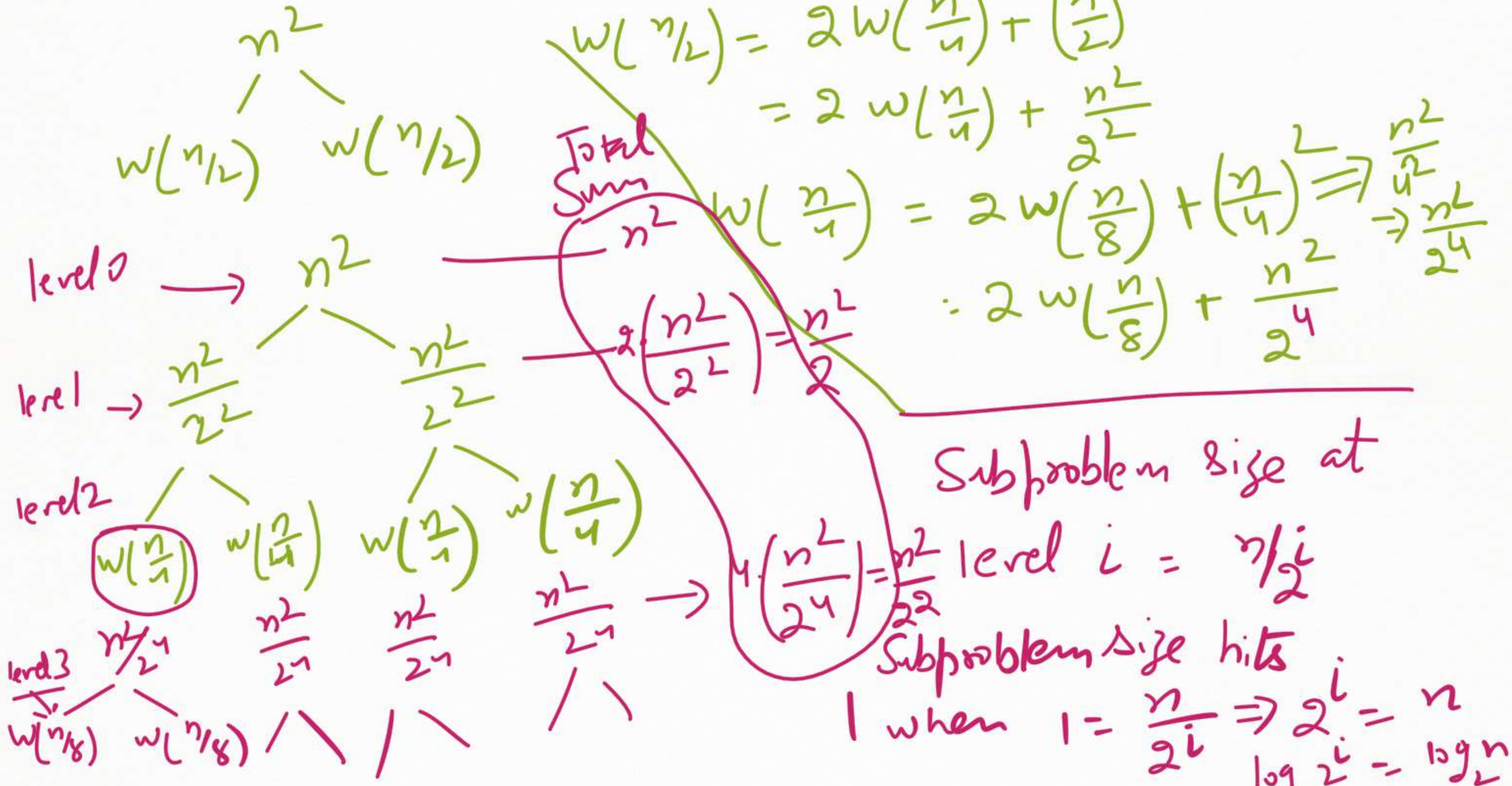
Theoretical Analysis of Algorithm

- Iterative
- Substitution
- Recursive Tree ⟷
- Master Theorem

Convert a recurrence into a tree
- Each node represent the cost incurred at various levels of recursion
- Sum up the costs of all the levels

$$W(n) = 2W(n/2) + n^2$$



$W(n/2) = 2W(\frac{n}{4}) + (\frac{n}{2})^2$

$= 2W(\frac{n}{4}) + \frac{n^2}{2^2}$

$W(\frac{n}{4}) = 2W(\frac{n}{8}) + (\frac{n}{4})^2 \Rightarrow \frac{n^2}{4^2} \Rightarrow \frac{n^2}{2^4}$

$\therefore 2W(\frac{n}{8}) + \frac{n^2}{2^4}$

Subproblem size at level $i = n/2^i$

Subproblem size hits 1 when $1 = \frac{n}{2^i} \Rightarrow 2^i = n$

$\log_2 2^i = \log_2 n \Rightarrow i = \log n$

Cost of the problem at level $i = \left(\frac{n}{2^i}\right)^2$

No. of nodes at level $i = 2^i$

level $0 = \left(\frac{n}{2^0}\right)^2 = n^2$

level $1 = \left(\frac{n}{2}\right)^2 = \frac{n^2}{2^2}$

level $2 = \left(\frac{n}{2^2}\right)^2 = \frac{n^2}{4^2}$

Total Cost $= \left(\sum_{i=0}^{\log n - 1} \frac{n^2}{2^i}\right) + 2^{\log_2 n}$

$W(1) = n^2 \sum_{i=0}^{\log n - 1} \left(\frac{1}{2}\right)^i + n$

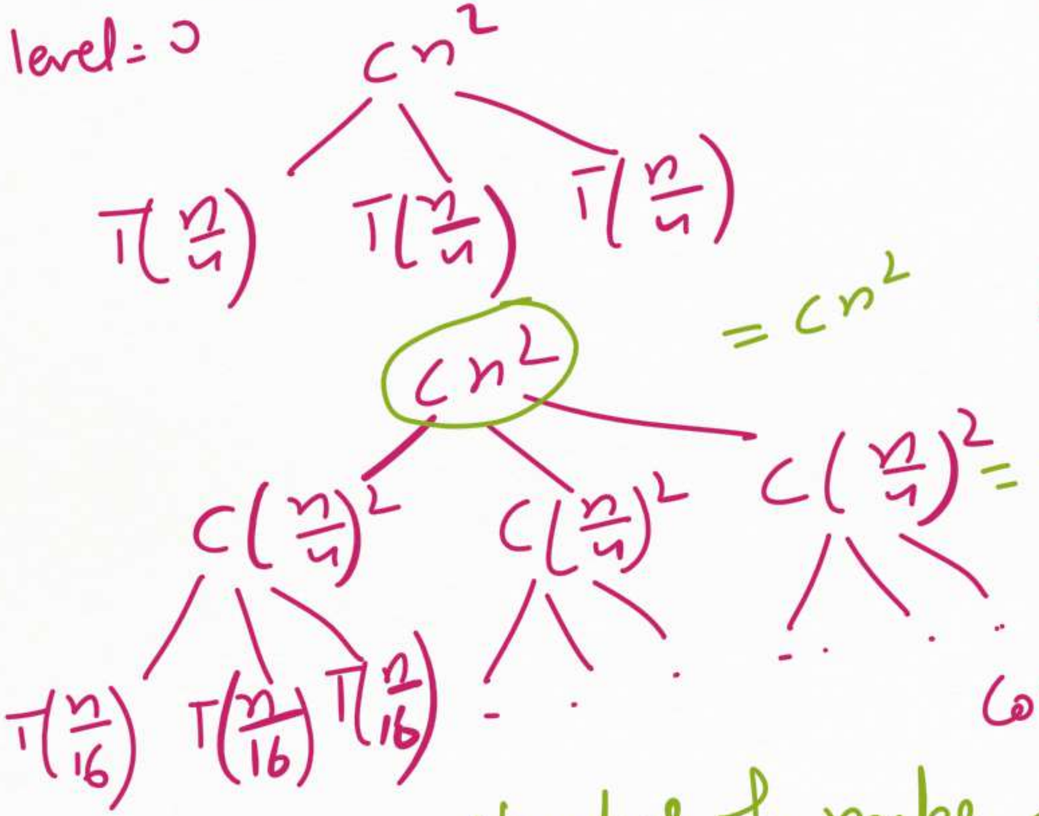
$\leq \frac{n^2 \cdot 1}{1 - 1/2} + O(n) \Longrightarrow 2n^2 + O(n)$

