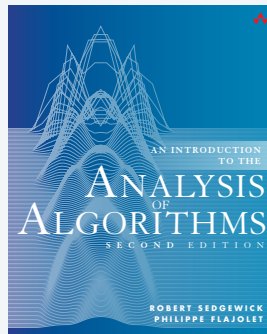
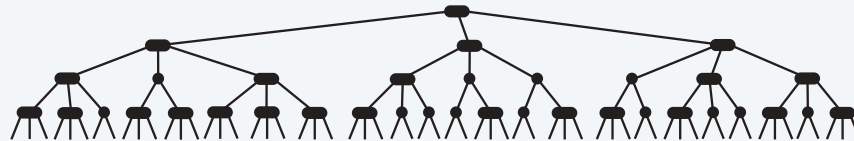
<http://aofa.cs.princeton.edu>

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?



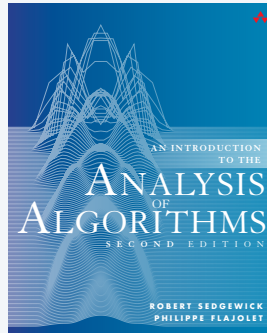
Exercise 2.17 [Yao] (“Fringe analysis of 2–3 trees”) Solve the recurrence

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

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 $1 \leq N \leq 972$.

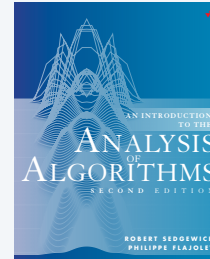
Assignments for next lecture

1. Read pages 41-86 in text.

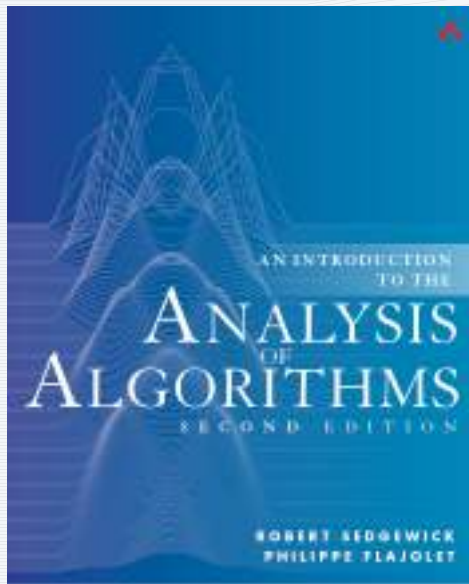
2. Write up solution to Ex. 2.17.

3. Set up StdDraw from *Algs* booksite

4. Do Exercise 2.69.



ANALYTIC COMBINATORICS PART ONE



<http://aofa.cs.princeton.edu>

2. Recurrences