CS101 Introduction to computing

Problem Solving (Computing)

A. Sahu and P. Mitra
Dept of Comp. Sc. & Engg.
Indian Institute of Technology Guwahati

Outline

- Problem Solving : Process involves
  - Definition, Analysis, Solution Approaches, Correctness, Programming, Testing
- Loop invariant and loop termination
- Many Problem Solving Examples
  - 7 Problems **(Solution Method not given)**
  - 3 problems **(Solution Method given)**

Reference : R G Dromey, "***How to solve it by Computer***", Pearson Education India, 2009

Fibonacci Computation

- **Problem:** Given a number **n**, generate nth member of Fibonacci sequence
- Definition of Fibonacci sequence $\mathbf{f_n}$ is

  $f_1=0$, $f_2=1$, $f_n=f_{n-1}+f_{n-2}$

  So sequence is

f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	F_{10}
0	1	1	2	3	5	8	13	21	34
		1+0	1+1	2+1	3+2	5+3	8+5	13+8	21+13

Nth Fibonacci Approach-1

- Start from f1 and f2, go upto nth

```
fnm2=0; fnm1=1; n=2;
while(n<=N){
      fn = fnm2 + fnm1;
      fnm2=fnm1;
      fnm1=fn;
      n = n + 1;
}
```

- How good it is ?
- Number of iteration in while loop : **N-2**

Nth Fibonacci Approach-2

- Is there any better approaches?

f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	F_{10}
0	1	1	2	3	5	8	13	21	34

- Observations
  - $f_8 = f_5^2+f_4^2 = 3^2+2^2=13$, $f_{10} = f_6^2+f_5^2 = 5^2+3^2=34$
  - If we look at closely
  - $f_{2n}=f_{n+1}^2+f_n^2$ and $f_{2n+1}=2f_nf_{n+1}+f_{n+1}^2$
  - So f_{2n} and f_{2n+1} depends on f_n, f_{n+1}
  - **Omitting prove for this as of now**

Nth Fibonacci

$f_n = f_{n-1} + f_{n-2}$, with $f_1=1$, $f_0=1$

$$\begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix}$$

$$\begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} f_1 \\ f_0 \end{pmatrix}$$

Nth Fibonacci Approach-2

– Can you think : Log time approach ?

```
int n, fn, fnpl, f2n, f2np1;
// Put code for Input n
fn=0; fnp1=1; i=1;
while(n > 1){
  if (n%2==1) d[i]=1; else d[i]=0;
  n=n/2; }
while(i>=1){
     f2n=fn*fn+fnp1*fnp1;
     f2np1=2*fn*fnp1+fnp1*fnp1;
     if(d[i]){fn=f2n; fnp1=f2np1;}
     else {fn=f2np1;fnpl=f2np1+f2n;}
     i=i-1;}
//Display fn as nth Fibonacci
```

Will be discussed after covering Arrays

Problem 7

GCD of two integer number

Greatest Common Divisor

- Given two numbers **n** and **m** find their **greatest common divisor**
- possible approach
  - find the common primes in the prime factorizations
- **Basic Approaches**

C Code: GCD using substraction

```
int n1, n2;;
// Put code for Input n1 and n2
while (n1!=n2)  {
  while (n1 > n2) n1 = n1 - n2;
  while (n2 > n1) n2 = n2 - n1;
}
//Put code to Display GCD=n1
```

C Code: GCD using substraction

```
int n1, n2;;
// Put code for Input n1 and n2
while (n1!=n2)  {
  if (n1 > n2)  n1 = n1- n2;
  else  n2 = n2 - n1;
}
//Put code to Display GCD=n1
```

Same behavior as earlier code

Euclid’s Algorithm

- **Euclid’s algorithm**: one of the oldest algorithms
- Based on simple observation (assume n > m)

  gcd(n,m) = gcd(n-m,m)   (and hence)

  gcd(n,m) = gcd(m, modulo(n,m))
- uses this property to reduce the smaller number repeatedly
- until the smaller number is 0
- larger number then is the **gcd**

C Code: GCD using reminder

```
int n1, n2, GCD;
// Put code for Input n1 and n2
while (!(n1==0 || n2==0))  {
   //while (n1 > n2) n1 = n1 - n2;
   if (n1>n2) n1=n1%n2;
   //while (n2 > n1) n2 = n2 - n1;
   else n2=n2%n1;
}
if (n1==0) GCD=n2; else GCD=n1;
//Put code to Display GCD
```

