COMP 208
Computers in Engineering

Lecture 25

Jun Wang
School of Computer Science
McGill University

Fall 2007

Formatted I/O in Fortran (lecture 10)

- embedded format string

```
WRITE(*,"(A15,F7.2)") "Total Cost: ", price+gst+pst
```

- format statement

```
WRITE(*, 100) "Total Cost: ", price+gst+pst
100 FORMAT (A15,F7.2)
```

- format codes
  - INTEGERS: `rIw`, e.g. `I4`, `3I4`
  - REAL: `rFw.d`, `rEw.d`, e.g. `F7.2`, `3F7.2`, `E15.3`
  - CHARACTER: `A`, `Aw`, e.g. `A`, `A10`

sub-programs

- FORTRAN
  - functions and subroutines
  - pass by reference
- C
  - functions
  - pass by value

```
type FUNCTION function-name
            (arg1, arg2, …, argn)
    IMPLICIT NONE
    [declarations]
    [statements]
END FUNCTION function-name
```

```
type function-name
          (arg1, arg2, …, argn)
{
    [declarations]
    [statements]
}
```

```
SUBROUTINE subroutine-name
            (arg1, arg2, …, argn)
    IMPLICIT NONE
    [declarations]
    [statements]
END SUBROUTINE subroutine-name
```

sub-programs: parameter passing

```
PROGRAM foo
  IMPLICIT NONE
  INTEGER :: fun, x=3, y;
  WRITE(*,*) x // 1st write
  y = fun(x)
  WRITE(*,*) x // 2nd write
  call sub(x)
  WRITE(*,*) x // 3rd write
  call sub(x+1)
  WRITE(*,*) x // 4th write
END PROGRAM

INTEGER FUNCTION fun(a)
  IMPLICIT NONE
  INTEGER :: a
  a = a+1
  fun = 0
END FUNCTION fun

SUBROUTINE sub(a)
  IMPLICIT NONE
  INTEGER :: a
  a = a+2
END SUBROUTINE sub
```

```
#include <stdio.h>
int fun(int a);
void sub(int a);
int main() {
  int x=3, y;
  printf(“%d\n”, x); // 1st write
  y = fun(x);
  printf(“%d\n”, x); // 2nd write
  sub(x);
  printf(“%d\n”, x); // 3rd write
  sub(x+1);
  printf(“%d\n”, x); // 4th write
}

int fun(int a)
{
  a = a+1;
  return 0;
}

void sub(int a)
{
  a = a+2;
}
```

sub-programs: parameter passing

```
#include <stdio.h>
int fun(int* a);
void sub(int* a);
int main() {
  int x=3, y;
  printf(“%d\n”, x); // 1st write
  y = fun(&x);
  printf(“%d\n”, x); // 2nd write
  sub(&x);
  printf(“%d\n”, x); // 3rd write
  //sub(x+1); //error!
  printf(“%d\n”, x); // 4th write
}

int fun(int* a)
{
  *a = *a+1;
  return 0;
}

void sub(int* a)
{
  *a = *a+2;
}
```

In C, for an actual parameter to be modified, its address must be passed

File I/O

- FORTRAN
  - open file

```
OPEN(UNIT=n, FILE=“filename”)
OPEN(n, FILE=“filename”)
```

```
OPEN (UNIT=10, FILE=“data.txt”)
OPEN (10, FILE=“data.txt”)
```

  - file I/O

```
READ (10, ”(I1)”) x
```

  - close file

```
CLOSE(UNIT=n)
CLOSE(n)
```

- C
  - open file

```
FILE *fp;
fp=fopen(filename, mode);
```

```
FILE* fp;
fp = fopen(“data.txt”, “r”);
```

  - file I/O

```
fscanf (fp, ”%d”, &x);
```

  - close file

```
fclose(fp);
```

C pointers

- Pointers are variables that hold memory address of other variables.
  - declaration:

```
type *name;
```

  - address-of operator (&):

```
int* ip;
int x;
ip = &x;
```

  - dereferencing operator(*):

```
int* ip;
int x;
ip = &x;
*ip = 3; //same as x=3
```

  - array and pointer
    - array name is a pointer to its first element

```
void foo(int a[]); // same as
void foo(int* a);
```

demo

Searching algorithms

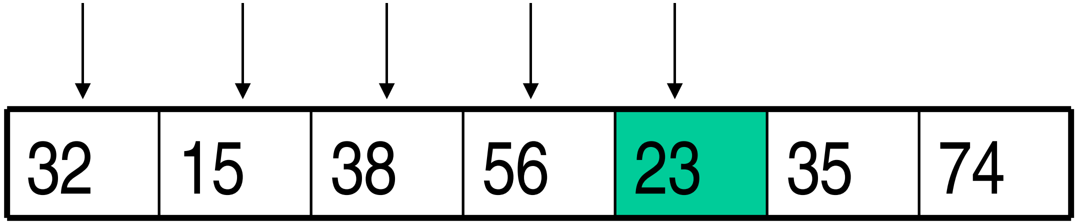
- Linear search
  - O(n)
- Binary search
  - requires data to be sorted in advance
  - O(logn)

Linear searching function

- An example declaration would be:

```
int find(int * list, int size, int number) {
  int i;
  for(i = 0; i < size; ++i)
   if (list[i] == number)
     return i;
  return -1;
}
```

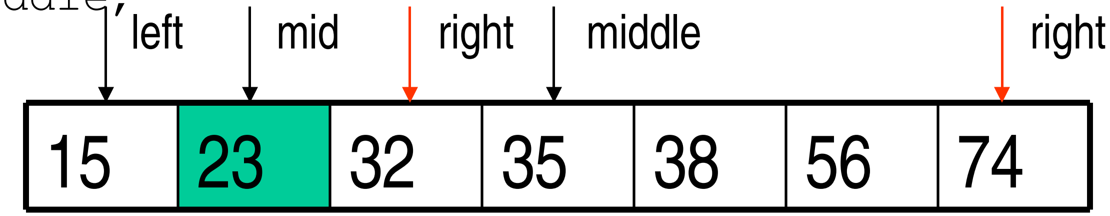
To find 23:



