→ **First, naive method:**

```
bool IsPrime (int N)      /* N ≥ 2 */
{ /* DIVIDE N BY ALL INTS UP TO N */
  int noDivisors, i;
  noDivisors = 0;
  for (i = 1; i <= N; i++)
  {
    if (N % i == 0)
       noDivisors += 1;
  }
  return (noDivisors == 2);
}
```

- UNAMBIGUOUS
- EFFECTIVE
- TERMINATES

→ **Improve EFFICIENCY:**

1) STOP when divisor $\neq 1, N$ found
2) 2 is PRIME, but all other even numbers are not
   ➡ check only odd numbers
3) for loop can STOP @ $i \leq \sqrt{N}$ /* WHY? */
   (→ P. 190)

```
bool IsPrime (int N)      /* N >= 2 */
{ int i, limit;
  if (N == 2) return (TRUE);
  if (N % [redacted] 2 == 0) return (FALSE);
  limit = (int) sqrt(N) + 1;     /* N=16 => sqrt "might" return 3.9999 */
  for (i = 3; i <= limit; i += 2)
  {
   if (N % i == 0) return (FALSE);
  }
  return (TRUE);
}
```

FIG. 6-3

EX.: GREATEST COMMON DIVISOR (GCD)

GCD (10, 15) ➡ 5
GCD (12, 24) ➡ 12
GCD (49, 35) ➡ 7

→ **Prototype:** $\boxed{\text{int GCD (int x, int y);}}$

(i) CLEAR 'BRUTE-FORCE':

INP: $x=10$, $y=15$ ($x<y$)

10? $10 < 15$

$10 \% 10 = 0$, $15 \% 10 = 5$

9? $10 \% 9 = 1$, $15 \% 9 = 6$

8? $10 \% 8 = 2$, $15 \% 8 = 7$

...

5? $10 \% 5 = 0$, $15 \% 5 = 0$ ✓

→ $GCD(x,y) = 5$

```
int GCD (...)
{ int gcd;
  gcd = x;   /* x<y */

  while ( x%gcd != 0 || y%gcd != 0 )
  {
    gcd--;
  }
  return (gcd);
}
```

→ **EUCLID'S ALGORITHM:** MORE EFFICIENT

Theorem: "GCD of x and y is also the GCD of y and x%y."

→ Algorithm Design ...

EX. $x=25$, $y=15$

	i=0	i=1	i=2
x	25	15	10
y	15	10	5
R	10	5	0 ✓

(R "remainder")

done

GCD is 5.

$x=36$, $y=28$

	i=0	i=1	i=2
x	36	28	8
y	28	8	4
R	8	4	0 ✓

done

GCD is 4.

$x=15$, $y=25$

x	15	25	15	10
y	25	15	10	5
R	15	10	5	0 ✓

DOES NOT MATTER WHETHER: $x<y$ OR $x>y$

EUCLID'S ALGORITHM

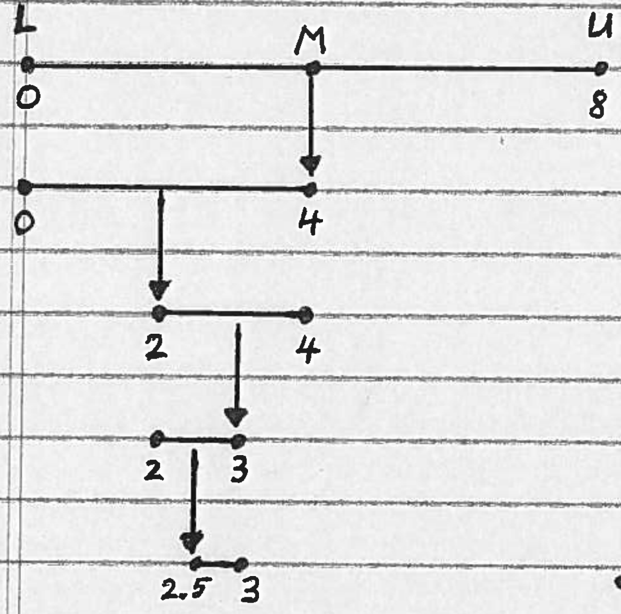
(1) Compute remainder $r = x \% y$.
(2) If $r=0$ then $GCD = y$,
else: $x=y$, $y=r$, repeat...

```
int GCD (int x, int y)
{ int r;
  while (TRUE)
  { r = x % y;
    if (r == 0) break;
    x = y;
  } y = r;
} return (y);
```

