Faculty of Information Technology, Monash University

FIT2004, S2/2016

Week 11: Topological Sort and Numerical Algorithms

Lecturer: Muhammad Aamir Cheema

ACKNOWLEDGMENTS

The slides are based on the material developed by Arun Konagurthu and Lloyd Allison.

Announcements

- Last assessment released
  - Due: 17-Oct-2016 10:00:00
- Programming Competition round 3 closes on Saturday 22-Oct-2016 23:59:00
  - Trophy and Certificates to be given next week
- Start preparing for the final exam earlier
  - Listen to the lectures (or read slides)
  - Attempt tutorial questions
  - Attempt lab questions
  - Attempt past paper - released
  - Most importantly, do not hesitate to seek help

Overview

- Directed Acyclic Graph (DAG)
- Topological Sort on DAG
- Numerical Algorithms
  - Finding root of a function
  - Numerical Integration

Recommended reading

- Cormen et al. Introduction to Algorithms.
  - Chapter 23, Pages 624-638

- http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Graph/DAG/

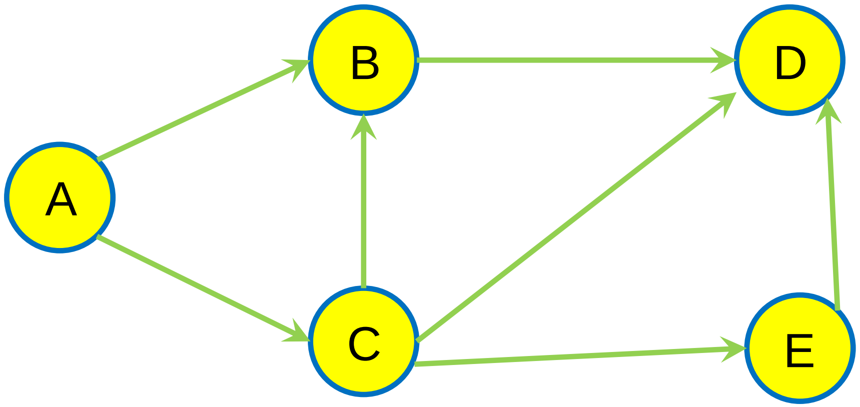
- http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Numerical/Integration/

Directed Acyclic Graph (DAG)

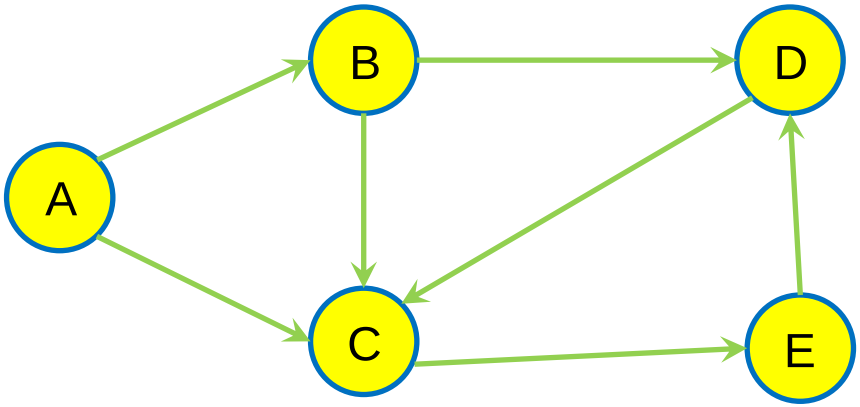
A Directed Acyclic Graph (DAG) is

- Directed
- Acylcic – has no cycles
- Graph

Which of the two graphs is a DAG?



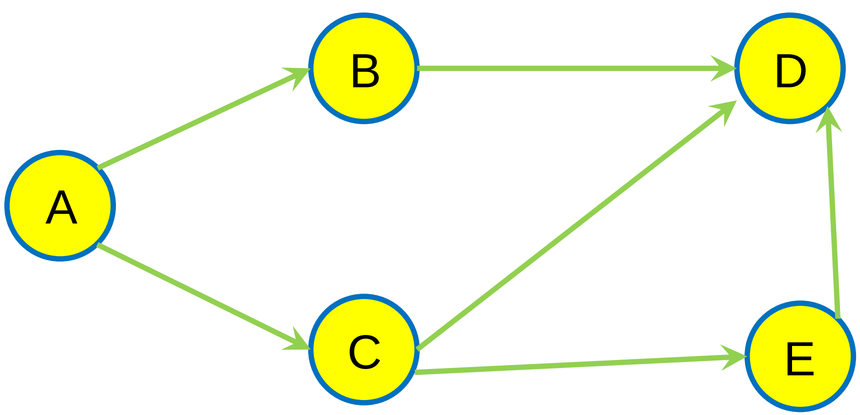
Graph 1

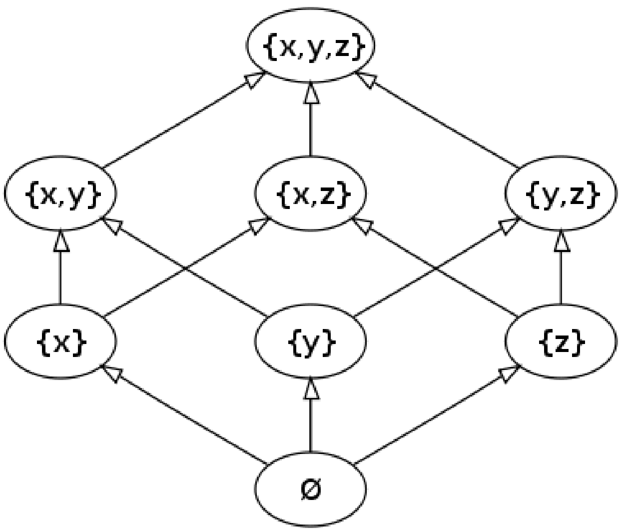


Graph 2

DAG: Examples

- sub-tasks of a project and "must finish before"
  - A → B means task A must finish before task A
  - so, DAGs useful in project management
- relationships between subjects for your degree -- "is prerequisite for"
  - A→B means subject A must be completed before enrolling in subject B
- people genealogy – "is an ancestor of"
  - A → B means A is an ancestor of B
- power sets and "is a subset of"
  - A → B means A is a subset of B