$O(n^2)$

$T(1) = 1$

$T(n) = 3T\left(\frac{n}{4}\right) + cn^2$

$T\left(\frac{n}{4}\right) = 3T\left(\frac{n}{16}\right) + c\left(\frac{n}{4}\right)^2$



Subproblem size at level $i = n/4^i$

Subproblem size hit $1 = \frac{n}{4^i}$

$\Rightarrow 4^i = n$

$\log_4 4^i = \log_4 n$

$i = \log_4 n$

Cost of a node at level $i = c\left(\frac{n}{4^i}\right)^2$

Number of nodes at level $i = 3^i$

last level has $3^{\log_4 n} \Rightarrow n^{\log_4 3}$ nodes

$$T(n) = \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + O\left(n^{\log_4 3}\right) + O(n)$$

$$\leq \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + O(n)$$

$$\frac{1}{1-\frac{3}{16}} \cdot cn^2 + O(n)$$

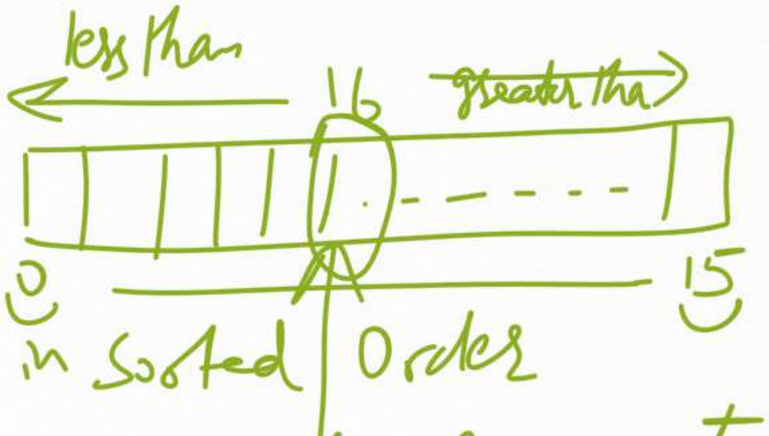
$$T(n) = O(n^2)$$

$$T(n) = T(n/3) + T(2n/3) + n$$

Home Assignment

Sub Problem Size

Binary Search



In the second call no of elements will be half.

$T(n) = T(n/2) + C$

$T(n/2) = T(\frac{n}{4}) + C$

$1 = \frac{n}{2^i}$

$2^i = n$

$i = \log n$

C

T(n/2)

C

C

T(n/4)

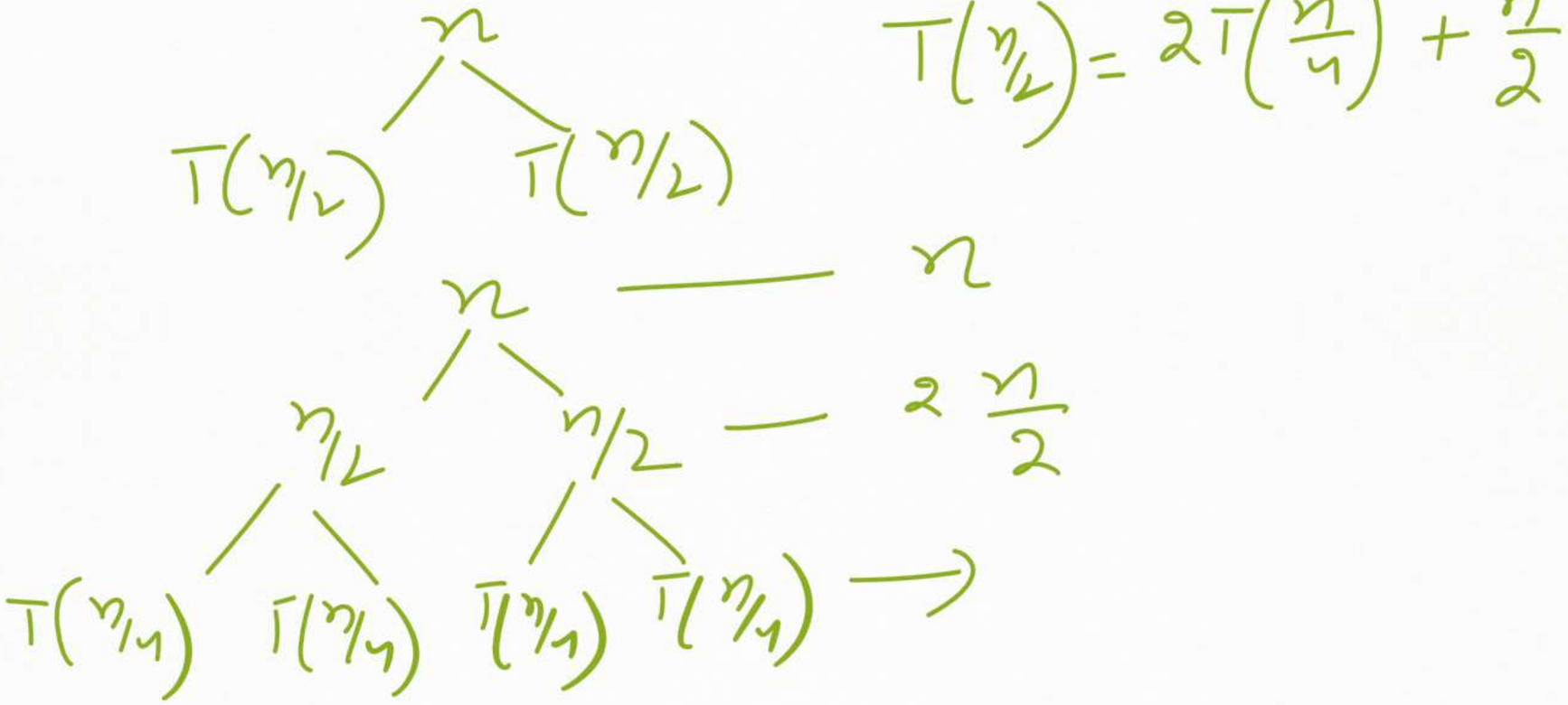
C + C + ... + C

log n

O(log n)

Merge Sort

$T(n) = 2T(n/2) + n$



$T(n/2) = 2T(\frac{n}{4}) + \frac{n}{2}$

The Master Method – a very simple technique for solving recurrences of the form

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is asymptotically positive function

where
- n is the problem size
- a is the number of subproblems in the recursion
- n/b is the size of each subproblem
- $f(n)$ is the cost of work done outside the recursive calls, which includes the cost of dividing the problem and cost of merging the solutions to the subproblems

There will be three cases :-

Case 1

If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then

$$T(n) = \Theta(n^{\log_b a})$$

Case 2

If $f(n) = \Theta(n^{\log_b a})$, then

$$T(n) = \Theta(n^{\log_b a} \log n)$$

Case 3

If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq c f(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

① $T(n) = 8T\left(\frac{n}{2}\right) + 100n^2$

$a = 8,\ b = 2,\ f(n) = 100n^2$, so

$\log_b a = \log_2 8 = \log_2 2^3 = 3\log_2 2 = 3$

$f(n) = 100n^2 = O(n^{3-\epsilon})$ for some constant $\epsilon > 0$

It belongs to case 1.

$T(n) = \Theta(n^{\log_b a}) = \Theta(n^3)$

② $T(n) = 2T\left(\frac{n}{2}\right) + 10n$

$a = 2, \; b = 2, \quad f(n) = 10n$

$\log_b a = \log_2 2 = 1$

$f(n) = 10n = \theta(n^{\log_b a}) = \theta(n)$

$T(n) = \theta(n^{\log_b a} \log n) = \theta(n \log n)$

③ $T(n) = 2T\left(\frac{n}{2}\right) + n^2$

$a = 2, \quad b = 2, \quad f(n) = n^2$

$\log_b a = \log_2 2 = 1$

$f(n) = n^2 = \Omega\left(n^{\log_b a + \epsilon}\right)$

$= \Omega\left(n^{1+\epsilon}\right)$

$\epsilon > 0$

$\epsilon = 1$

Case 3 conditions are satisfied

$T(n) = \Theta(f(n))$

$= \Theta(n^2)$

$n^2 = \Omega(n^2)$

$a f(n/b) \leq c f(n)$ for some constant $c < 1$

$2\left(\frac{n}{2}\right)^2 \leq cn^2 \rightarrow \frac{n^2}{2} \leq cn^2$

$0.5n^2 \leq cn^2$

1. $T(n) = 4T(n/2) + n$ $\quad a=4$

2. $T(n) = 4T(n/2) + n^2$ $\quad b=2$

3. $T(n) = 4T(n/2) + n^3$ $\quad \log_b a = 2$

Solution

1 → $f(n) = n = O(n^{2-\epsilon}) \quad \epsilon > 0$

$\epsilon = 1$

$T(n) = \theta(n^{\log_b a}) = \theta(n^2)$

2. → $f(n) = n^2 = \theta(n^{\log_b a}) = \theta(n^2 \log n)$

$f(n) = n^3 = \Omega(n^{2+\epsilon})$ $\epsilon > 0$

$T(n) = \theta(f(n)) = \theta(n^3)$

$a f(n/b) \leq c f(n)$

$4\left(\frac{n}{2}\right)^3 \leq c n^3$

$4 \times \frac{n^3}{2^3 \cdot 2}$ $\left(\frac{1}{2}\right) n^3 \leq c n^3$

$0.5 n^3 \leq c n^3$

$c < 1$

$T(n) = 2^n T(n/2) + n^n$

a is not a constant

Doubt

$T(n) = 2T(n/2) + \frac{n}{\log n}$ ⟹ non-polynomial diff between $f(n)$ and $n^{\log_b a}$