Numerical Algorithms

Ex: Square-root approximation: $\sqrt{8} = ? \rightarrow 0 < \sqrt{8} < 8$

"REPEATED BISECTION"



L, L^2	U, U^2	M, M^2
0, 0	8, 64	4, 16
0, 0	4, 16	2, 4
2, 4	4, 16	3, 9
2, 4	3, 9	2.5, 6.25

- M converges to $\sqrt{8}$.
- STOP when $|L^2 - U^2| < \varepsilon$.

```
double sqrt (double x)   /* x ≥ 1 */
{ double l, u, m;   /* lower bound, upper bound, middle */
  l = 0;
  u = x;
  m = (l+u)/2;
  while "|m^2 - x| > ε"          /* REPEATED BISECTION */
  { if (m*m <= x)
     { l = m;
     }
    else
     { u = m;
     }
    m = (l+u)/2;
  }
  return (m);
}
```

/* Convergence Speed? */

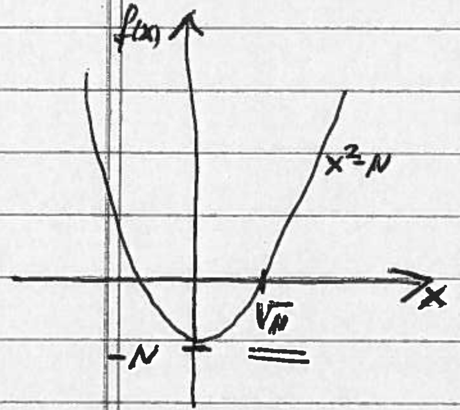
/* Intervals 50% smaller per step */

NEWTON'S METHOD: Increased efficiency!

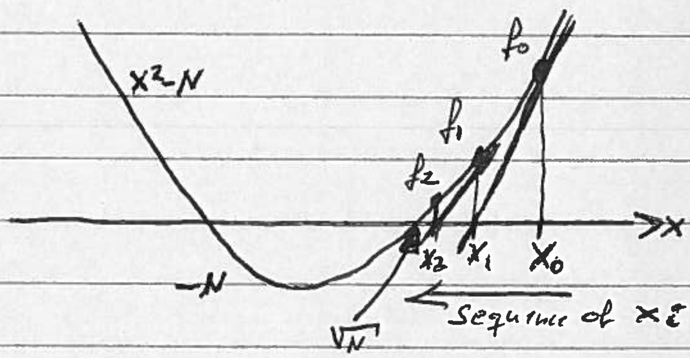
$x^2 = N$

$\Leftrightarrow \boxed{x^2 - N = 0}$

"Find zero of function $\boxed{f(x) = x^2 - N}$!"



- Iteration principle:



- FORMULA:

$x_0 =$ "initial value"

$$\boxed{x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}}$$

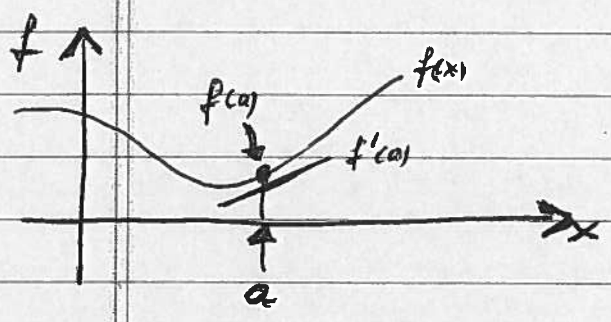
$$= x_i - \frac{x_i^2 - N}{2x_i} = \frac{2x_i^2 - x_i^2 + N}{2x_i} = \frac{x_i^2 + N}{2x_i} = \frac{1}{2}\left(x_i + \frac{N}{x_i}\right)$$

```
double sqrt (double x)
{double app;      /* approximate value of √ */
  if (x==0) return (0);
  if (x<0)  Error ("...\n");
              ↑ DEFINED IN "genlib.h"
                → PROGRAM TERMINATES WITH THIS ERROR MESSAGE.
  app = x;
  while (!ApproxEqual (x, app*app))        /* MUST BE ROBUST! */
  { app = 0.5*( app + x/app);
  }                                        /* Roberts p. 181 */
  return (app);
}
```

```
bool ApproxEqual (double x, double y)
{return (    AbsValue(x-y)
           / Minimum (AbsValue(x), AbsValue(y))
           < EPSILON );
               ↑ DEFINE AS 0.000001  ???
```