32	15	38	56	23	35	74

Binary search

```
int bfind(int list[], int size, int number){
  int left, right, middle;

  left=0;
  right=size-1;

  do {
    middle = (left+right)/2;
    if(list[middle]==number)
      return middle;
    else if(list[middle] > number)
      left=middle;
    else // if(list[middle] < number)
      right=middle;
  }while(left < right);
  return -1;
}
```

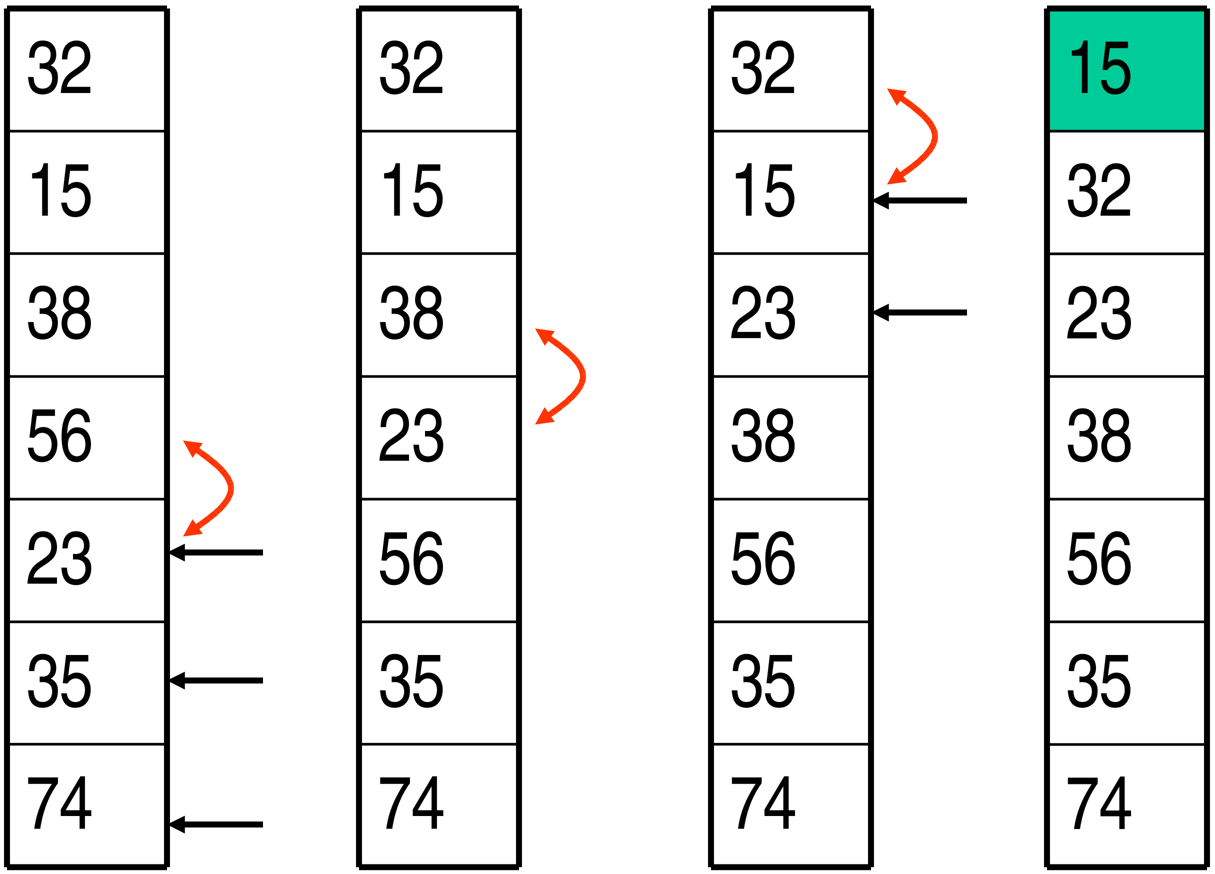


Sorting Algorithms

- bubble
- selection
- insertion
- merge

Bubble sort:

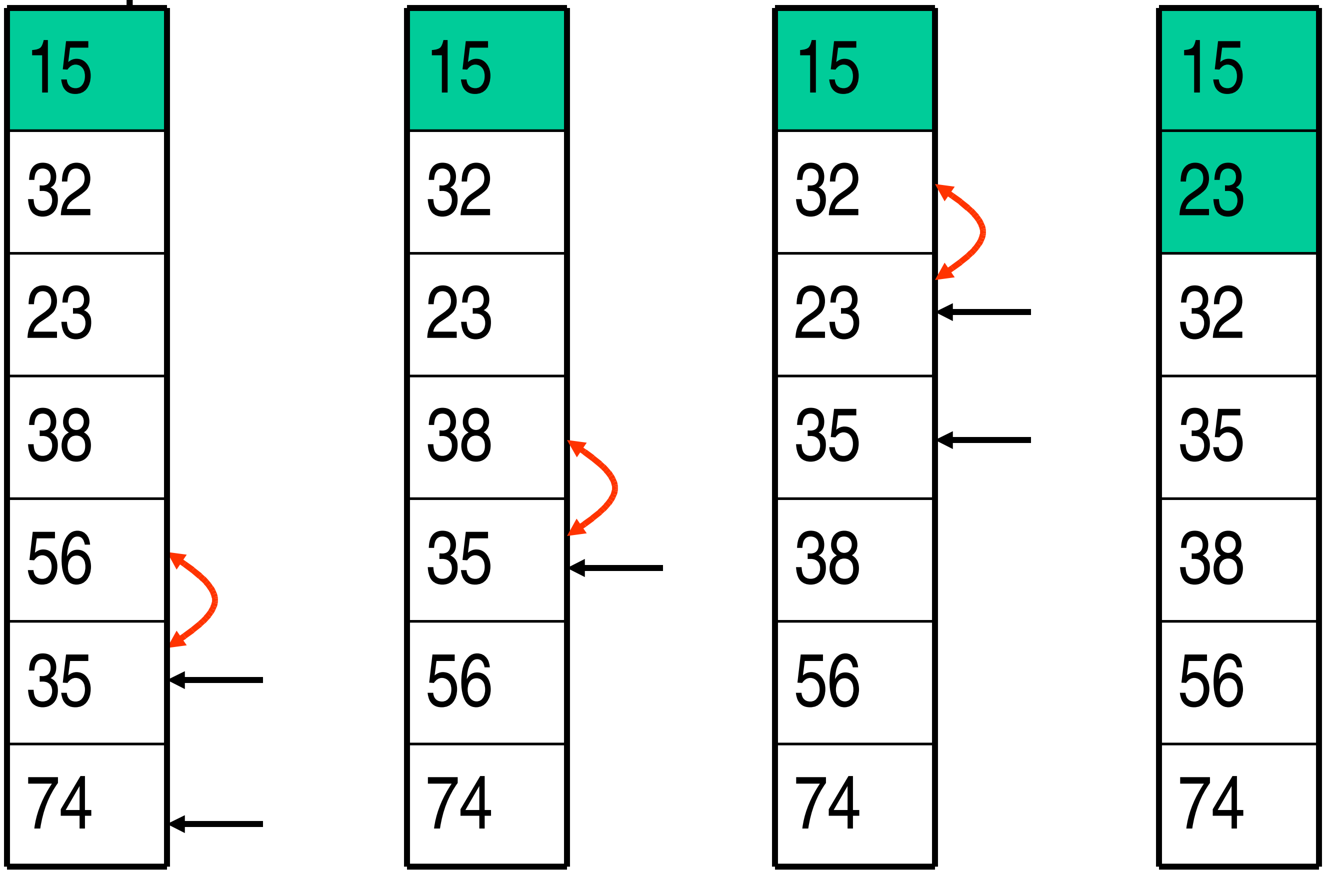
- First pass bubbling:



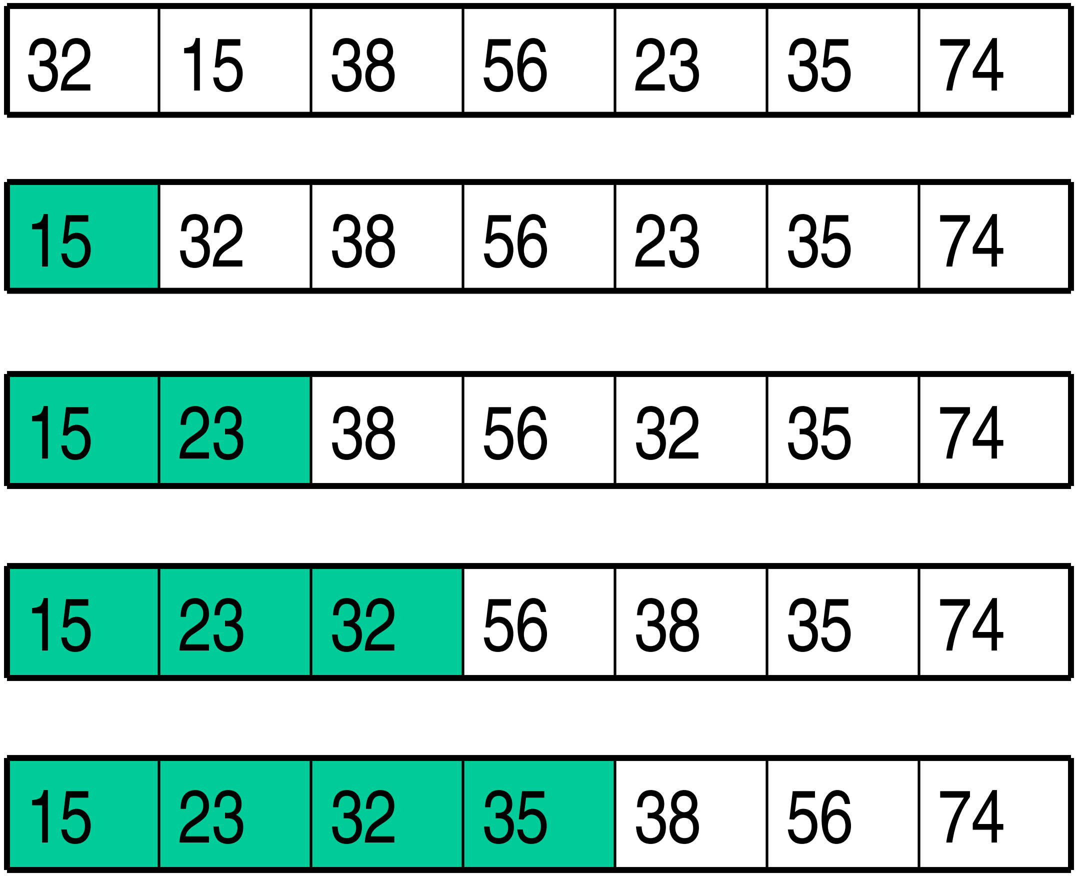
32	32	32	15
15	15	15	32
38	38	23	23
56	23	38	38
23	56	56	56
35	35	35	35
74	74	74	74

Bubble sort

- 2nd pass



Selection sort



32	15	38	56	23	35	74

15	32	38	56	23	35	74

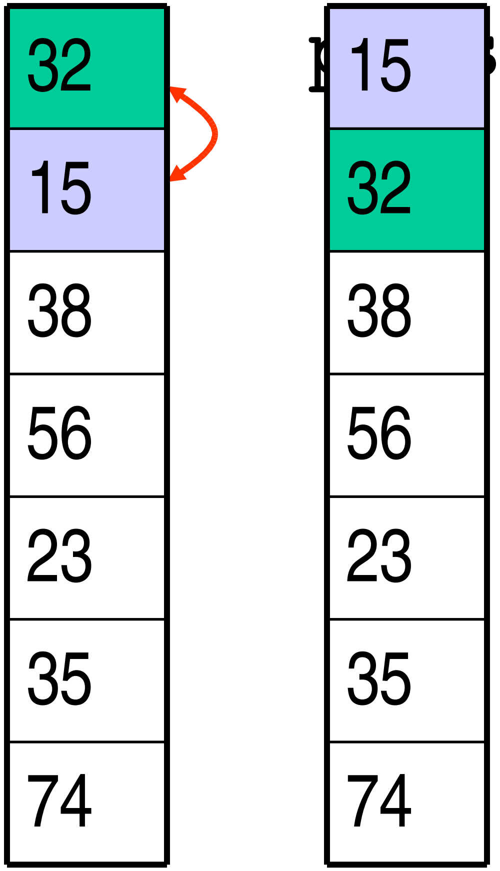
15	23	38	56	32	35	74

15	23	32	56	38	35	74

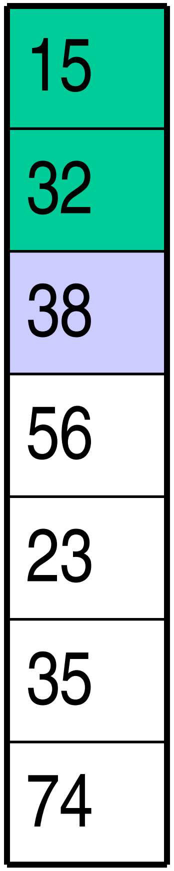
15	23	32	35	38	56	74

Insertion sort:

- First pass:

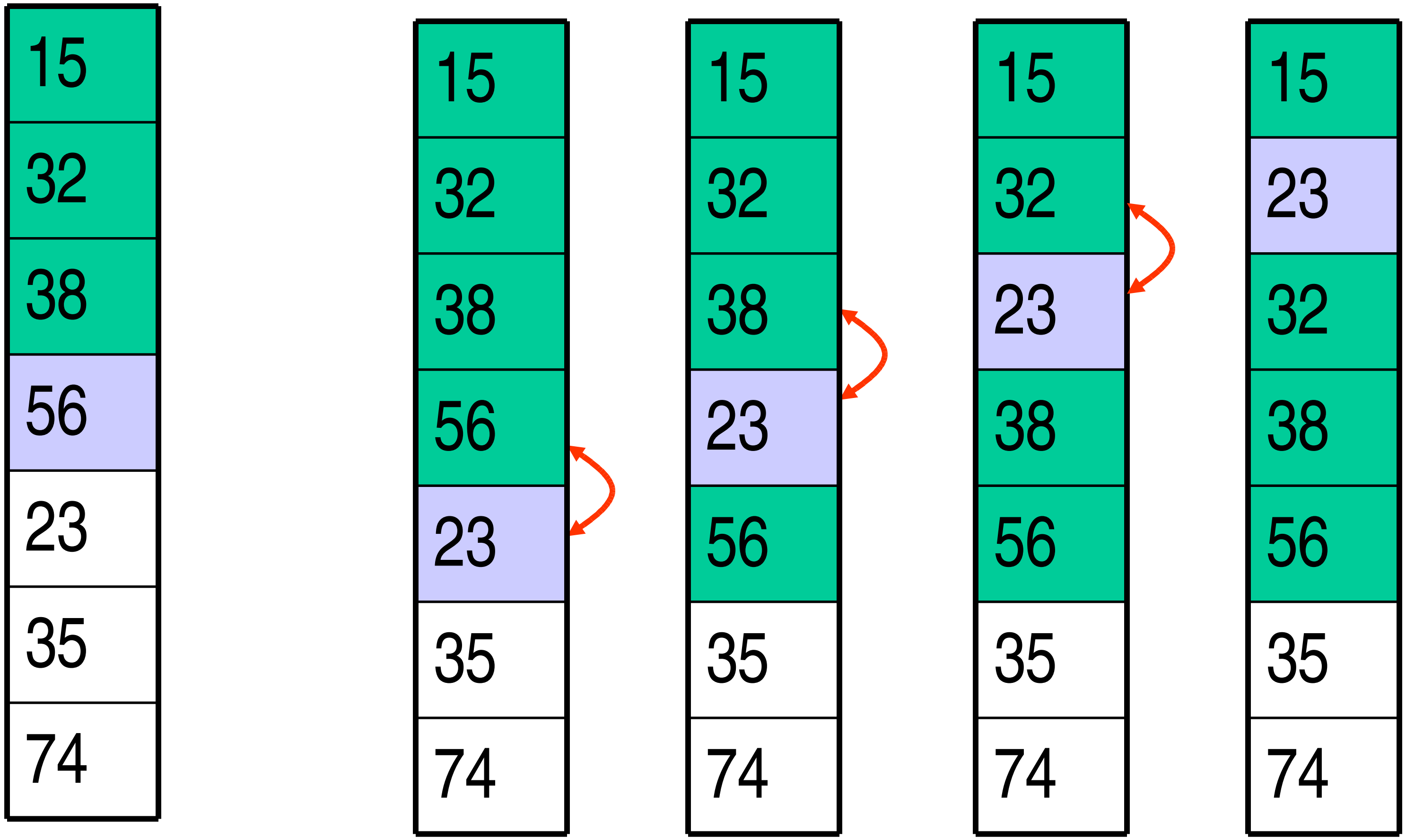


- 2nd pass:



Insertion sort:

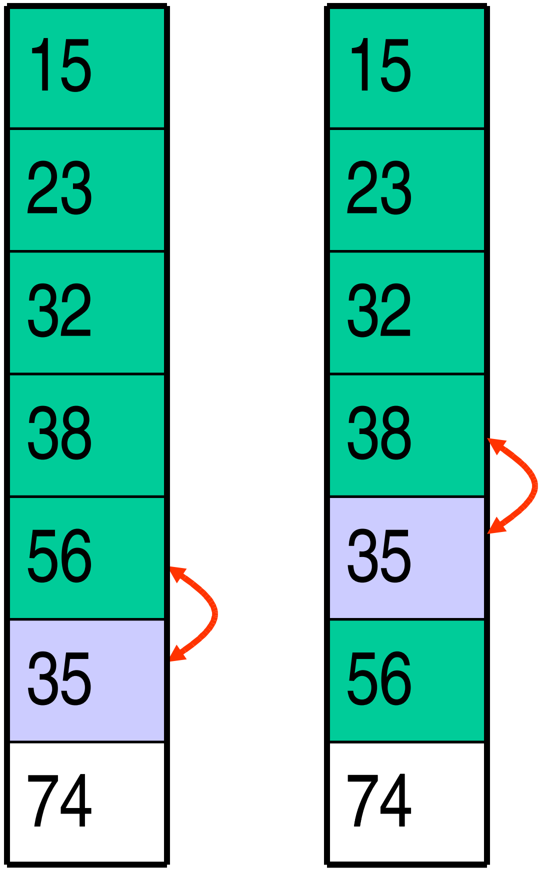
- 3rd pass:
- 4th pass:



3rd pass	4th pass			
15	15	15	15	15
32	32	32	32	23
38	38	38	23	32
56	56	23	38	38
23	23	56	56	56
35	35	35	35	35
74	74	74	74	74

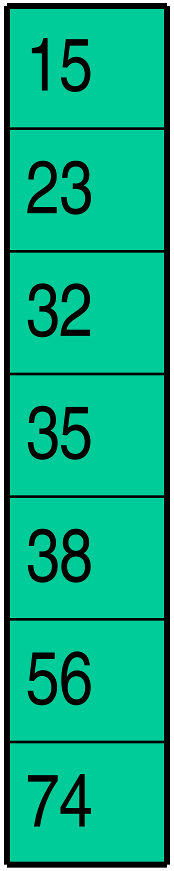
Insertion sort:

- 5th pass:



15	15	15
23	23	23
32	32	32
38	38	35
56	35	38
35	56	56
74	74	74

- 6th pass:



15
23
32
35
38
56
74

Recursion

- Expressing the solution to a problem using the solution to the same problem with a “smaller” size.
- There is a “smallest” case that can be solved directly.
- Example:

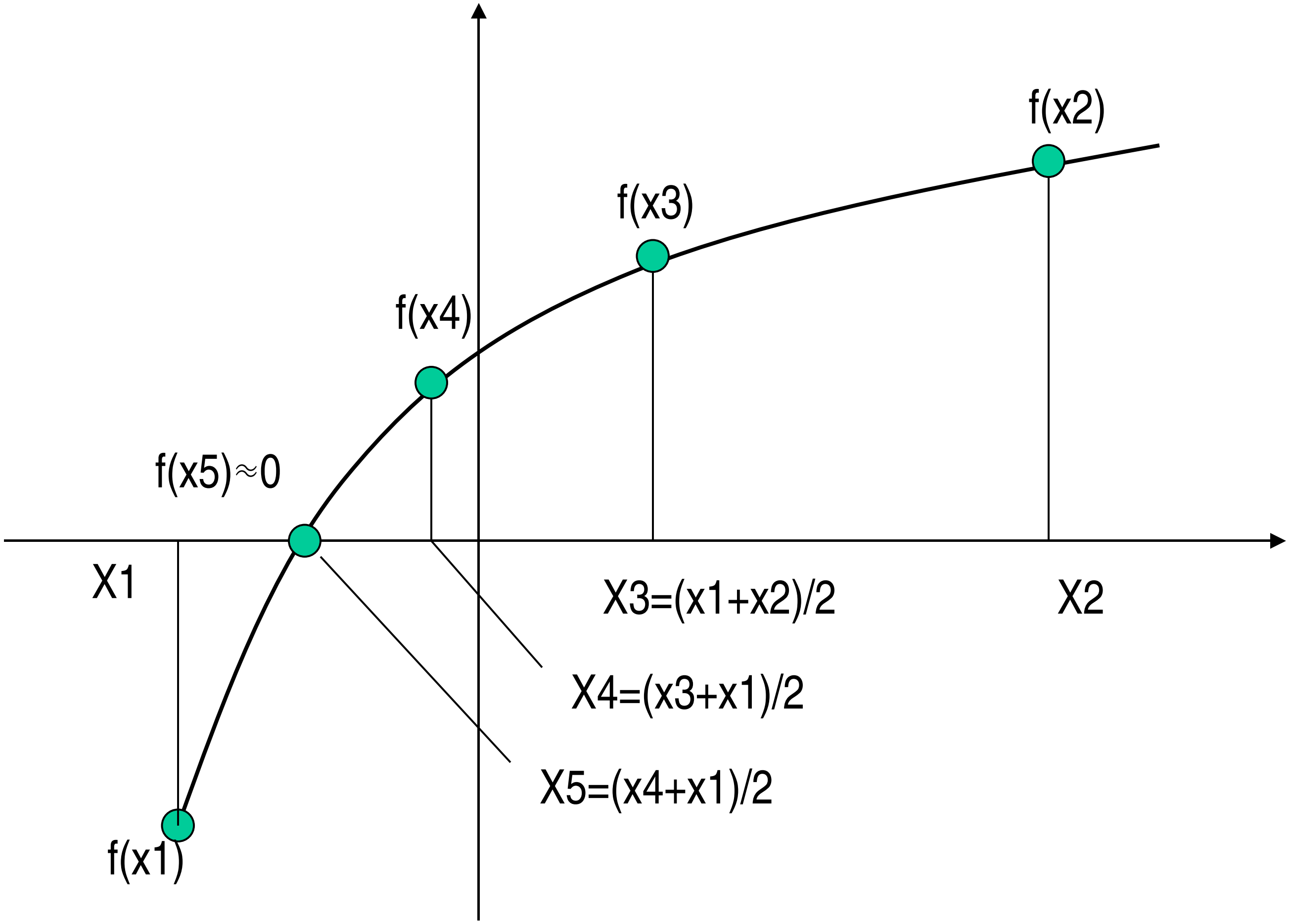
  factorial(n) = 1,   n=1

  factorial(n) = n * factorial(n-1),   for all n>1

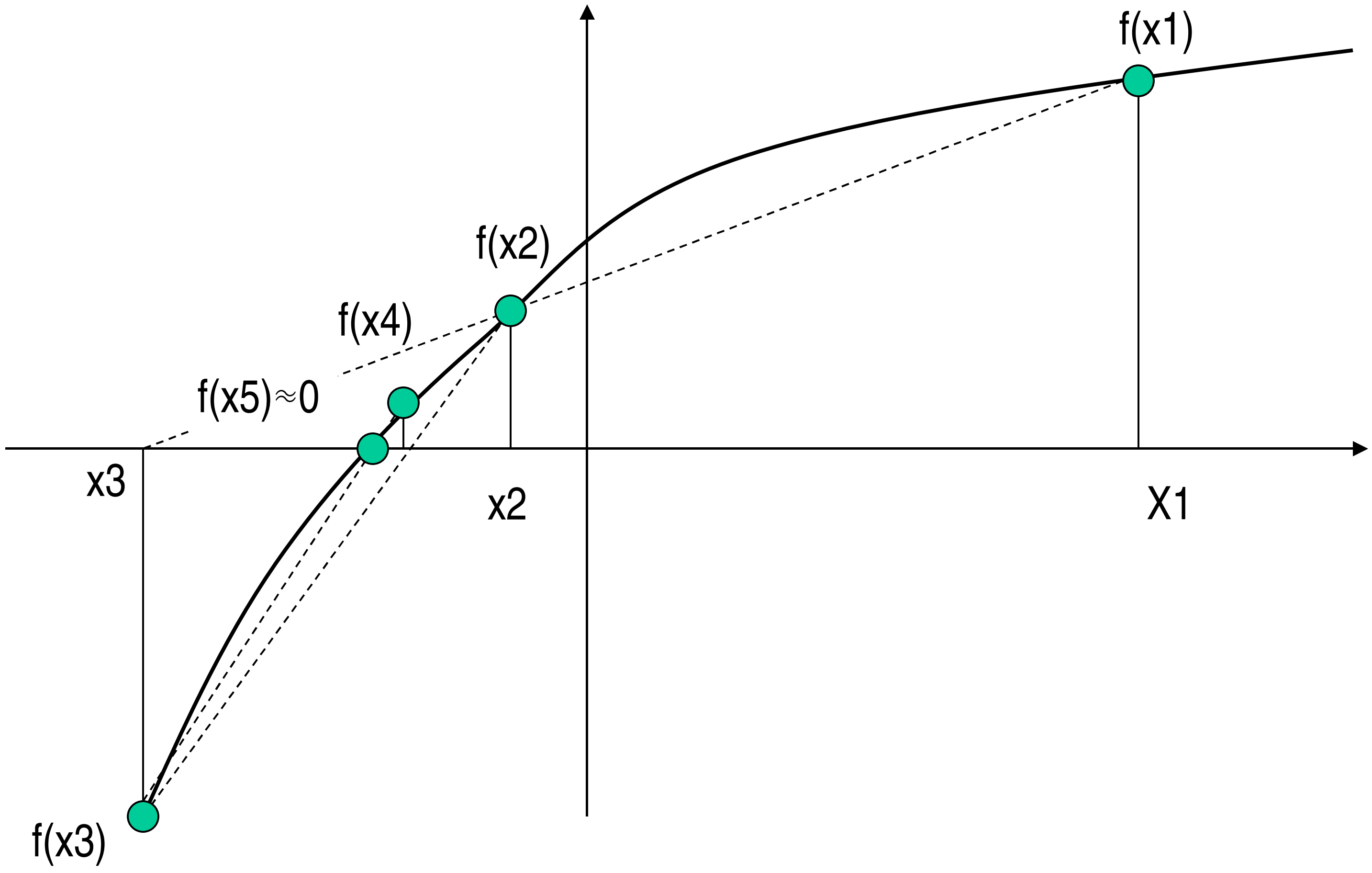
Root finding

- Generate a sequence of numbers that get closer and closer to the root
  - bisection (2 initial numbers)
  - secant (2 initial numbers)
  - false position (2 initial numbers)
  - Newton-Raphson (1 initial number)