$T(n) = 0.5T(n/2) + n$

a is less than 1

$a \geq 1$

$T(n) = 4T(n/2) + n^2 \log n$

$T(n) = 4T(n/2) + n^2 \log n$ $\quad a = 4,\ b = 2$

$f(n) = n^2 \log n \neq O(n^{\log_b a - \epsilon})$ $\quad \log_b a = 2$

$= O(n^{2-\epsilon})$ $\quad \epsilon > 0$

$n^2 \log n = \Theta(n^2)$

$n^2 \log n \neq \Omega(n^{\log_b a + \epsilon})$

$n^2 \quad = \Omega(n^{2+\epsilon})$ $\quad \epsilon > 0$

for sufficiently large n

$$a f(n/b) \leq c f(n)$$

$$4.\left(\frac{n}{2}\right)^2 \lg\left(\frac{n}{2}\right) \leq c n^2 \lg n$$

$$4.\frac{n^2}{4} \lg\left(\frac{n}{2}\right) \leq c n^2 \lg n$$

$$\lg n - \lg 2 \leq c \lg n$$

$$\lg n - 1 \leq c \lg n \qquad c < 1$$

Divide and Conquer Method

- Binary Search
- Sorting < Merge Sort, Quick Sort

$O(N^3)$

Matrix Multiplication

2×2 Matrices

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$C_{11} = A_{11} B_{11} + A_{12} B_{21}$: 2 Mult + 1 Addition

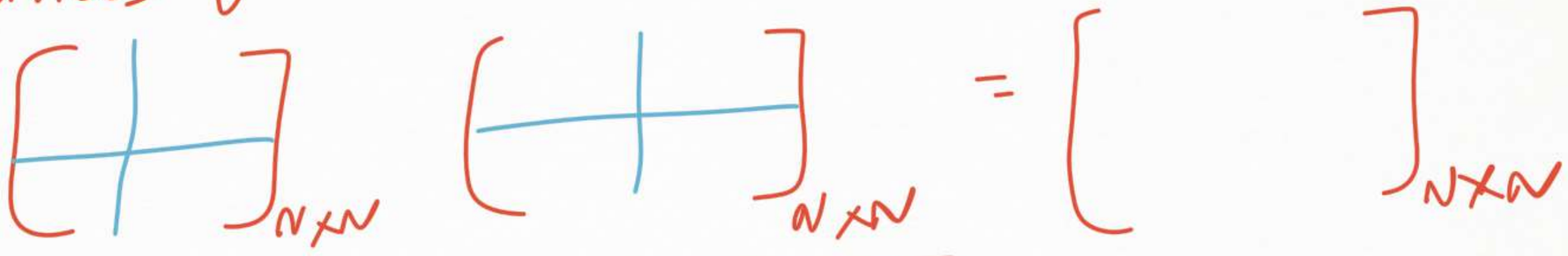
$C_{12} =$ ——— : 2 Mult + 1 Addition

$C_{21} =$ ——— : 2 Mult + 1 Addn

$C_{22} = A_{21} B_{12} + A_{22} B_{22}$: 2 Mult + 1 Addition

Overall → 8 Multiplications and 4 additions

Multiply Matrices of N×N



Divide and Conquer Strategy

Problem of Size N

Divide into Subproblem 1 of Size N/2

Get Solution of Subproblem 1

Divide into Subproblem 2 of Size 2

Get Solution of Subproblem 2

Combined → Solution to the Original problem

$$T(n) = \begin{cases} b & n \le 2 \\ 8T\left(\frac{n}{2}\right) + cn^2 & n > 2 \end{cases}$$

b & c are constants

n is a power of 2

A.B → 8 Multiplication and $n/2 \times n/2$

4 additions $n/2 \times n/2$

$O(n^2)$

cn^2

Master Theorem

$$aT\left(\frac{n}{b}\right) + f(n)$$

$a = 8, \; b = 2$

$f(n) = cn^2$

$\log_b a = \log_2 8 = 3$

$$f(n) = O(n^{\log_3 4 - \epsilon}) \qquad \epsilon > 0$$

$$= O(n^{3-\epsilon}) \qquad \epsilon > 0$$

$$\epsilon = 1$$

$cn^2 \quad O(n^2)$

$$T(n) = \Theta(n^3)$$

$O(N^3)$

```
for( i=0, i<N ; i++)
{  for( j=0 ; j<N ; j++)
     {   c[i][j]= 0 ;
         for( k=0; k<N; k++)
         { c[i][j]= c[i][j]+ A[i][k]*B[k][j] ;
         }
     }
}
```

Strassen's Matrix Multiplication

$$\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} = \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix}$$

$$m_1 = (a_{00} + a_{11}) * (b_{00} + b_{11})$$

$$m_2 = (a_{10} + a_{11}) * b_{00}$$

$$m_3 = a_{00} * (b_{01} - b_{11})$$

$$m_4 = a_{11} * (b_{10} - b_{00})$$

$$m_5 = (a_{00} + a_{01}) * b_{11}$$

Mult	2 Add
M	Add
M	Sub
M	Add
M	Add

$$m_6 = (a_{10} - a_{00}) * (b_{00} + b_{01})$$

$$m_7 = (a_{01} - a_{11}) * (b_{10} + b_{11})$$

$$C_{00} = m_1 + m_4 - m_5 + m_7$$

$$C_{01} = m_3 + m_5$$

$$C_{10} = m_2 + m_4$$

$$C_{11} = m_1 + m_3 - m_2 + m_6$$

1 Mult	2	Add/Sub
1 Mult	2	Add/Sub
7	10	Add
	3	Add/Sub
	1	
	1	
	3	Add
7 M	18	Add

$$T(n) = \begin{cases} b & n \le 2 \\ 7T\left(\frac{n}{2}\right) + an^2 & n > 2 \end{cases}$$

here a and b are constants

Master's Theorem

$a = 7, \; b = 2 \qquad f(n) = an^2$

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 7})$$

$$\approx \Theta(n^{2.81})$$

Algorithm: Strassen (A, B, n)

// Input: A and B are n×n matrices

// Output: C = A*B n×n matrix

// assumption is n is a power of 2

if n=1

return C = A*B

else

Partition $A = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix}$ and $B = \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$

M1 ← Strassen($A_{00}+A_{11}$, $B_{00}+B_{11}$, n/2)

M2 ← Strassen($A_{10}+A_{11}$, B_{00}, n/2)

!

$M7 \leftarrow \text{Strassen}(A_{01} - A_{11}, B_{10} + B_{11}, n/2)$

$C_{00} \leftarrow M_1 + M_4 - M_5 + M_7$

$C_{01} \leftarrow M_3 + M_5$

$C_{10} \leftarrow M_2 + M_4$

$C_{11} \leftarrow M_1 + M_3 - M_2 + M_6$

return $C = \begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix}$

Little bit better than Brute force $O(n^3)$ method.

Parallel Algorithms

Multiple processors will be doing computations (addition & multiplication) parallely. Because of this, time may be reduced.