- **TAYLOR APPROXIMATION**

Idea: Use Taylor approx. of a function "centered" at a to estimate $\sqrt{\ }$.



"Developing" function $f(x)$ for $x = a$:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2 + \frac{f'''(a)}{6}(x-a)^3 + \ldots$$

$$= \sum_{i=0}^{\infty} \frac{f^{(i)}(a)}{i!}(x-a)^i$$

"Approximate a function only in terms of polynomials!"

"Taylor Expansion of f(x)" – Used in math.c

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2 + \frac{f'''(a)}{6}(x-a)^3 + \cdots = t_0 + t_1 + t_2 + t_3 + \cdots$$

- HERE: $f(x) = \sqrt{x} = x^{1/2}$

		$a = x = 1$		
$f =$	$x^{1/2}$	1	$t_0 =$	$1 \cdot \frac{(x-1)^0}{0!}$
$f' =$	$\frac{1}{2}x^{-1/2}$	$\frac{1}{2}\cdot 1$	$\Rightarrow t_1 =$	$(\frac{1}{2})\cdot 1 \cdot \frac{(x-1)^1}{1!}$
$f'' =$	$(-\frac{1}{2})\frac{1}{2}x^{-3/2}$	$(-\frac{1}{2})\frac{1}{2}\cdot 1$	$t_2 =$	$(-\frac{1}{2})(\frac{1}{2})\cdot 1 \cdot \frac{(x-1)^2}{2!}$
$f''' =$	$(-\frac{3}{2})(-\frac{1}{2})\frac{1}{2}x^{-5/2}$	$(-\frac{3}{2})(-\frac{1}{2})\frac{1}{2}\cdot 1$	$t_3 =$	$(-\frac{3}{2})(-\frac{1}{2})(\frac{1}{2})\cdot 1 \cdot \frac{(x-1)^3}{3!}$

$$\boxed{\begin{aligned} t_0 &= 1 \\ t_i &= \left(\frac{3}{2} - i\right)\cdot \frac{x-1}{i} \cdot t_{i-1} \end{aligned}}$$

"Initialize $\sqrt{x}$ as t_0, then add terms t_i until $(\text{approx})^2 \approx x$."

(Note: Convergence radius: $(0,2)$; $\Rightarrow$ use $\sqrt{x} = \sqrt{4\hat{x}} = 2\sqrt{\hat{x}}$ to lead to problem in $(0,2)$.)

```
double sqrt(double x)
 {double  sqRt, term;
  int i;
  term = 1;
  sqRt = 1.0;
  for (i = 1; !ApproxEqual(x, sqRt*sqRt); i++)
  { term *= (1.5-i) * (x-1)/i;
    sqRt += term;
  }
  return (sqRt);
 }
```

FIG. 6-8

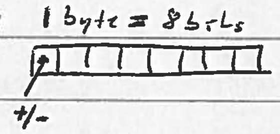
▪ Range of Numerical Types

(A) int → at least up to 32767 $= 2^{15}-1$

(2 bytes = 16 bits, 1 bit for sign)

"ANSI C"

(sometimes up to $2^{31}-1$, 4 bytes)

(

$2^7-1 = 128-1 = \underline{127}$)

Long → at least $2^{31}-1 = \ldots$ (4-bytes repr.)

"ANSI C"

→ printf ("... %ld ..."); — long

unsigned — int → 2 bytes (sometimes 4 bytes)
— long → 4 bytes
— short → 1 byte

unsigned int → at least (up to) 65,535 $= 2^{16}-1$

→ printf ("... %u ...");

(B) Floating-point numbers

→ **float** — least precise of all floating-point repres.

→ **double** — most commonly used ...

→ **long double** — for high-precision scientific computing