Source: wikipedia

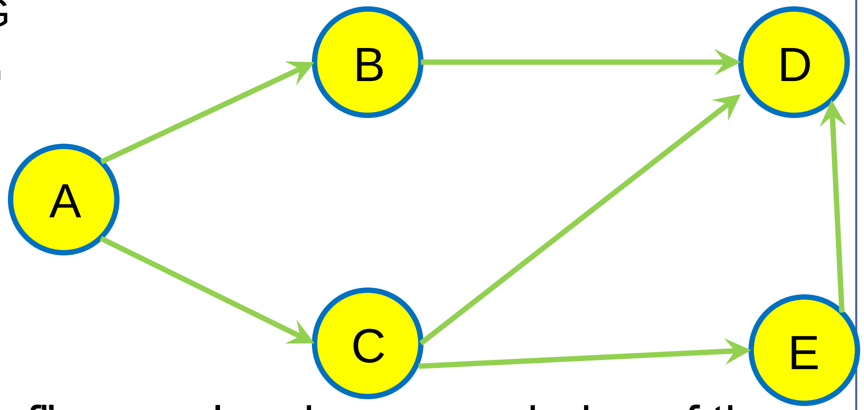
Topological Sort of a DAG

Partial order of vertices in a DAG

- A < B if A→B.
  - Note that if A → B and B→D, we have A < B and B < D which implies that A < D (i.e., transitivity).
- Some vertices may be incomparable (e.g., B and C are incomparable), i.e. A< B and A < C but we do not know whether C < B or B < C.

A topological Sort

- is a permutation of the vertices in the original DAG
- such that for every directed edge u→v of the DAG
- u appears before v in the permutation



Example: A, B, C, E, D

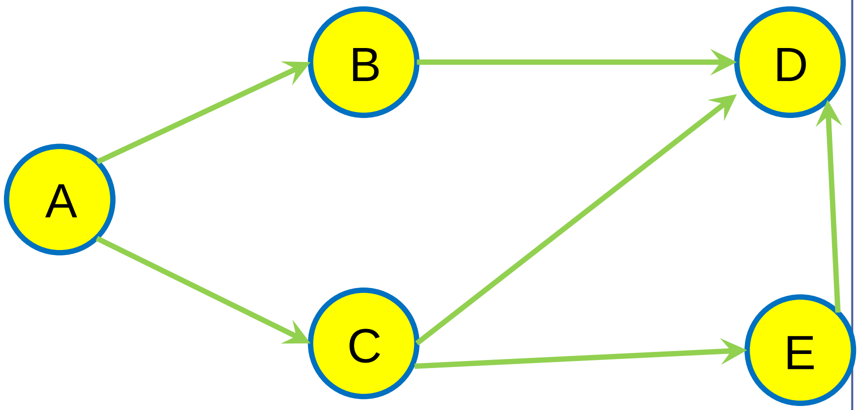
- Topological sort of a DAG of “is prerequisite of” example gives an ordering of the subjects for studying your degree, one at a time, while obeying prerequisise rules.

Topological Sort of a DAG

- A DAG can have many valid topological sorts, e.g., let u and v be two incomparable vertices, u may appear before or after v.

Which of these is not a valid topological sort of the DAG

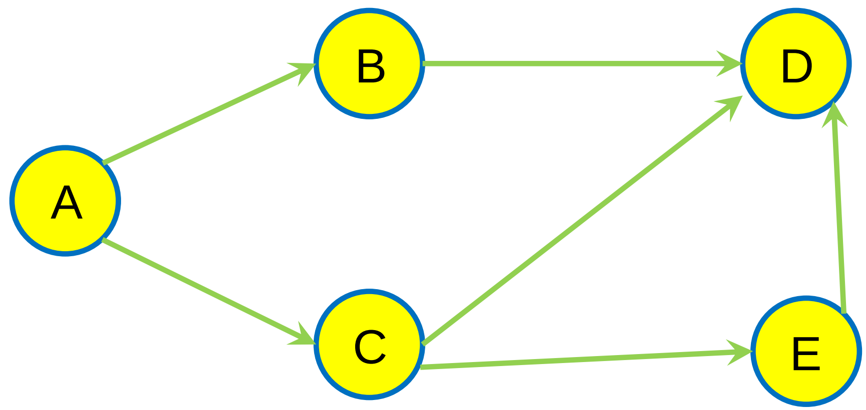
1. A, B, C, E, D
2. A, C, B, E, D
3. A, C, E, B, D
4. A, B, C, E, D
5. A, B, E, C, D



Kahn's Algorithm

```
initialize Sorted to be empty # Sorted will contain the topological sort
initialize a list L with vertices that do not have any incoming edge ?
while L is not empty:
    remove any vertex v from L
    S = S + {v}
    for each outgoing edge <v,u> of v:
        remove edge <v,u> from the graph
        if u has no other incoming edge: ?
            insert u in L # all the vertices that must appear before u has already been added to S
if graph still has some edges:
    return error # graph has a cycle
else:
    return Sorted
```

Time Complexity?



L: | [illegible] | C |

Sorted: | A | B | C | E | D |

Kahn’s Algorithm: Complexity

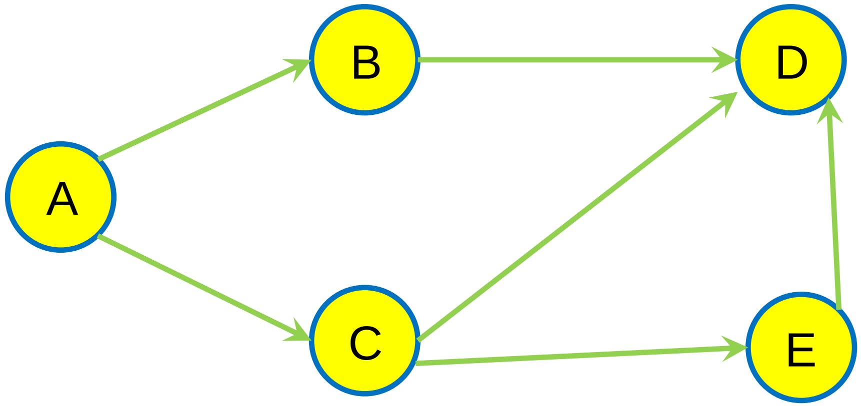
```
initialize Sorted to be empty # Sorted will contain the topological sort
initialize an array IncomingEdges[] with all values initialized to 0
for each edge <u,v>:
    IncomingEdge[v] += 1
initialize a list L with vertices for which IncomingEdges[v] = 0
while L is not empty:
    remove any vertex v from L
    S = S + {v}
    for each outgoing edge <v,u> of v:
        remove edge <v,u> from the graph
        IncomingEdges[u] = IncomingEdges[u] - 1
        if IncomingEdges[u] == 0: # u has no incoming edge
            insert u in L
if graph still has some edges:
    return error # graph has a cycle
else:
    return Sorted
```