Sequential Algorithm

At a moment only one Processing element is Active

processor (core)

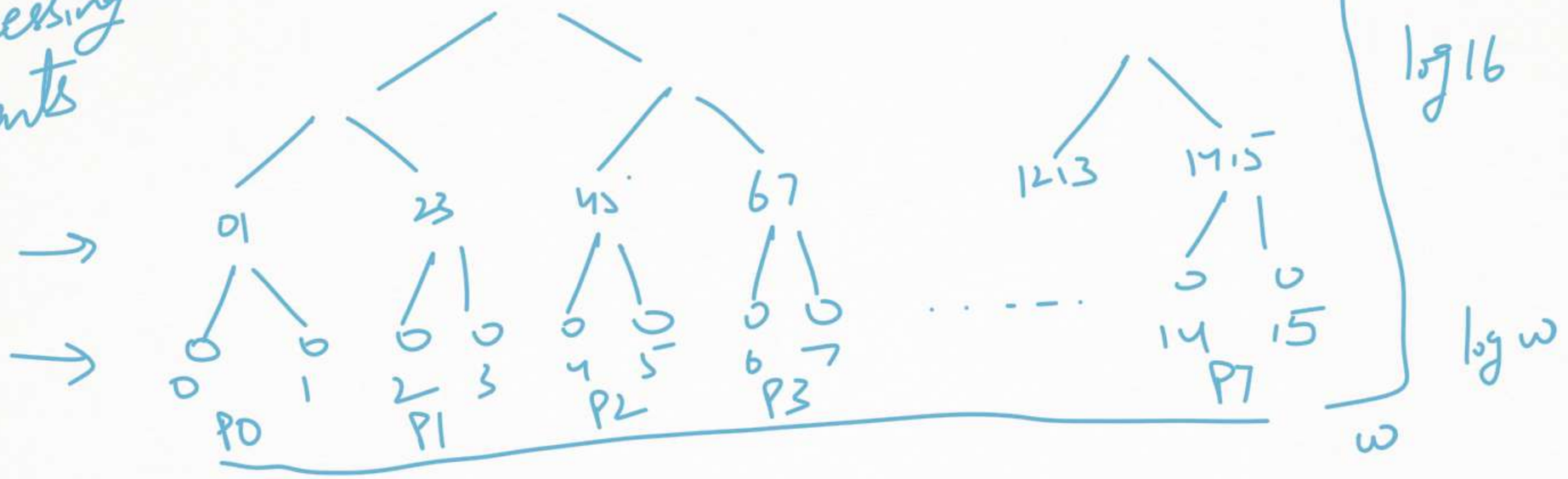
Today's Desktop / Laptop → these are multiprocessor Systems.

You have Quad Core 4 processors

(Sum / Max.) of n elements using 1 Processor] O(N)

Parallel Algorithm

8 Processing Elements



log n Steps → Sum of n elements

Work Time Complexity =

No. of Processor × Time

Seq. Algorithm = $1 \cdot O(N) = O(N)$

Parallel Algorithm = $O(N) \log N = O(n \log n)$

Not an optimal

⇓

Instead of $O(n)$ processors, we will take $O(N/\log N)$ processors →

$\left(\frac{N}{\log N}\right)$ → Compute their Sum Sequentially

Each processor will find the Sum of $(\log N)$ elements in Sequential method → in $\log N$ time

$\left(\frac{N}{\log N}\right)$ partial Sums } compute Sum in tree paradigm

$\log N$ + $\log\left(\frac{N}{\log N}\right)$ → To Sum these partial Sum terms

partial Sum terms

$\log N + \log N - \log\log N$

$= O(\log N)$

$\frac{16}{\log_2 16} = 4$

Work Time Complexity $\left(\frac{N}{\log N}\right) \log N = N$

Parallel Matrix Multiplication

Think About this

Greedy Algorithm

Simplest method to find solution of an optimization problem

- Decision is taken on the basis current available information without worrying about the effect of current decision in future
- Build a solution part by part, choosing the next part in such a way that it gives an immediate benefit.
- This approach never reconsiders the choices taken poerously

- Easy to implement and quite efficient also in most cases
- here we follow a heuristic to pick the local optimal choice at each step with the hope of finding the global optimal solution
- In many problems, it may not produce an optimal solution though it gives an near optimal solution in a reasonable time

Application → Shortest Path between two vertices using Dijkstras Algorithm
→ Minimum spanning Tree in a graph using Prim's / Kruskal's Algorithm.

Knapsack Problem

- Fractional Knapsack → Greedy Approach gives the solution
- 0-1 Knapsack → Greedy Approach Does not work

↳ Given a set of items, each with a weight and value, determine a subset of items to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible

Item	A	B	C	D
Profit	280	100	120	120
weight	40	10	20	24
$\left(\frac{p_i}{w_i}\right)$	7	10	6	5

Sort

	B	A	C	D
	100	280	120	120
weight	10	40	20	24
$\left(\frac{p_i}{w_i}\right)$	10	7	6	5

Knapsack $\Rightarrow W = 60$

Choose B $\quad 60 - 10 = 50$

Choose A $\quad 50 - 40 = 10$

capacity left in knapsack is 10 but weight of whole C is 20. So, I cannot pick whole of C instead a faction is to be taken

Problem Scenario: A thief is robbing a store and can carry a maximal weight of W into his knapsack.
There are n items available in the store and

weight of i^{th} item is w_i $\quad w_i > 0$

profit of i^{th} item is p_i $\quad p_i > 0$

what items should the thief take?

Fractional: thief may take only a fraction x_i of i^{th} item

$$0 \leq x_i \leq 1$$

x_i fraction of i^{th} item contributes $x_i \cdot w_i$ weight to the total weight in the knapsack and profit $x_i \cdot p_i$ to the total profit.

objective maximize $\sum_{i=1}^{n} (x_i \cdot p_i)$

subject to the constraint $\sum_{i=1}^{n} (x_i \, w_i) \leq W$

In this problem optimal solution can be obtained by

$$\sum_{i=1}^{n} (x_i \cdot w_i) = W$$

Compute $\left(\frac{p_i}{w_i}\right)$ sort in the decreasing

$\left(\frac{10}{20}\right)$ of C is chosen

Total weight = $10 + 40 + \frac{10}{20} \times 20 = 60 = W$

Total Profit = $100 + 280 + \left(\frac{10}{20}\right) \times 120$

$= 100 + 280 + 60 = 440$

Greedy Approach

w[i] → weight of i^{th} item

p[i] → profit of i^{th} item

Algorithm

```
for i = 1 to n
    x[i] = 0
weight = 0
for i = 1 to n
    if weight + w[i] ≤ W then
        x[i] = 1
        weight = weight + w[i]
    else
        x[i] = (W - weight) / w[i]
        weight = W
        break;
return x
```

Home Assignment

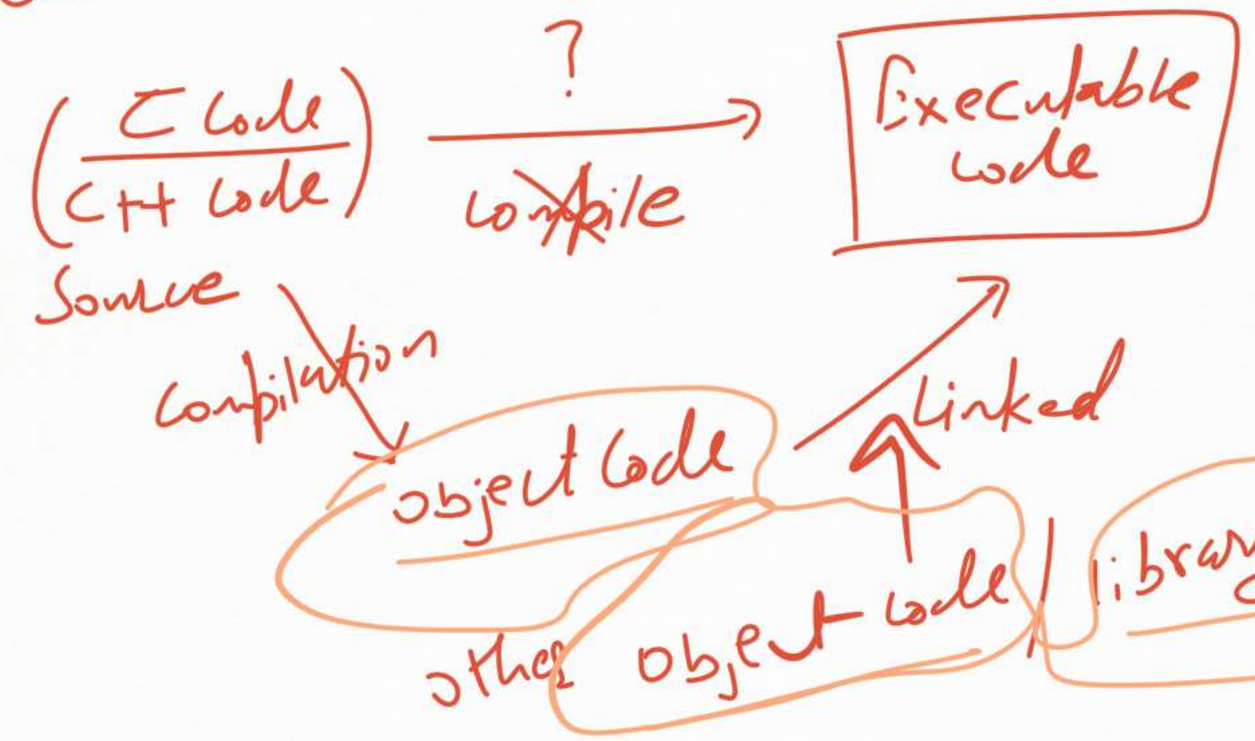
② Write a program to implement solution of Fractional Knapsack Problem.