Bisection example

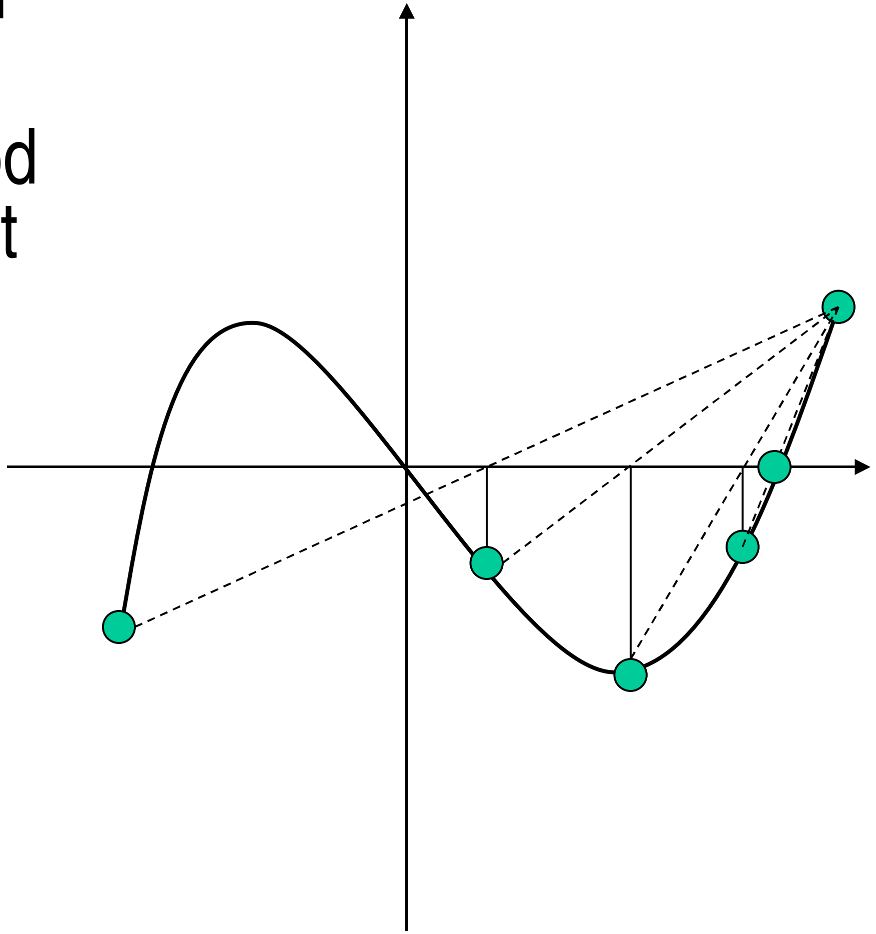


Secant method



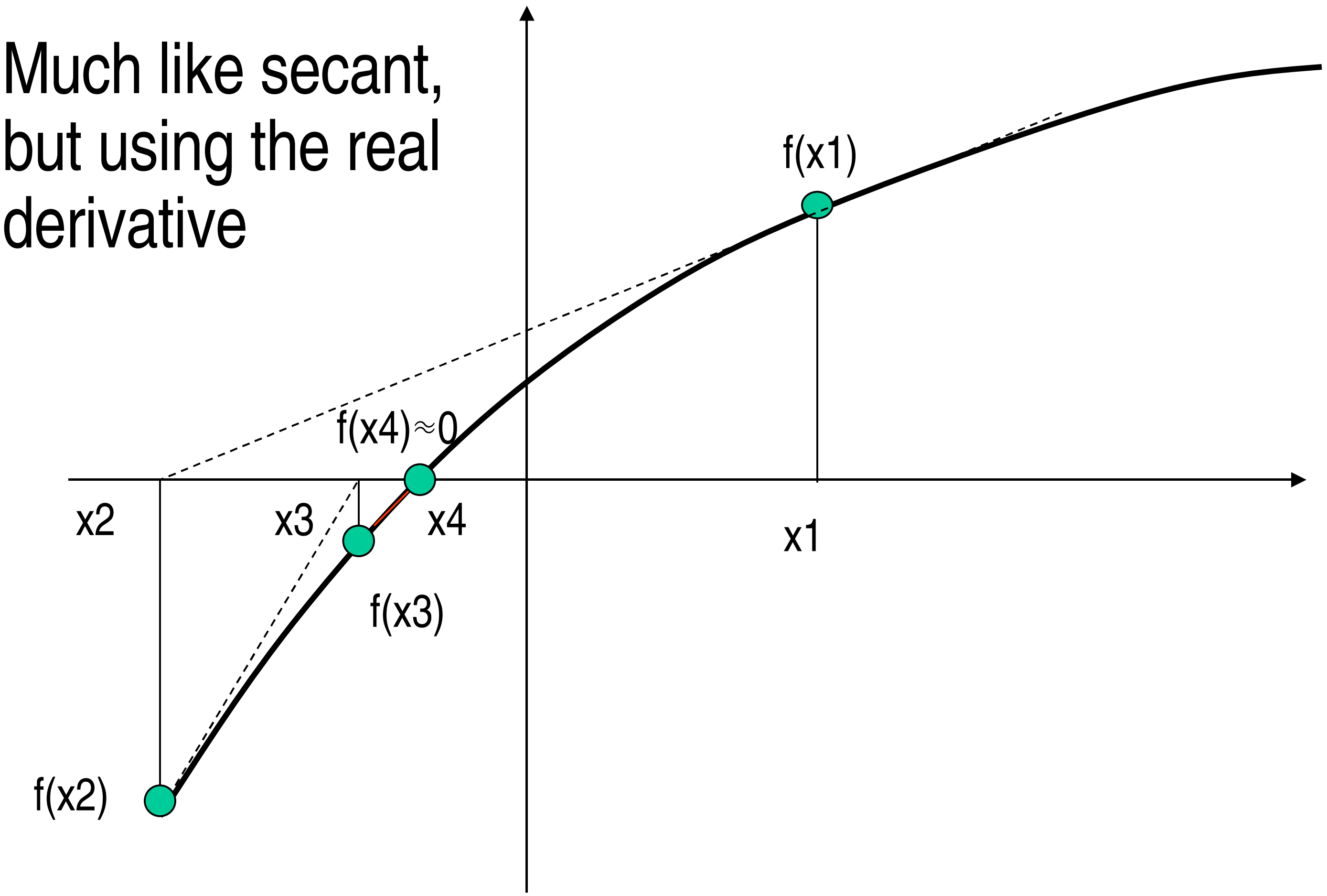
False position method

- The **secant** method retains only the most recent estimate, so the root does not necessarily remain bracketed.
- Combined with bisection in order to bracket the root.



Newton-Raphson

- Much like secant, but using the real derivative



Initial value problems

- Given $y'(x) = f(x,y)$, $y(x_0) = y_0$
  - Euler
  - Runge-Kutta

Euler Method

We want to find an approximate solution to:

```
y' = f(x,y)
y(x0) = y0
```

Now `f(x0,y0)` is the slope of the function at `(x0,y0)`

Approximate the function value at `x0+h` by

```
y0 + h*f(x0,y0)
```

Repeat this process so that

```
x_(n+1) = x_n + h