O(E+V)

O(E+V)

Time Complexity: O(V+E)

Space Complexity: O(V+E)



Depth First Search

Time Complexity: O(V+E)

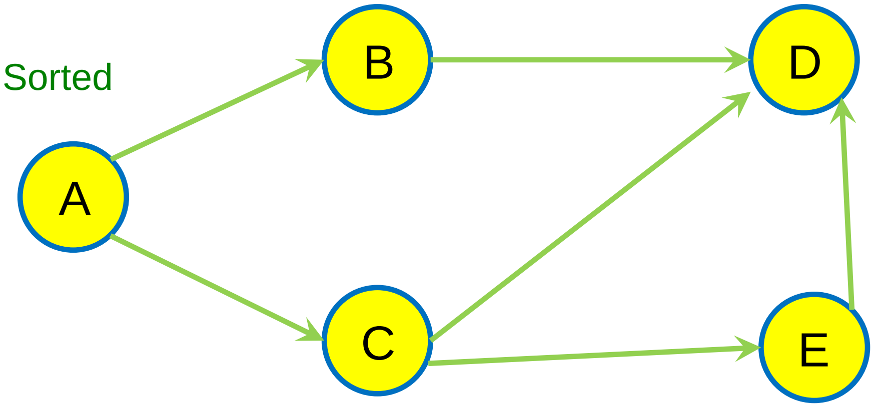
Space Complexity: O(V+E)

```
# Initialization
Sorted = null # stores sorted list of vertices
Color all vertices yellow # yellow indicates not touched
while there are yellow vertices:
        select a yellow vertex x # any vertex can be selected
        DFS(x)

function DFS(vertex x):
        if x is colored green: # green indicates it was accessed during a DFS
                return ERROR #graph has a cycle
        if x is colored yellow:
                color x green
                for each neighbor y of x: # i.e., an edge x-->y
                        DFS(y)
                color x red # red means added to Sorted
                add x to the head of Sorted
# Note that  a vertex is added to Sorted only after all
#its neighboring vertices have been added
```

Sorted:

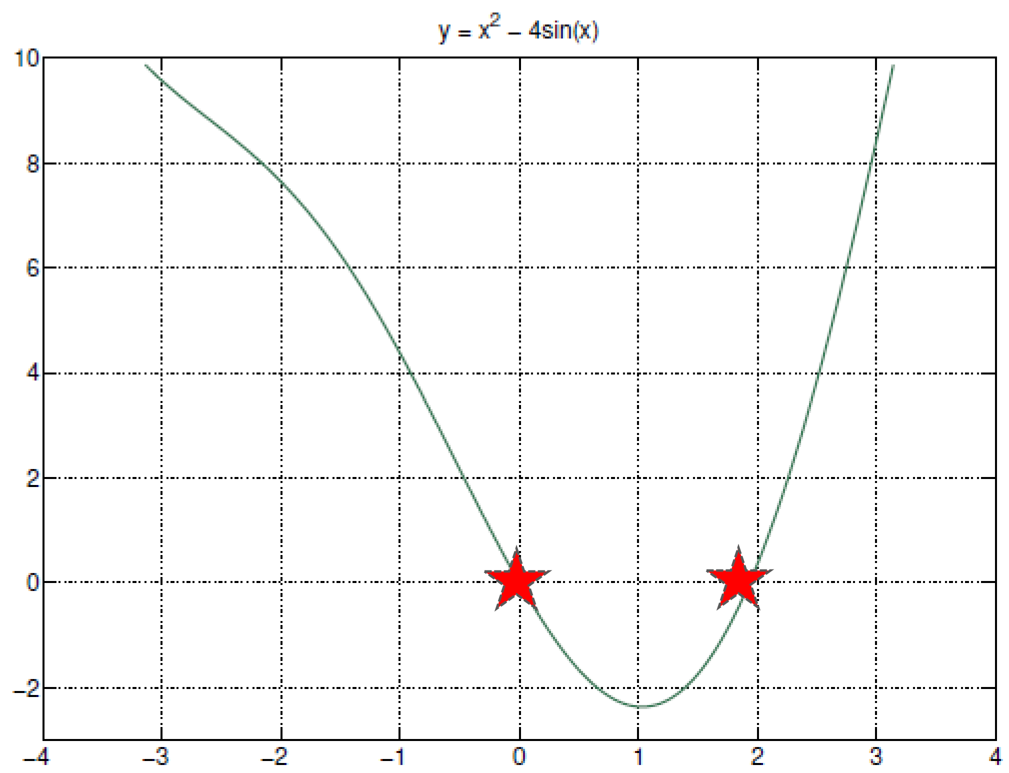
A	B	C	E	D



Algorithms to solve f(x) = 0

Problem

- For what value(s) of x does the function f (x) take on the value 0.
  - E.g., Solve $x^2 - 4 \, Sin(x) = 0$
  - The solution(s) of x such that f (x) = 0 is called:
    - the root of the equation or
    - the zero of the function f



Algorithms to Solve f(x) = 0

Stopping Criteria

- When f (x) = 0 is a nonlinear equation, it CANNOT be solved in a finite number of steps.
- One resorts to iterative methods that produces increasingly accurate approximations to a solution.
- The process terminates once the result is “sufficiently" accurate.

Issues to consider

- In finite precision computing, there may be NO machine representable number x* such that f (x*) is exactly zero.
- The function might have multiple roots.
- The function can be discontinuous.