① Implement Strassen's Matrix Multiplication Algorithm.

C/C++/Python/Java

Basic Programming concepts

object code Linking, Creation



gcc compiler

Code blocks

gcc hello.c } output file name would be a.out by default

Both compilation & linking }

gcc hello.c -o myout

-o flag to rename output file

Inplace of a.out, now the outfile file name would be myout.

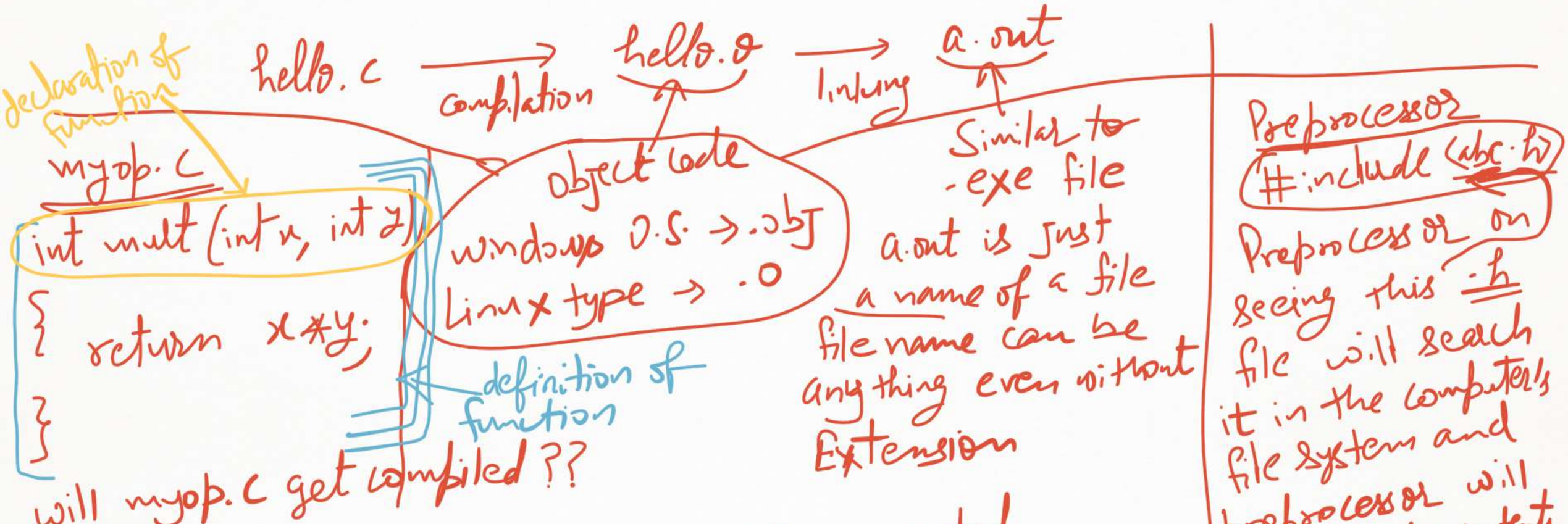
Debugging Flag gcc -g hello.c -o myout

Debugging information should be embedded in the generated executable file.

a.out

default name given to binary executable file when a C/C++ source file is compiled and linked successfully on a Unix like System

Linux
Mac OS


declaration of function
hello.c
compilation
hello.o
linking
a.out
myop.c
int mult (int x, int y)
{
return x*y;
}
object code
windows O.S. → .obj
Linux type → .o
definition of function
Similar to .exe file
a.out is just a name of a file
file name can be anything even without Extension
Preprocessor
#include <abc.h>
Preprocessor on seeing this .h file will search it in the computer's file system and preprocessor will copy all the contents of this header file at the place of this line
will myop.c get compiled ??
Yes
→ on compilation (myop.o / myop.obj) will be created
During compilation - mainly syntax checking is carried out
However, compilation does include preprocessor step

main.c

```
#include <stdio.h>
extern int mult(int x, int y);
int main( )
{  int x = 10;
   int y = 20;
   int z;
   z = mult(x,y);
   printf("The result is %d\n", z);
   return 0;
}
```

```
#include "myop.h"
```

myop.h

```
extern int mult(int x, int y);
```

Will main.c get compiled?

NO

- mult is not declared
- printf is not declared

.c/.cpp

→ Source code is never included

→ only .h is included.

→ .h files only contain function declaration

(definition & declaration are two different terms)

what is there in stdio.h for printf function?

→ only the declarations (functions / variables)

Then where is the definition of printf??

---

where is the definition of printf available and when is it used

in location of an executable file?

→ definition of library functions is available in Standard C library file.

The library file are used at the linking time to generate an executable by linking library files, User generated object code files

gcc main.c

perform compilation and linking both

→ compilation will be successful but linking stage will show error because definition of mult function is not available.

Proper procedure would be to generate object code files of all source code files and link them to get an executable

only compilation { gcc -c myop.c ⟹ myop.o
gcc -c main.c ⟹ main.o

✓ gcc myop.o main.o -o outfile

Linking of myop.o, main.o and library file to get executable outfile

- -c
- -g → Debugging
- -o output renaming
- -l library
- -L
- -I to specify header file name with complete path

-l → to specify library file name
-L → to specify the complete path of the directory where library file is residing

what is Library file

Just a collection of object code file

Two types — Static Library
Dynamic Linkable Library

Library File Extension

Linux	Windows (not known to me)
.a	.dll
.so	

.so → Shared Object
.a → archive (collection)

Example

Math Library File Name

Linux → libm.a, libm.so } containing definition of math library function

Header file to use math library function → math.h → containing declaration of math library function

gcc -lm

used to specify that please link math library file

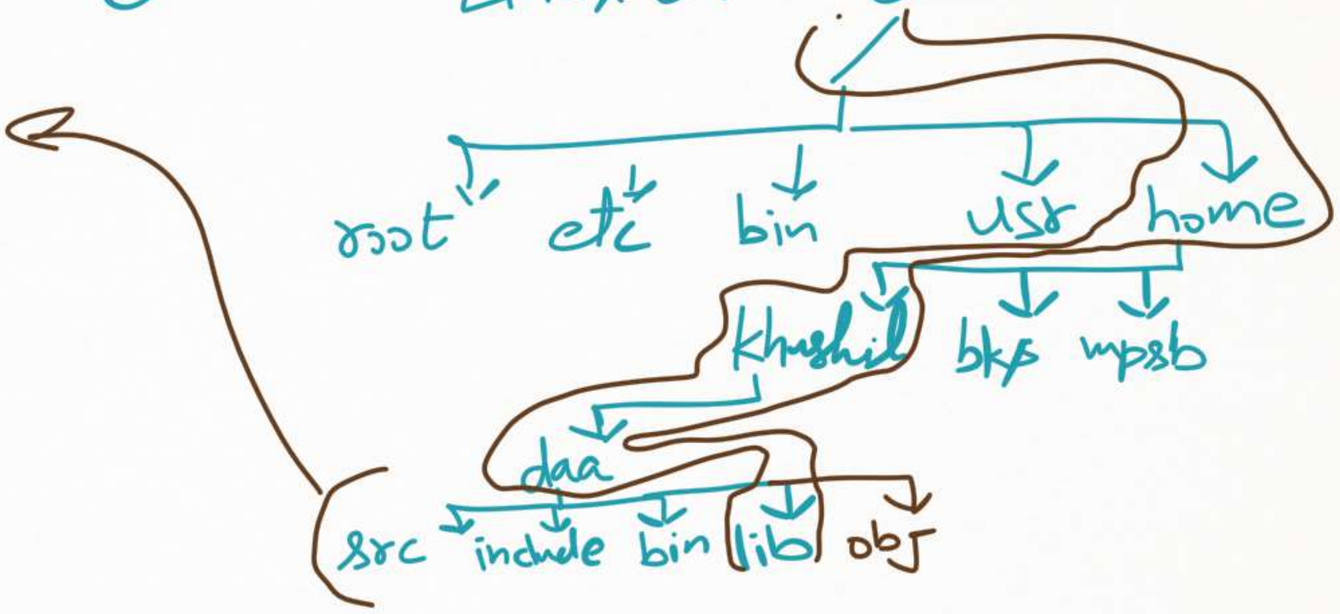
However path is not required to be specified because it is normally present in Standard library path