y_(n+1) = y_n + hf(x_n,y_n)
```

A Very Simple Example

- Given an ODE:
  - dy/dx = x^3 + x*y + y^3
- and an initial value y0=1 at x0=0, calculate the approximation of y1 at x+Δx and y2 at x+2*Δx using Euler method, Δx = 1

solve it

Runge-Kutta Method

The Euler method is not very accurate since the error tends to keep growing.

In the (fourth-order) Runge_Kutta method the derivative is evalutated four times

1. At the initial point
2. Twice at a trial midpoint
3. At a trial endpoint

Runge-Kutta Formula

Use the following to compute the next step

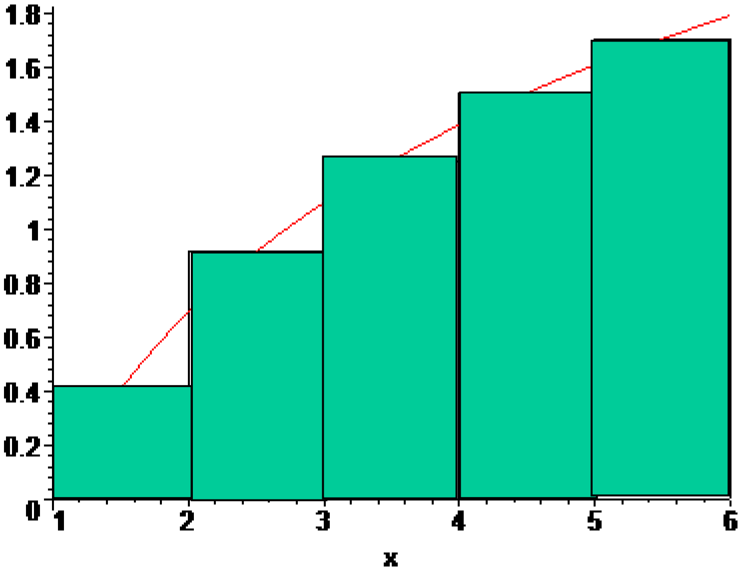
```
k_1 = h*f(x_n, y_n)
k_2 = h*f(x_n+h/2, y_n+k_1/2)
k_3 = h*f(x_n+h/2, y_n+k_2/2)
k_4 = h*f(x_n+h,y_n+k_3)

y_(n+1) = y_n + (k_1+2*k_2+2*k_3+k_4)/6
```