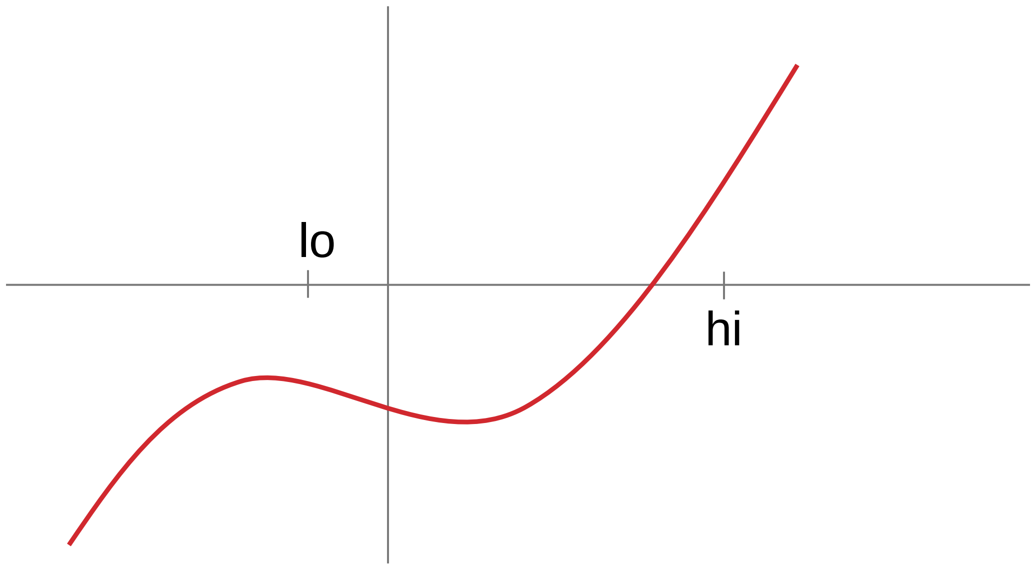
We will study two approaches

- Interval Bisection Method
- Newton’s Method

Interval Bisection Method

Binary Searching on a Continuous Domain

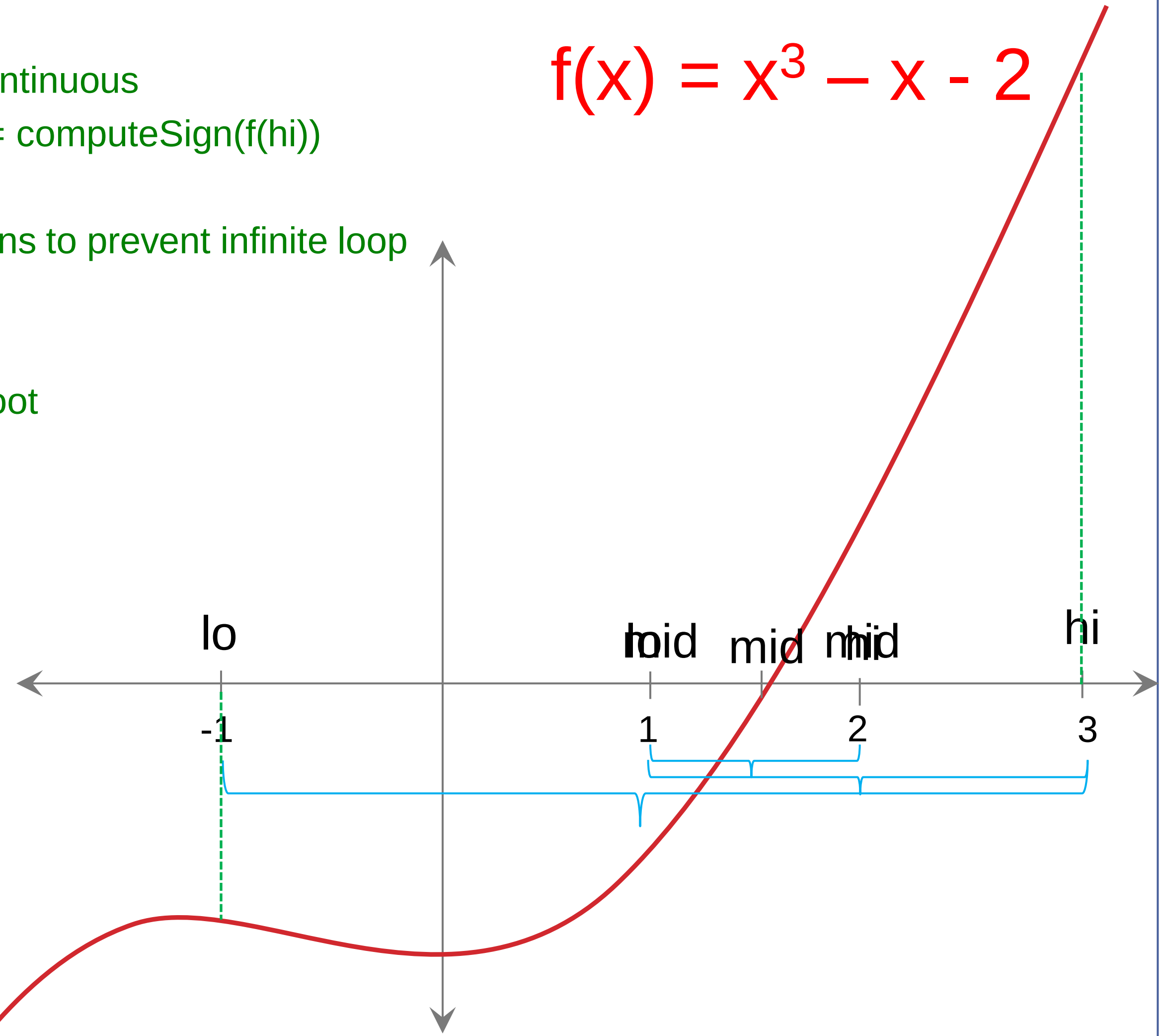
- Start with an initial range [lo,hi] under the constraint that the root falls betweeen lo and hi. This is called bracketing the root.
  - In other words, the bracket [lo,hi] is chosen, such that f (lo) has a value with an opposite sign compared to f (hi).
- At each iteration, evaluate the function at the midpoint of the bracketed interval and discard half of the interval
- Repeat this process until we converge to the true solution up to some tolerance/precision.



Interval Bisection Method

```
#INPUT: Root brakcet [lo,hi]
#PRECONDITION: requires f(x) to be continuous
#PRECONDITION: computeSign(f(lo)) != computeSign(f(hi))
loSign = computeSign(f(lo))
while (hi-lo)/2 > threshold: # limit iterations to prevent infinite loop
    mid = (lo + hi)/2 # new midpoint
    midSign = computeSign(f(mid))
    if midsign == 0: # when mid is the root
        lo = hi = mid
    else if midSign == loSign:
        lo = mid
    else:
        hi = mid
return lo
```

$f(x) = x^3 - x - 2$



What is f(lo)?
What is f(hi)?
What is f(mid)?

This graph is not a true representation and is just for illustration

Newton's Method

- Requires computing the derivative f `(x) of the function f(x)
- Iteratively formula $x_{n+1} = x_n - f(x_n) / f`(x_n)$

1. Set n = 0 and make an initial guess x_0
2. Compute $x_{n+1} = x_n - f(x_n) / f`(x_n)$
3. If $|x_{n+1}/ x_n| <$ threshold
   - Return x_{n+1}
4. Else go to step 2

What is x_1?
What is x_2?