Hierarchy of Directory

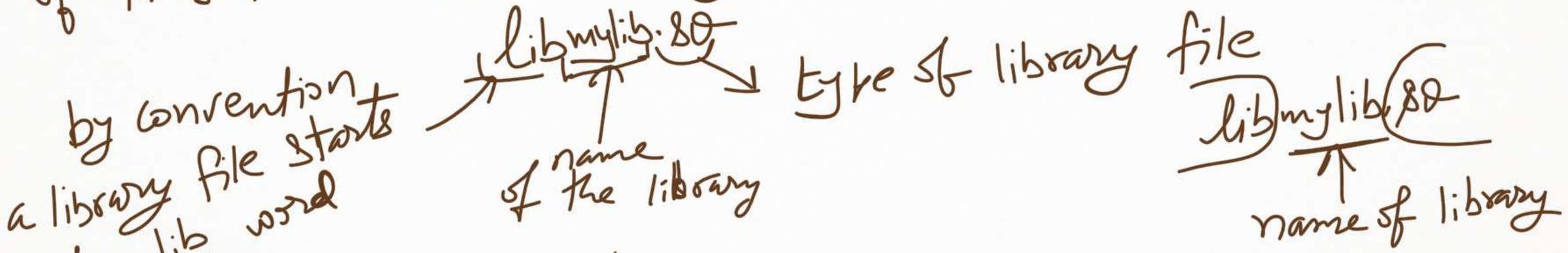
Example :- user created library file

src → Source code files
include → header files
bin → executable files
lib → library files
obj → object code

Linux directory Tree



Say lib directory in daa directory in home directory of khushil containg following library file



I want to use it

gcc -lmylib -L/home/khushil/daa/lib

name of library to be used.

Directory path which containg the library file to be used

gcc → Gnu C/C++ Compiler
g++

Debugging :-

gdb will be used for debugging || gdb → Gnu Debugger

For this, executable must be created using -g flag

```
gcc -g hello.c -o myhello
```

(get loaded)

myhello executable will run within Gnu Debugger gdb; prompt will change to gdb prompt

```
$ gdb myhello
# gdb myhello
```

prompt symbols

enter key

```
gdb>
```

we can issue commands for debugging

gdb> run ↵ { r ↵

run → is the command to start executing the executable file inside the gdb

There was some pointer related error →

program is trying to Access NULL pointer

⇓

This will lead to a situation known as "Segmentation Fault"

[Actually, the program receives a signal (software interrupt) named SIGSEGV]

we need debuggers to find out the exact location where we are by mistake accessing the NULL pointer.

gdb > break } b —

gdb > break
        ↑
    line number

Say there are 1000 lines in a program
b 200 :- I have placed a breakpoint at line number 200.

As the gdb tries to execute code at line number 200, there is a breakpoint, so execution of the program will suspend at that line

gdb) break
        ↑
    function name

The program execution will suspend at the entry point of the function

gdb > r ↵ } The program will run without any interruption till first breakpoint w.r.t main function

gdb > step // s ↵ it goes inside the function also while executing.

gdb > next // n ↵ > it executes a program line by line i.e. executes a function call in one go.

> print // p

To print the value of a variable

> quit // q

quit

> backtrace // bt

tells the sequence of functions that have been called.

```
int n=10;
int *ptr;
ptr = &n;
```

gdb > help

gdb > help print

Debugging → to handle Bugs encountered at RunTime

(maybe for → any logical error)

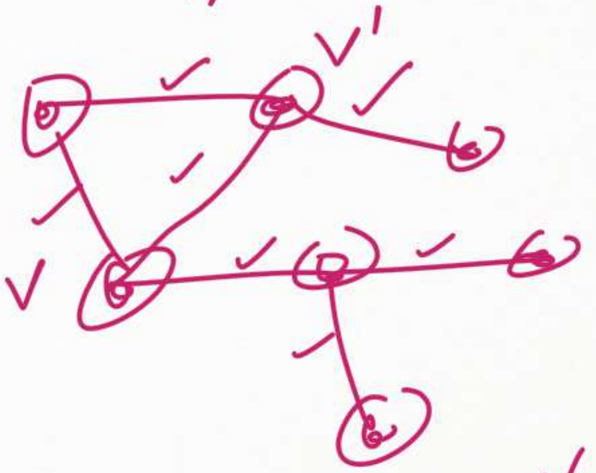
Home Assignment →

1) (create a program that might generate Segmentation Fault Problem.) (just one example case)
2) Try to debug it in Codeblocks/gdb

Study Greedy Algorithm Approach

Graph :- Formally, a Graph consists of two parts $G(V, E)$
- Set of vertices
- Set of edges E

Some



- Undirected Graph → edges don't have any directions
- Directed Graph → edges are directional

Undirected graph (v, v') and (v', v) are the same edge

Directed Graph (v, v') is an edge from v to v'
- does not guarantee that (v', v) is also an edge

Assumptions :- Finite set of vertices

Say V has n vertices → 1, 2, ... n

- Each edge is now a pair of number (i, j)

$1 \le i, j \le n$

Graph Representation → Adjacency Matrix

↘ Adjacency List

Adjacency matrix A(i, j) = 1 if (i, j) is an edge

else 0

A is an n x n matrix

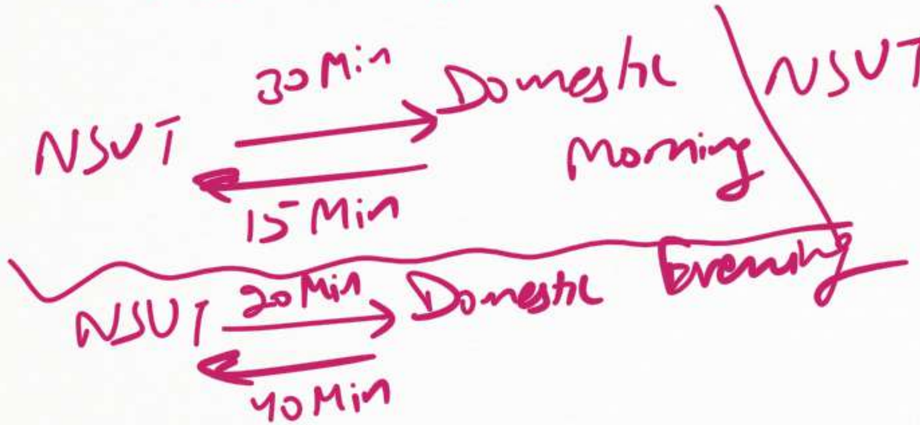
In case of Undirected graph, the Adjacency Matrix would be symmetric w.r.t the diagonal from top left to bottom right

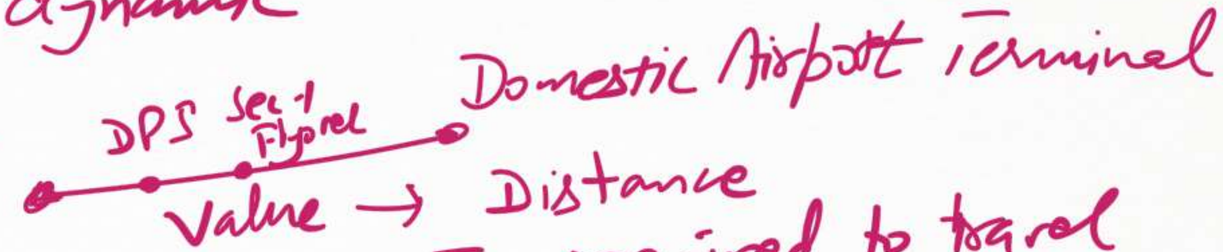
Undirected entry (4,5) will be same as (5,4)

Will not be true for Directed Graph

Weighted graph → Each connecting edge will have a weight/cost/ value associated with it.