Euclid's Algorithm

- A **fixed number of operations** performed in each iteration
- **Time depends on number of iterations**
- after every 2 iterations, value of m is reduced by at least half
  – **if modulo(n,m) > m/2** then
    modulo(m,modulo(n,m)) < m/2
- number of iterations is at most $\mathbf{2(\log_2 m+1)}$

Problem Solving Example

- Set B **(Solution Method given)**
  1. Value of PI
  2. Finding values sin(x) using series
  3. Finding root of a function/equation using Bisection Methods

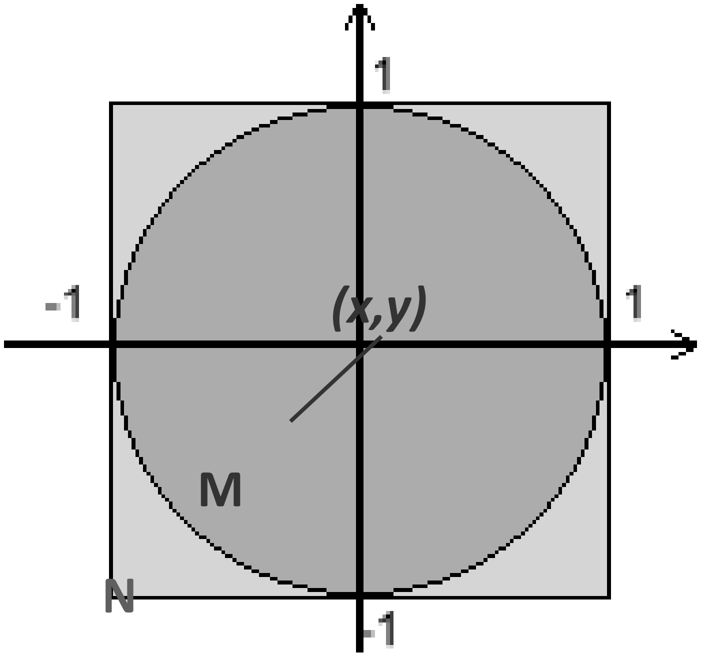
Set B Problem 1

Estimating π using Randomized Method

Estimating π using Randomized Method

- Area of Circle : π . r^2
- If r =1, A=π
- Area of circle in (+,+) quadrant : π/4
- Area of unit square

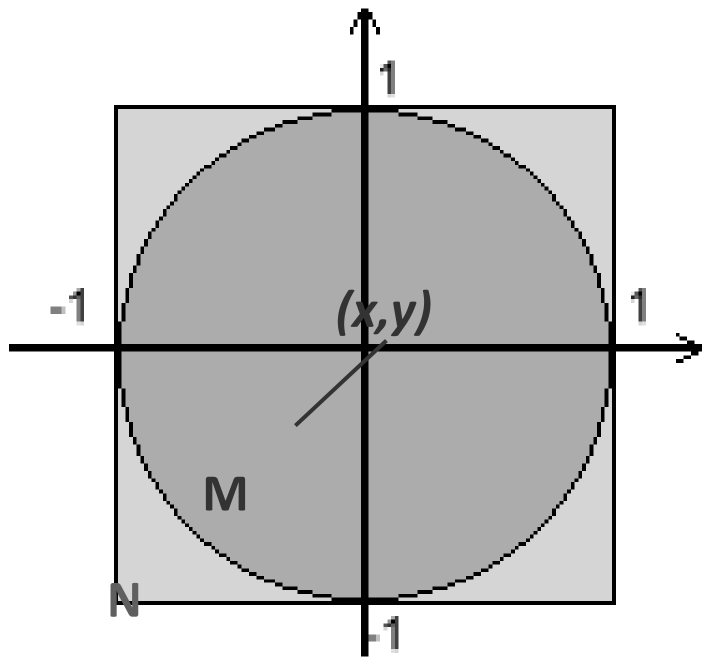
  [x=0 to 1][y=0 to 1] is 1
- **Generate *N* random points**
  - **Point have x, y values**
  - **Between [x=0 to 1][y=0 to 1]**