Example: compute square root of 300, i.e., $f(x) = x^2 – 300$
Or solve for x where $x^2 = 300$

f `(x) = 2x

Initial guess → $x_0 = 10$

$x_1 = x_0 - f(x_0)/f`(x_0) = 10 - (-200/20) = 20$

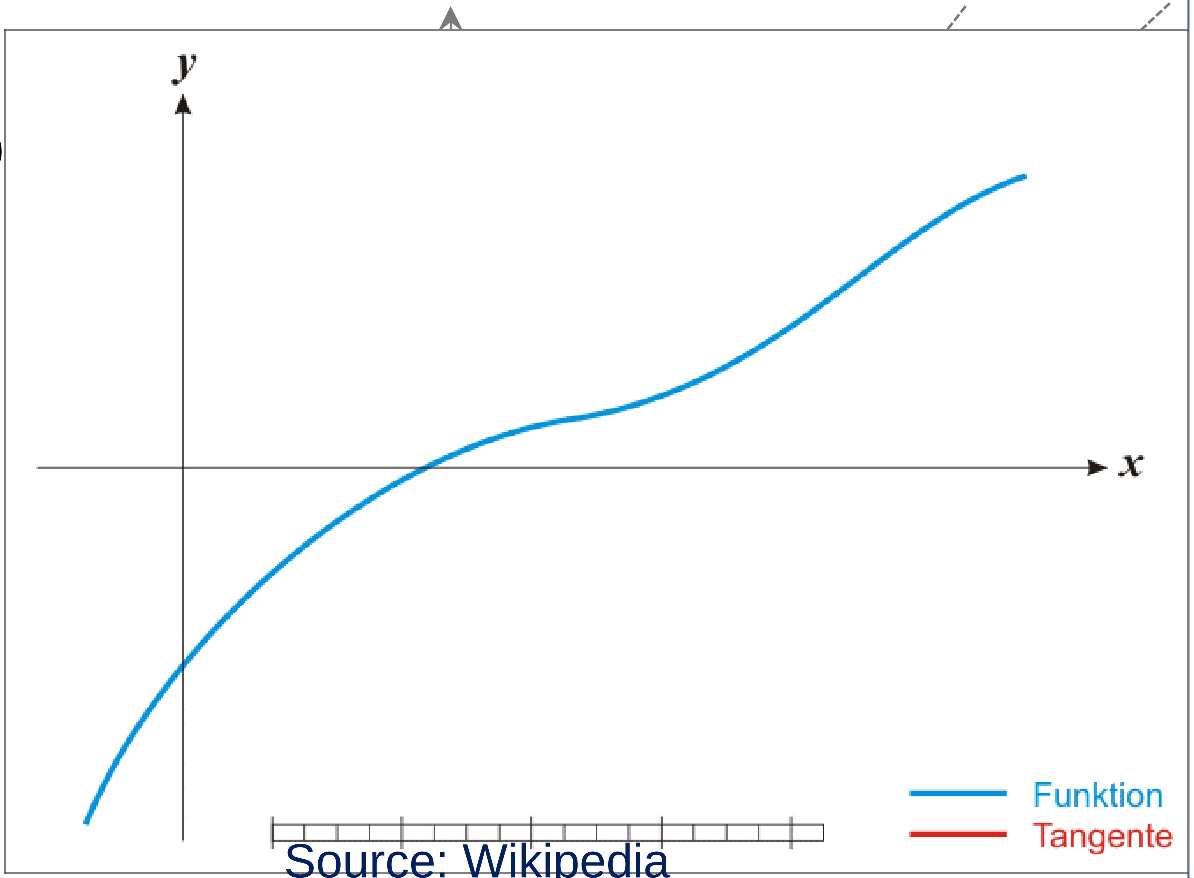
$x_2 = x_1 - f(x_1)/f`(x_1) = 20 – (100/40) = 17.5$

$x_3 = x_2 - f(x_2)/f`(x_2) = 17.5 – (6.25/35) = 17.321428571$

…

Intuition behind Newton's Method

- $h = x_0 - x_1$
- $\tan(\theta) = f(x_0)/h$
- Tangent of a curve f(x) at a point x_0 is $f`(x_0)$
- $f`(x_0) = f(x_0)/h$
- $h = f(x_0)/ f`(x_0) = x_0 - x_1$
- Hence, $x_1 = x_0 - f(x_0)/ f`(x_0)$



Source: Wikipedia

Limitations of Newton's Method

Although Newton's method is a very powerful technique and it usually converges on the root much faster than Interval Bisection Metho, it has the following limitations:

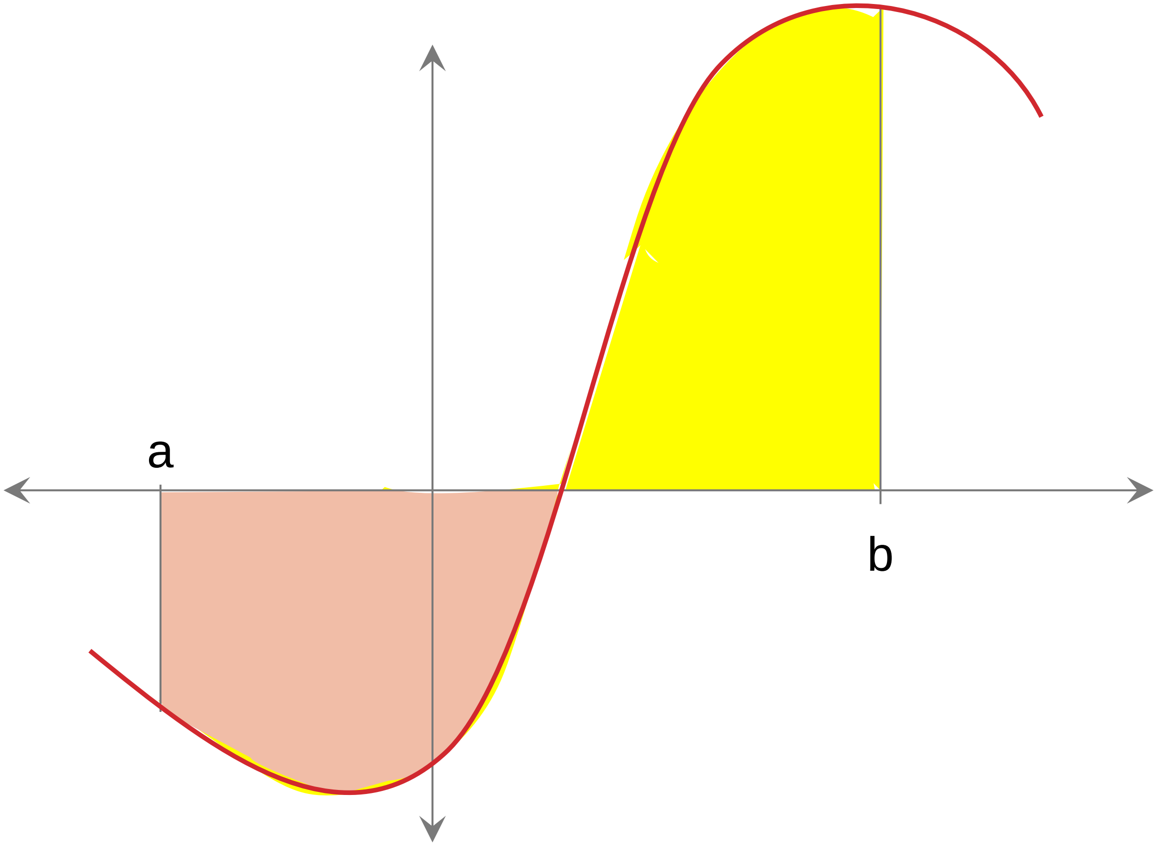
- Calculating the derivative may be difficult or expensive
- The derivative may be zero and the procedure may halt due to division by zero
- May fail to converge
  - E.g., if $f(x) = 1 - x^2$ and initial guess is 0