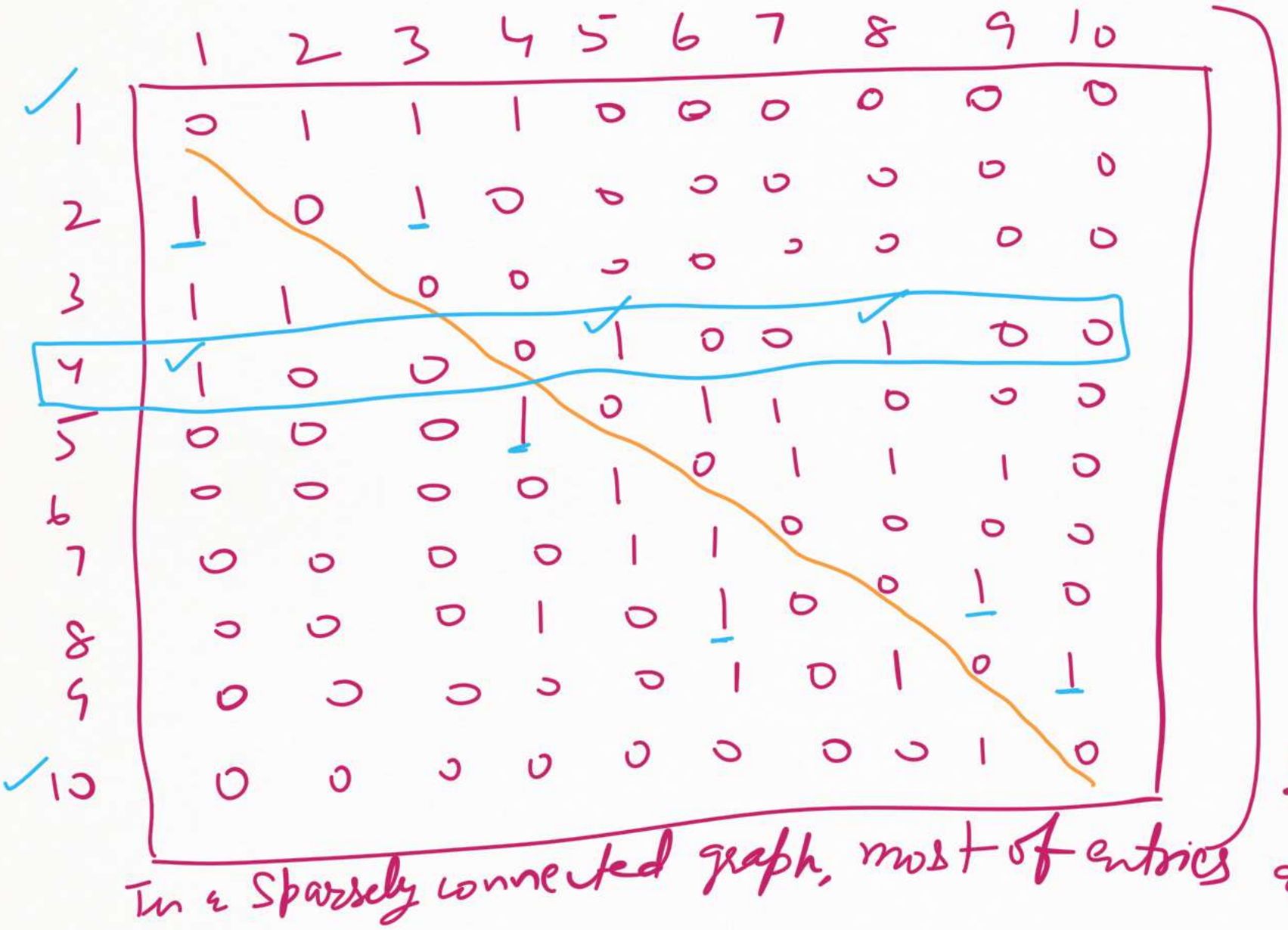
This weight may be dynamic





Value → Distance

→ Time required to travel



In a Sparsely connected graph, most of entries are Zeros.

Adjacency Matrix

undirected Graph

Presence of a link = 1

No link between two vertices = 0

For n vertices Size of matrix is n x n

Neighbours of i

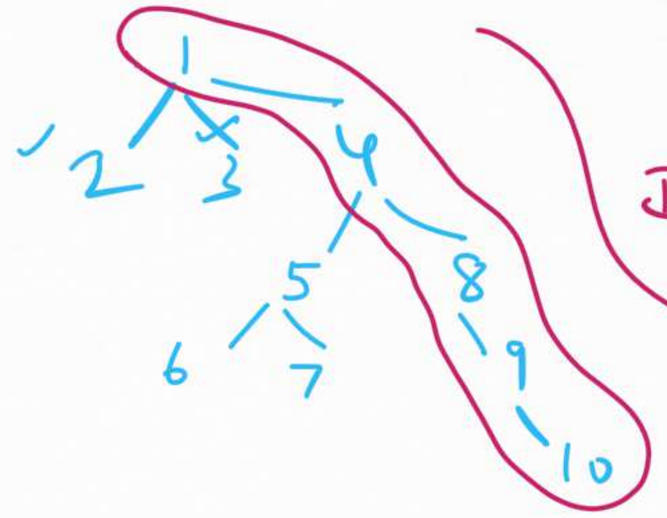
In row i → any column j with entry 1

Scan row i from left to right to identify all neighbours

example neighbours of 4 are {1, 5, 8}

Finding a path from one vertex (src vertex) to another (dst vertex) :- We will find neighbours of source vertex then neighbours of neighbours and so on.

Path from 1 to 10



Graph Traversal Problem

- Depth First
- Breadth First

Breadth First

→ Mark vertices that have been visited

→ Keep track of vertices whose neighbours have already been explored

→ Avoid going round indefinitely in loops/cycles

Adjacency List → typically it requires less space

→ Finding all the neighbours will require less time] List is better data structure.

→ Problem is Is j a neighbour of i?

in Adj. Matrix just check one entry A[i][j] = 1 ?

but in list, scan all neighbours of i in the Adj-List.] Matrix is a better representation

Greedy Algorithm for processing of Graph

Minimum Spanning Tree

- Minimum connectivity: no loops

Relation

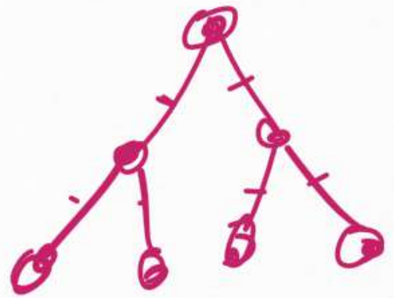
Tree ~ Graph

→ Connected acyclic graph - Tree

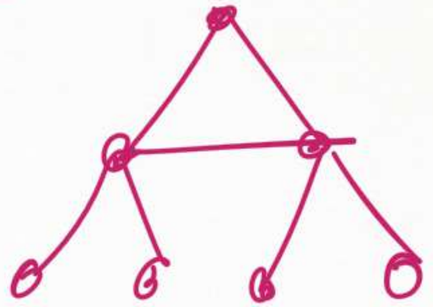
→ A tree of n vertices has exactly (n-1) edges

→ Adding an edge to a tree must create a cycle

- In a tree, every pair of nodes is connected by a unique path.



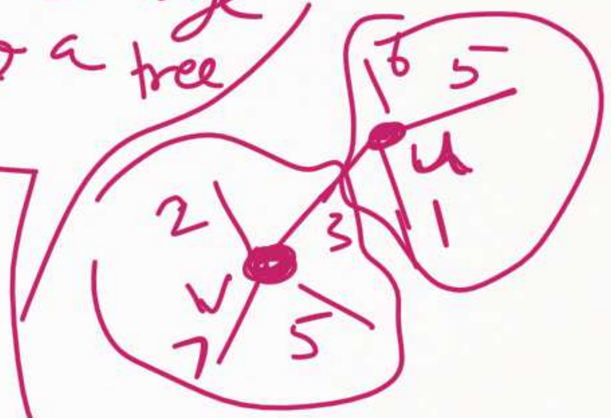
Tree



Graph

M.S.T. → Prim's Algorithm → Start with smallest edge and grow it into a tree

↘ Kruskal's Algorithm → scan edges in ascending order of cost and connect these components to form a tree

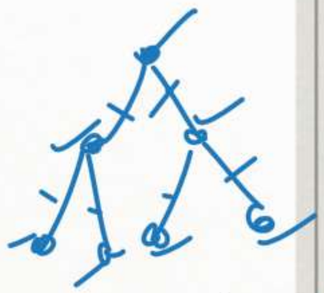


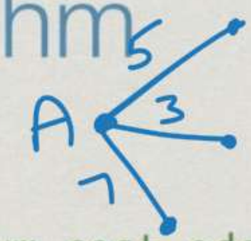
Prim's Algorithm → Build a MST by picking edges using a Greedy Approach (weight Edge)