Integration

- Midpoint
- Trapezoid
- Simpson
- Monte-Carlo

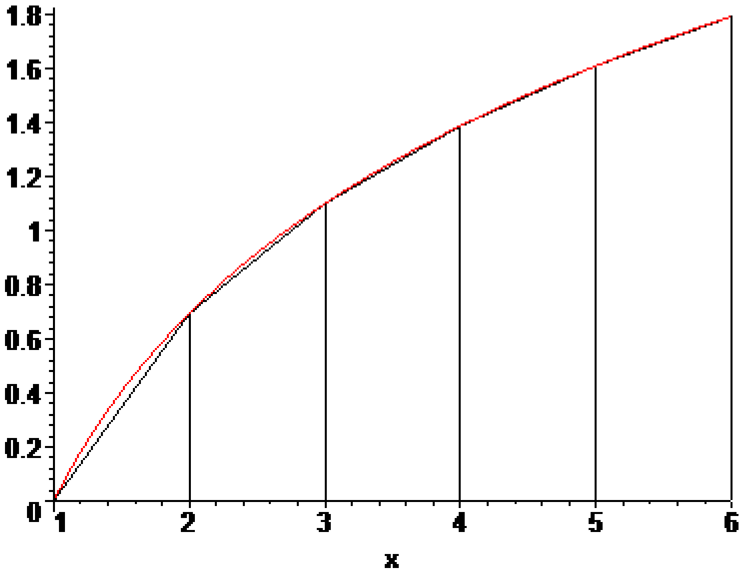
Midpoint rule

- Each rectangular band which makes up the integral is evaluated at the midpoint
- Hopefully the overestimation and underestimation effects cancel out on each panel



Trapezoidal rule

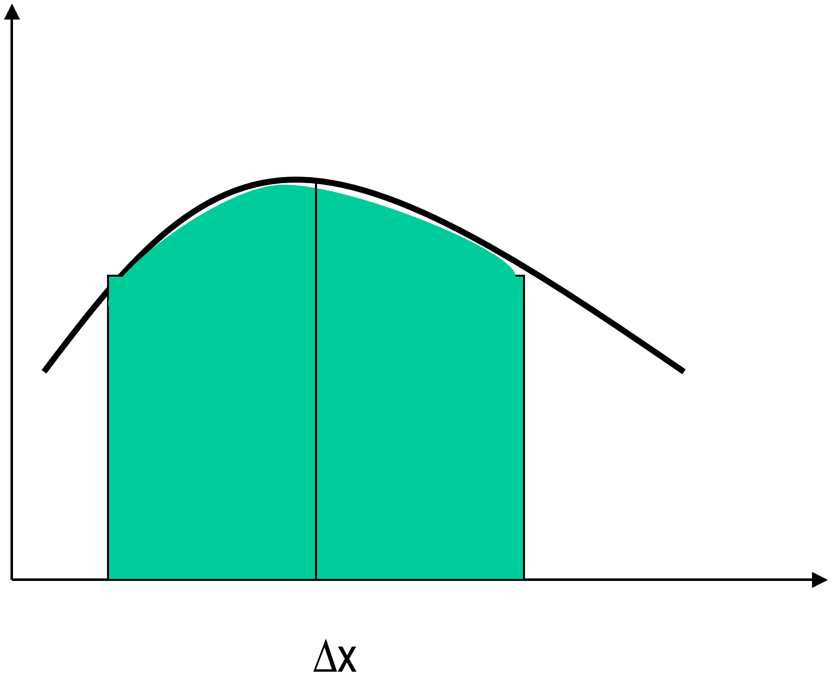
Approximate curves as sequences of straight lines instead of sequences of constants



Simpson: Using a parabola

- As the trapezoidal rule for integration finds the **area** under the line connecting the endpoints of a panel, **Simpson's** rule finds the **area** under the **parabola** which passes through 3 points (the endpoints and the midpoint) on a curve.
- Area under parabola:

$$area = \frac{\Delta x}{6}\left(f(x) + 4f\left(x + \frac{\Delta x}{2} \right) + f(x + \Delta x) \right)$$



Simpson’s formula

- Again, the formula comes from the taylor series:

$$\int_{x}^{x+\Delta x} f(x) = \frac{\Delta x}{6}\left(f(x) + 4f\left(x + \frac{\Delta x}{2} \right) + f(x+\Delta x) \right) + O\left(\Delta x^{5}\right)$$

Gauss Elimination No Pivoting

$$\begin{bmatrix} 1 & -1 & 1 \\ 2 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 & 1 \\ 0 & 3 & -1 \\ 0 & 0 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 6 \end{bmatrix}$$

Gauss Elimination with Pivoting

$$\begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 7 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 4 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 & 1 \\ 0 & 1.5 & 1.5 \\ 0 & -1.5 & 0.5 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 7.5 \\ -1.5 \end{bmatrix} \longrightarrow \begin{bmatrix} 2 & 1 & 1 \\ 0 & 1.5 & 1.5 \\ 0 & 0 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 7.5 \\ 6 \end{bmatrix}$$

Back substitution

after elimination

$$\begin{bmatrix} 2 & 1 & 1 \\ 0 & 1.5 & 1.5 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 7.5 \\ 6 \end{bmatrix}$$

1st pass

$$\begin{bmatrix} 2 & 1 & 1 \\ 0 & 1.5 & 1.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 7.5 \\ 3 \end{bmatrix} \longrightarrow \begin{bmatrix} 2 & 1 & 0 \\ 0 & 1.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \\ 3 \end{bmatrix}$$

2nd pass

$$\begin{bmatrix} 2 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 3 \end{bmatrix} \longrightarrow \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 3 \end{bmatrix}$$

3rd pass

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$