- Hence, although Newton's method is faster, interval bisection method is applicable to more scenarios

Calculating Integral

How to solve the numerical integral of the form

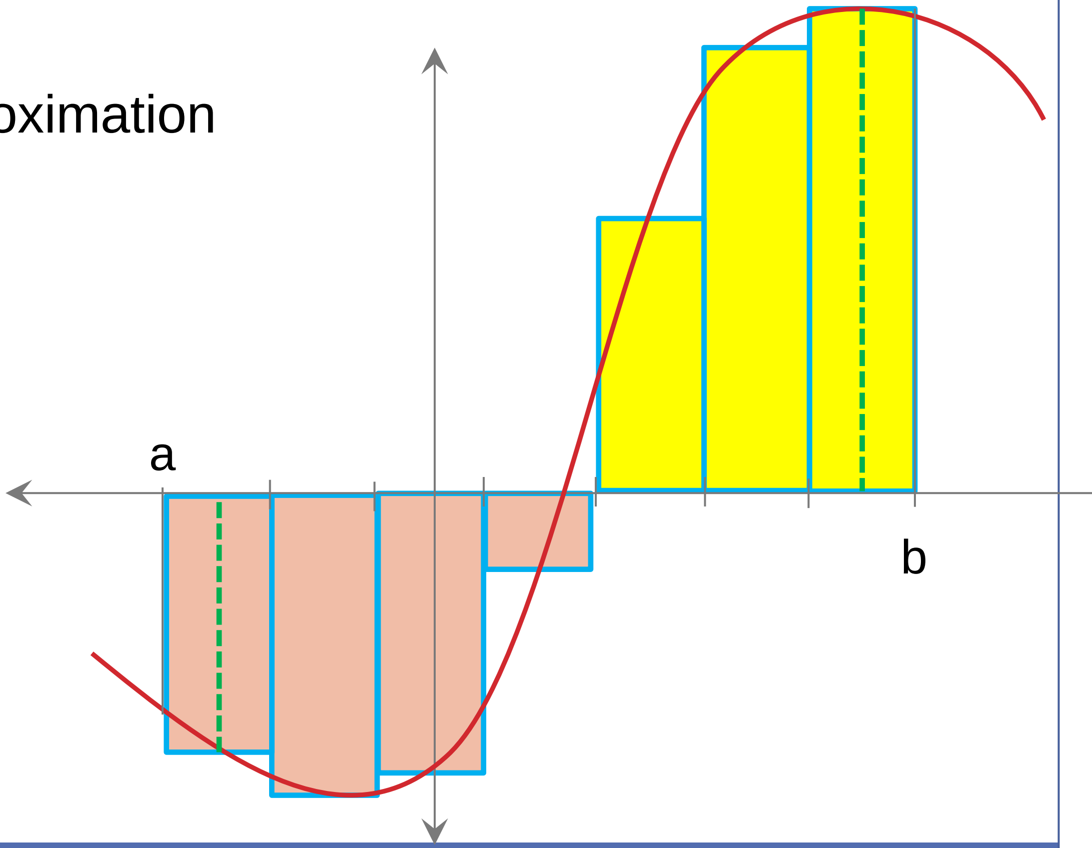
$$\int_a^b f(x)$$



Calculating Integral: Rectangle Rule

- Divide the range [a,b] in N equal width rectangles
- Let m be the mid point along the width of a rectangle
- Its height is then f(m)
  - The height can be negative (resulting in a negative area)
- Add the areas of all rectangles
- Larger N results in better approximation