Estimating π using Randomized Method

- The probability of a random point lying inside the unit circle:

$$\mathbf{P}\left(x^2+y^2<1\right)=\frac{A_{circle}}{A_{square}}=\frac{\pi}{4}$$



- If pick a random point *N* times and *M* of those times the point lies inside the unit circle:

$$\mathbf{P}^{\circ}\left(x^2+y^2<1\right)=\frac{M}{N}$$

- If *N* becomes very large, P=P⁰

$$\pi=\frac{4\cdot M}{N}$$

Value of PI: Randomized Method

```
#define N 10000
   int M=0,i;
    double x,y,z;
    for ( i=0; i<N; i++) {
        x = (double)rand()/RAND_MAX;
        y = (double)rand()/RAND_MAX;
        z = x*x+y*y;
        if (z<=1) M++;
        }
    pi=4.0*(double)M/N;
// Display value of PI
```

Set B Problem 2

Finding values Sin(x) using series sum

Finding values Sin(x) using series sum

- Problem: Design an efficient approach to evaluate the function sin(x) as defined by infinite series of expansion

  $\sin(x) = x/1! - x^3/3! + x^5/5! - x^7/7! + \ldots$

- Approach 1
  - For every term to be used : calculate the value of ith term : Suppose we want to calculate $x^i/i!$
  - Sum all the terms upto

    term > accuracy

```
Ti=1; j=1;
while(j<=i) {
  Ti=Ti * x/j;
  j = j+1;
}
```

Finding values Sin(x) using series sum

- $\sin(x) = x/1! - x^3/3! + x^5/5! - x^7/7! + \ldots$
- Approach 2
  - Current ith term = $-x^2/(i*(i-1))$ * Previous i-1th Term
  - Sum all the terms upto **term > accuracy**

Finding values Sin(x) using series sum

- $\sin(x) = x/1! - x^3/3! + x^5/5! - x^7/7! + \ldots$
- Approach 2
  - Current ith term = $-x^2/(i*(i-1))$ * Previous i-1th Term
  - Sum all the terms upto **term > accuracy**

- Each iteration

```
i = i+2;
term = - term * x*x/(i*(i-1));
SinxVal= SinxVal+ term;
```

C Code Sin(x) using series sum

```
int i=1;
float SinXVal=0, term;
float x, sqrOfx, accuracy;
// Put code for Input x & accuracy
while (term < accuracy) {
    i = i+2;
    term = - term * x*x/(i*(i-1));
    SinxVal= SinxVal+ term;
}
//Put code to Display SinxVal
```

Set B Problem 3

Finding root of a function Bisection Methods

Bisection Method

- **Bisection Method** is a numerical method in Mathematics to find a **root** of a given *function*
- **Root of a function *f(x)*** **is** value *a* such that:

$$f(a) = 0$$

- **Example:**

Function: $f(x) = x^2 - 4$

Roots: $x = -2, \; x = 2$

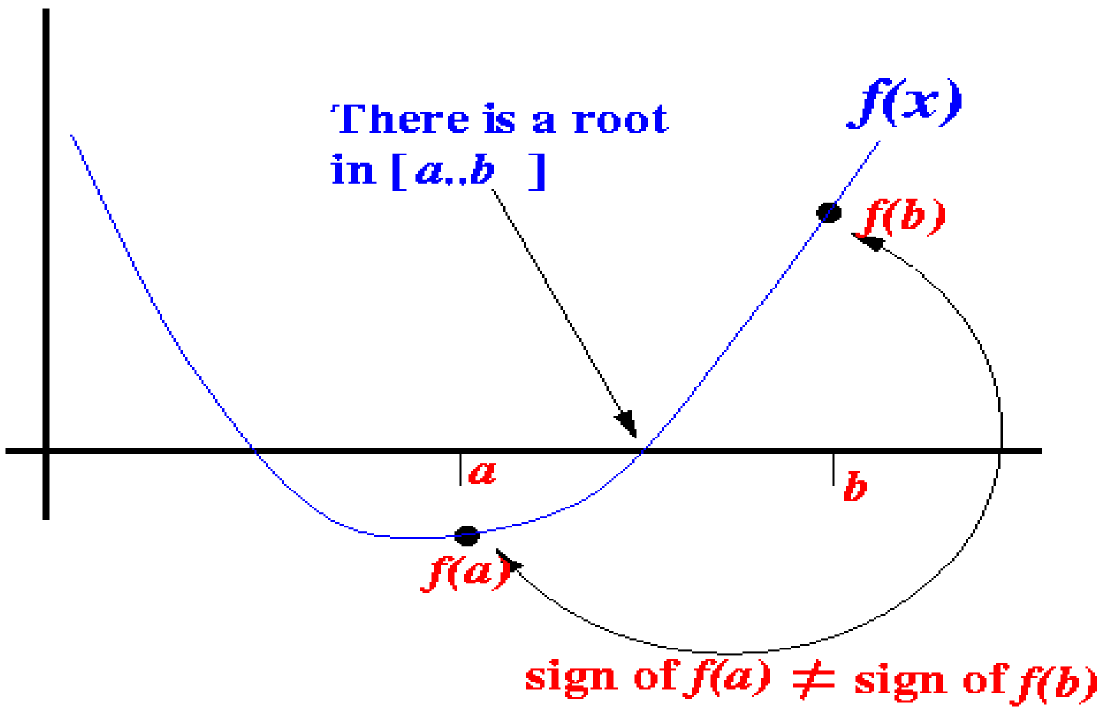
Because:

$$f(-2) = (-2)^2 - 4 = 4 - 4 = 0$$

$$f(2) = (2)^2 - 4 = 4 - 4 = 0$$

A Mathematical Property

- If a function $f(x)$ is continuous on the interval [a..b] and sign of $f(a) \neq$ sign of $f(b)$, then
- There is a value $c \in [a..b]$ such that: $f(c) = 0$
  I.e., there is a root c in the interval [a..b]



Bisection Method

- Is a *successive* approximation method that narrows down an interval
  - that contains a root of the function $f(x)$
- G*iven* an initial interval [$a..b$]
  - Contains a root
  - sign of $f(a) \neq$ sign of $f(b)$

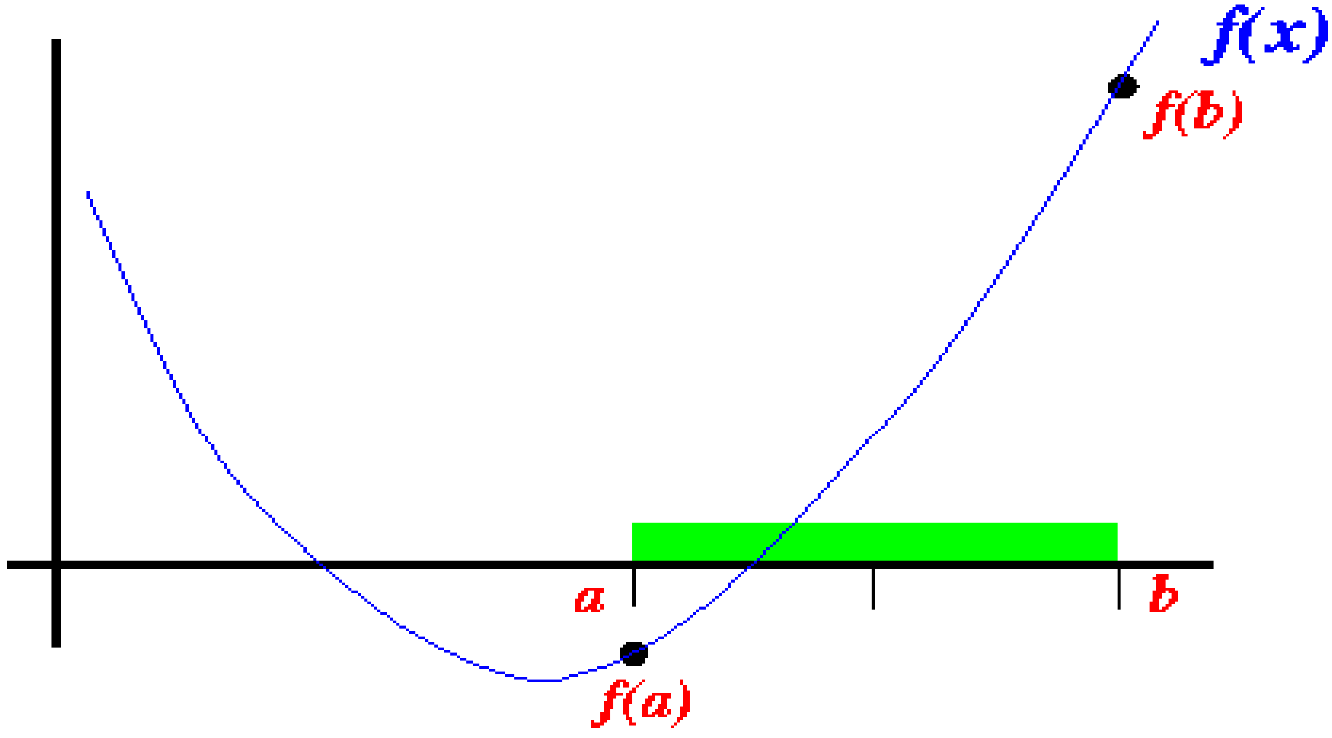
Bisection Method

- **Bisection Method** will
  - *Cut the interval* into 2 halves and check which half interval contains a root of the function
  - will keep *cut the interval* in halves until the resulting interval is extremely small

The root is then *approximately equal* to *any value* in the final (very small) interval.

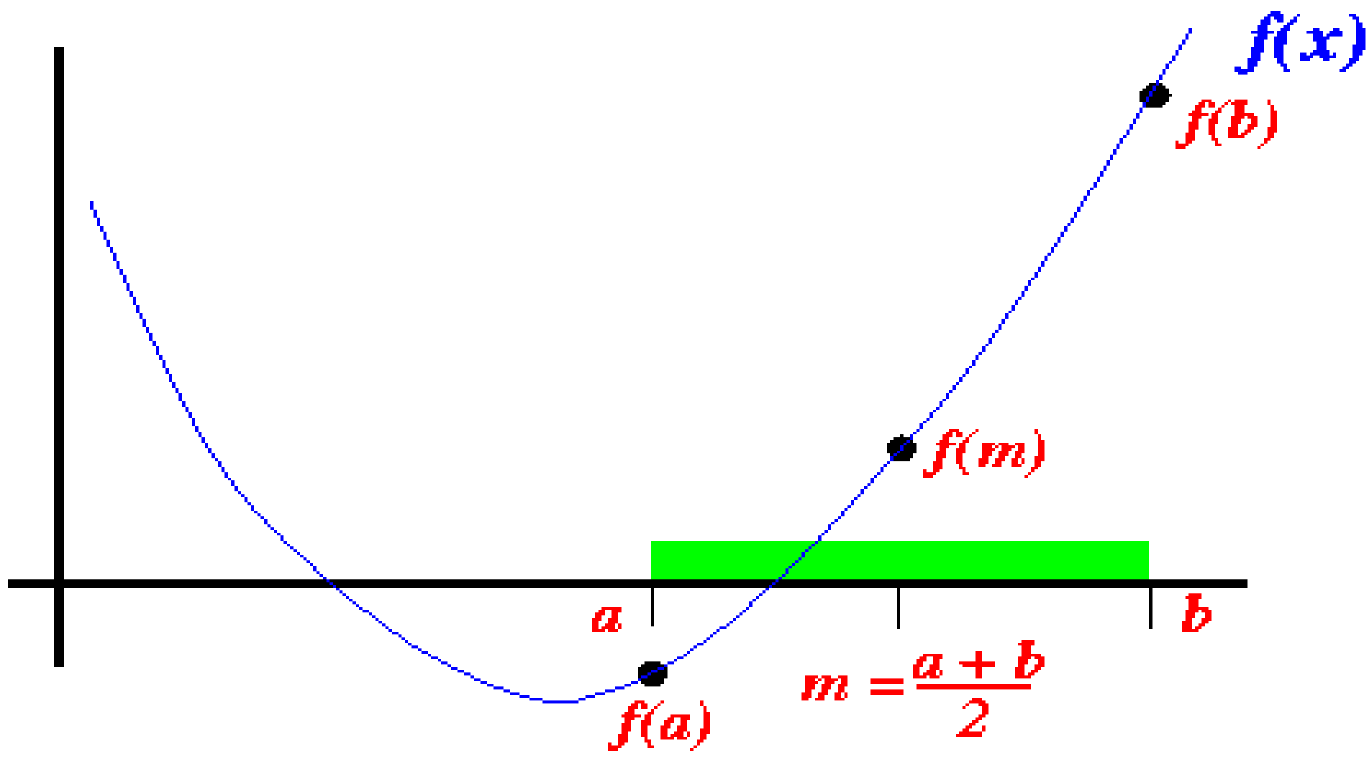
Bisection Method Example

- Suppose the interval [*a*..*b*] is as follows:



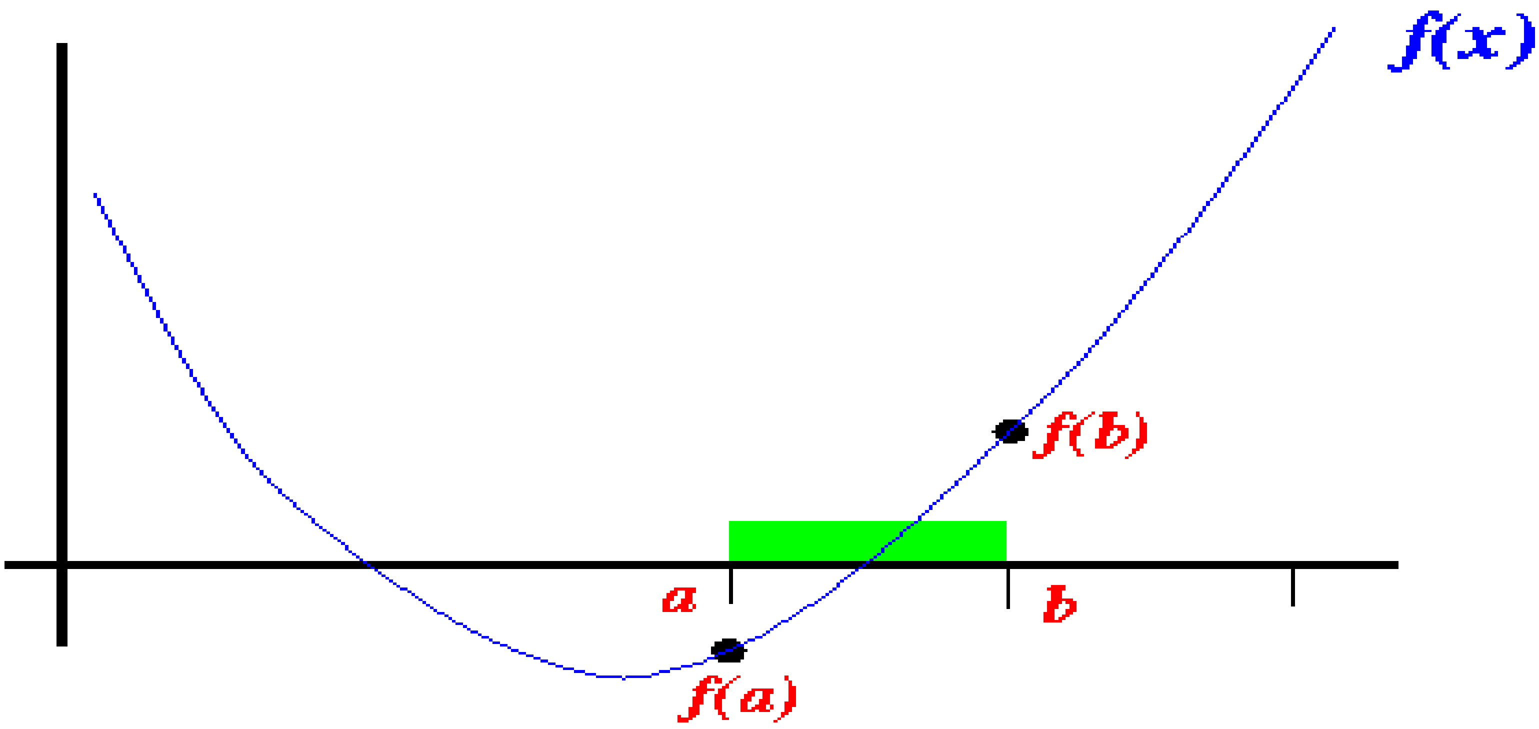
Bisection Method Example

We cut the interval [a..b] in the middle: $m = (a+b)/2$



Bisection Method Example

- Because sign of $f(m) \neq$ sign of $f(a)$, we *proceed* with the search in the *new* interval [$a..b$]:



C Code: Bisection Method

For Function : x^3+2x-5, a=2, b =3

```
float a, b, Fa, Fb, Fx;
//Code for Input a, b and accuracy
Fa=a*a*a-2*a-5; Fb=b*b*b-2*b-5;
x=a;x1=b;
while( abs(x-x1)>accuracy) {
     x1=x; x=(a+b)/2;
     Fx=x*x*x-2x-5;
     if(Fa*Fx<0) b=x; else a=x;
}
//Code print root X
```