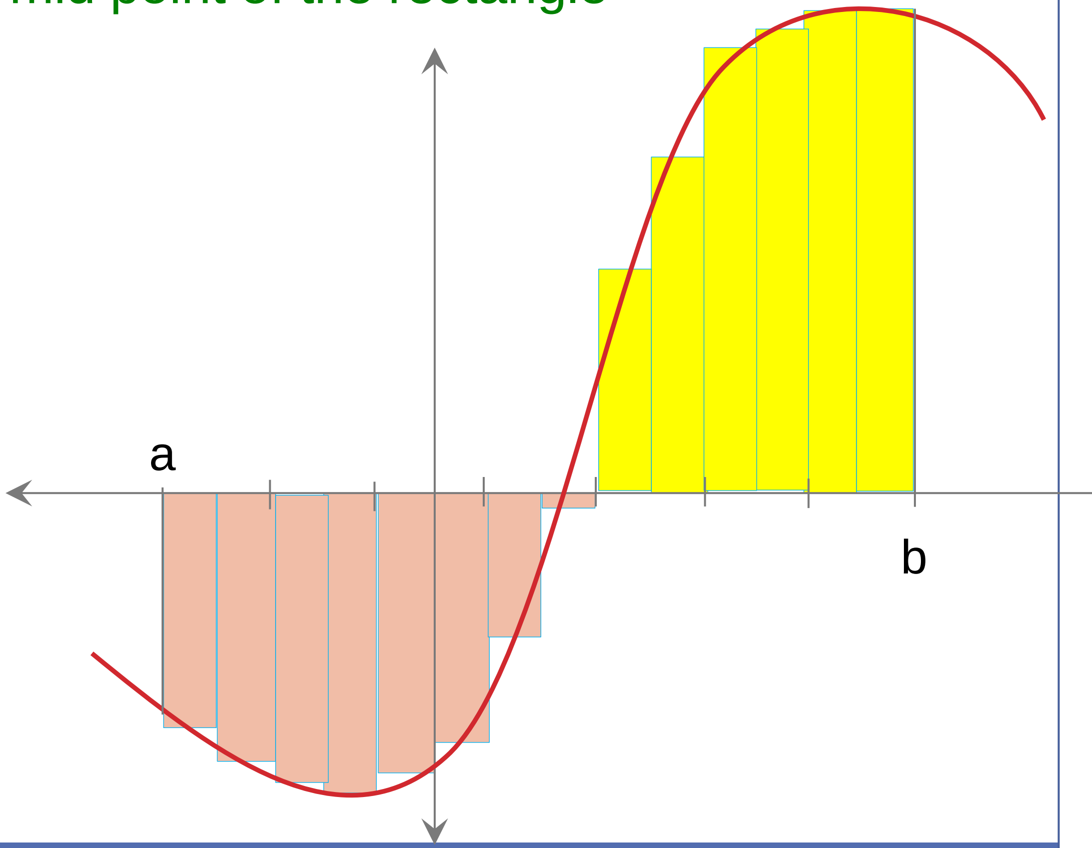
$$\int_a^b f(x)$$



Calculating Integral: Rectangle Rule

```
width = (b-a)/N; # divide into N rectangles of equal width
area = 0
for i=0 to N-1:
        mid = a + (i+0.5)*width #mid point of the rectangle
        height = f(mid)
        area += height*width
return area
```

$$\int_a^b f(x)$$



Calculating Integral: Trapezoidal Rule

- Use trapezoids instead of rectangles

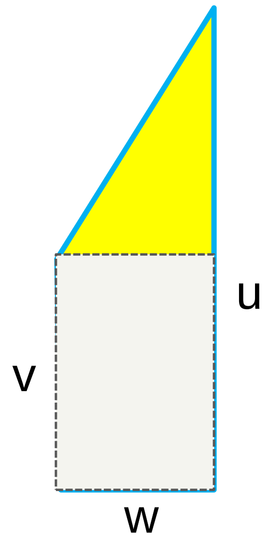
$$\int_a^b f(x)$$

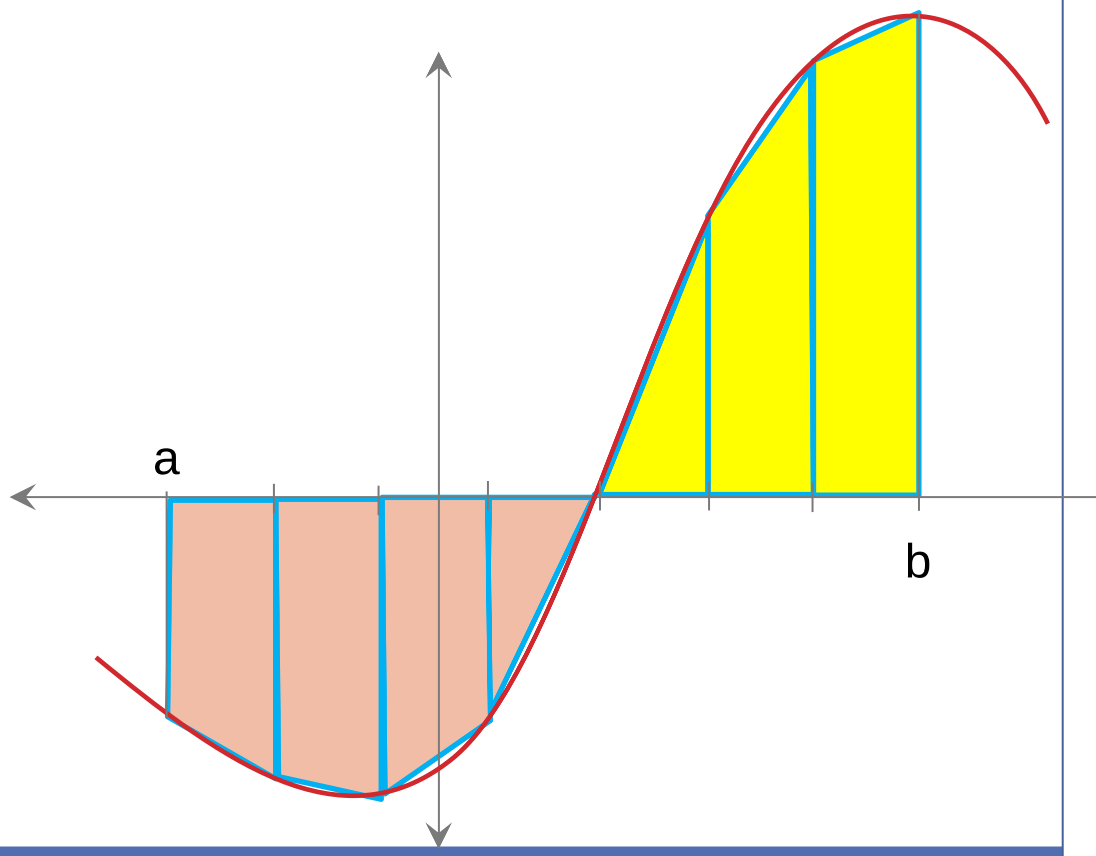
Area of a trapezoid with width w and side lengths u and v

= wv + 0.5w (u-v)

= wv + 0.5wu – 0.5wv

= 0.5wv + 0.5 wu = 0.5w(u+v)

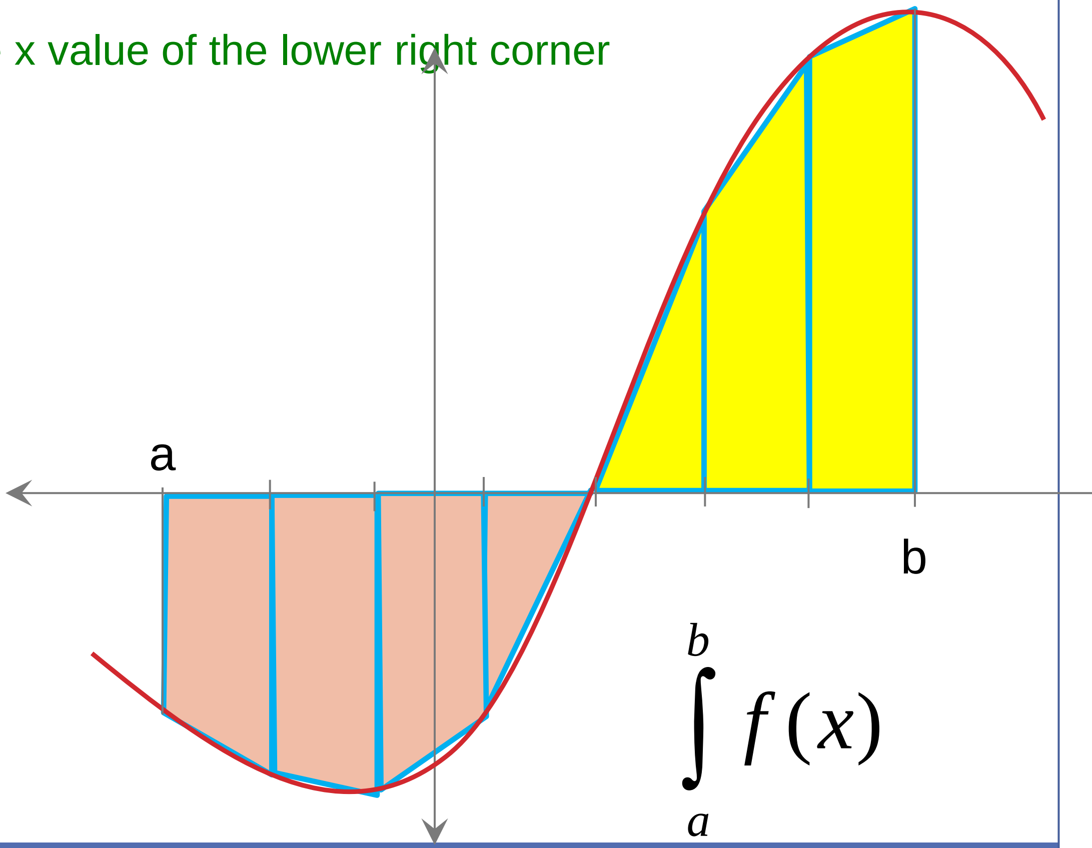




Calculating Integral: Trapezoidal Rule

```
width = (b-a)/N
area = 0
for i=0 to N-1:
        start = a + i*width # start is the x value of the lower left corner of the rectangle
        u = f(start)
        v = f(start + width) # start is the x value of the lower right corner
        area += 0.5*w*(u+v)

return area
```



$$\int_a^b f(x)$$

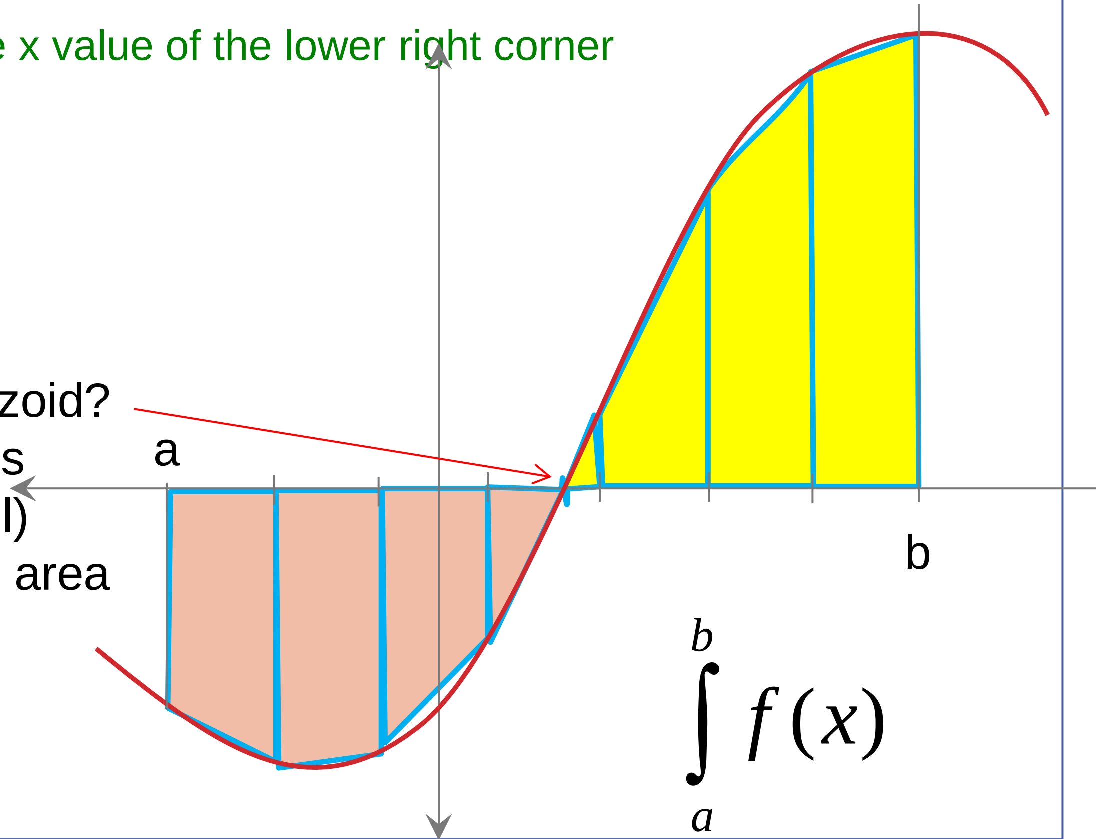
Calculating Integral: Trapezoidal Rule

```
width = (b-a)/N
area = 0
for i=0 to N-1:
        start = a + i*width # start is the x value of the lower left corner of the rectangle
        u = f(start)
        v = f(start + width) # start is the x value of the lower right corner
        area += 0.5*w*(u+v)

return area
```

What if the root lies between a trapezoid?

- Ignore – will add more error (if N is sufficiently large, the error is small)
- Create one trapezoid for negative area and another for positive area! (requires the root of the function)



$$\int_a^b f(x)$$

Summary

Content covered

- Directed Acyclic Graphs and algorithms for topological sort
- Numerical algorithms for computing root and integrals

Things to do (this list is not exhaustive)

- Make sure you understand topological sort and numerical algorithms
- Write programs for the numerical algorithms – these are very easy to implement and will significantly increase your understanding

Coming Up Next

- Primality Testing
- Recursion and Design Principles
- Format of the Final Exam and how/what to prepare