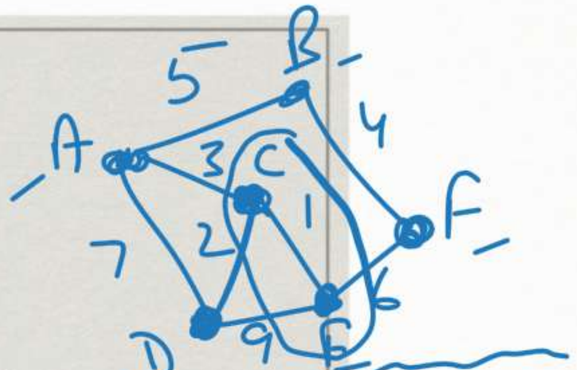
We keep on adding edge to grow a tree. We will start with a tree of just one edge and keep on adding edge to grow this tree further

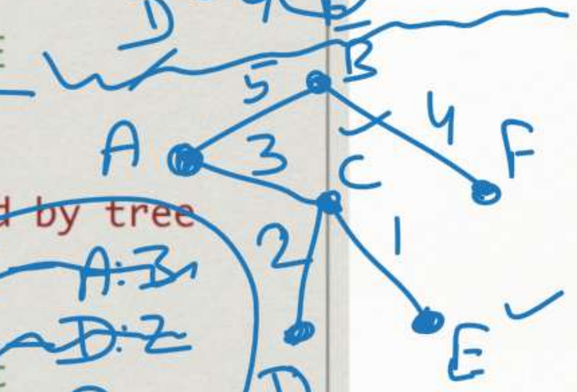
Prim's algorithm

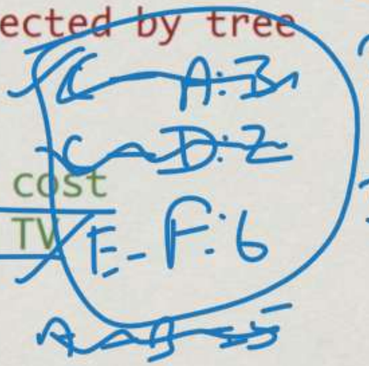
```
algorithm Prim_V1

Let e = (i,j) be minimum cost edge in E

TE = [e]      //List of edges in tree
TV = [i,j]    //List of vertices connected by tree

for i = 3 to n
   choose edge f = (u,v) of minimum cost
     such that u in TV and v not in TV
   TE.append(f)
   TV.append(v)

return(TE)
```











Further observations

- Need not start with smallest edge overall
  - For any vertex v, smallest edge attached to v must be in the minimum cost spanning tree
    - Consider the partition {v}, V-{v}
  - Can start with any such edge

Prim's algorithm revisited

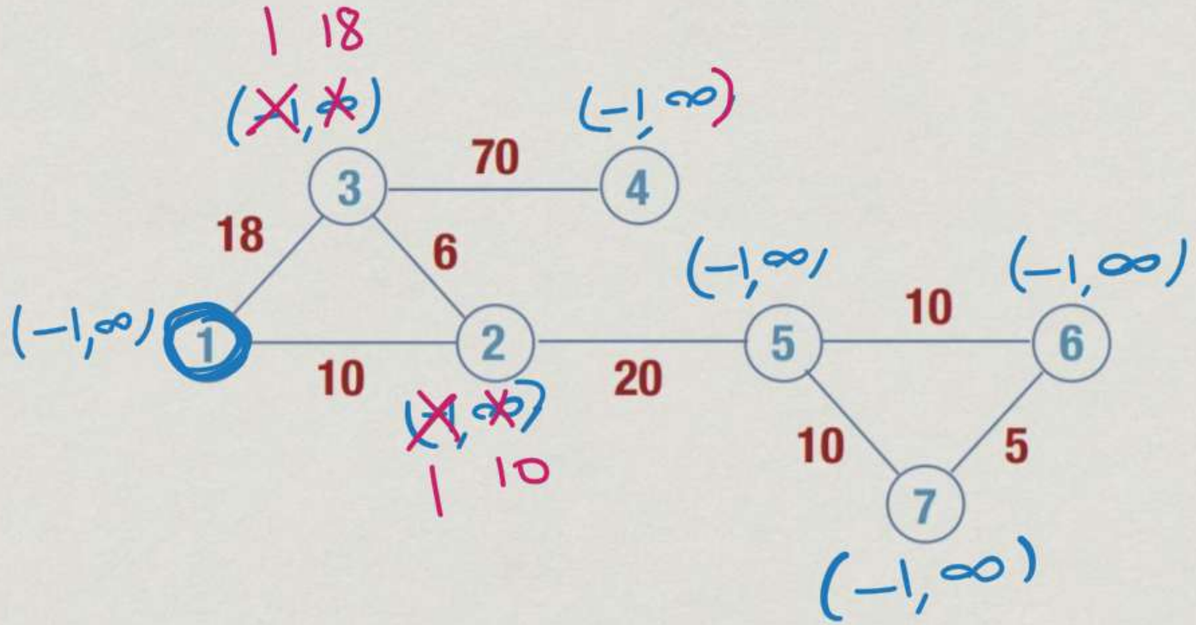
- Start with TV = {s} for any vertex s
- For each vertex v outside TV, maintain
  - Distance_TV(v), smallest edge weight from v to TV
  - Neighbour_TV(v), nearest neighbour of v in TV
- At each stage, add to TV vertex u with smallest Distance_TV(u)
  - Update Distance_TV(v), Neighbour_TV(v) for each neighbour of u
- Very similar to Dijkstra's algorithm!

Shortest Path

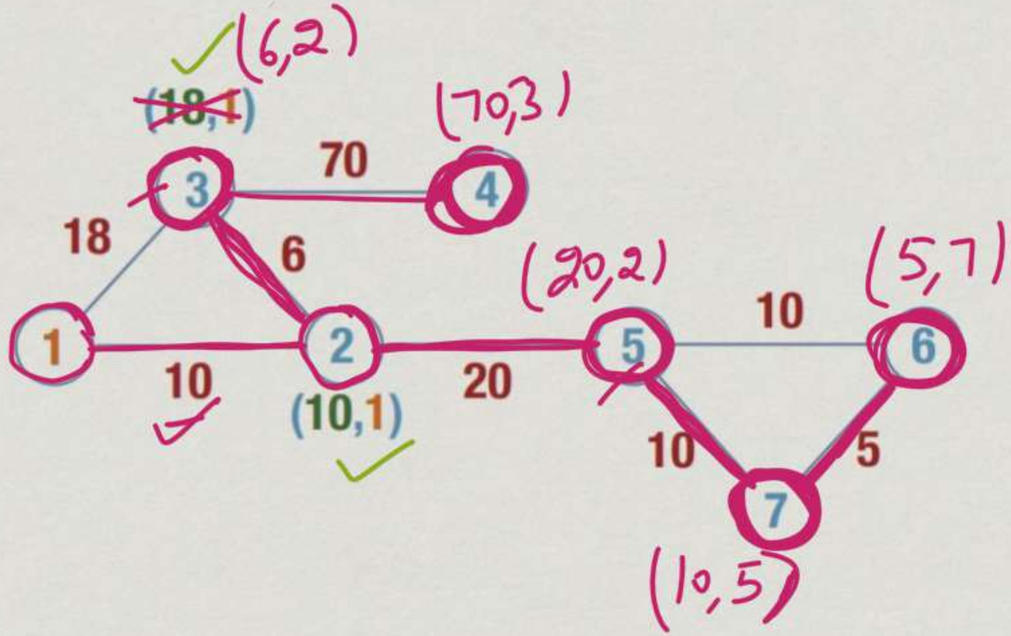
Prim's algorithm, refined

```
function Prim
  for i = 1 to n
    visited[i] = False; Nbr_TV[i] = -1; Dist_TV[i] = infinity

  TE = []   //List of spanning tree edges
  visited[1] = True
  for each edge (1,j)
    Nbr_TV[j] = 1; Dist_TV[j] = weight(1,j)

  for i = 2 to n
    Choose u such that Visited[u] == False
                       and Dist_TV[u] is minimum
    Visited[u] = True
    TE.append{(u,Nbr_TV[u])}
    for each edge (u,v) with Visited[v] == False
      if Dist_TV[v] > weight(u,v)
        Dist_TV[v] = weight(u,v); Nbr_TV[i] = u
```

Prim's algorithm



Prim's algorithm



Prim's

1, 2, 3

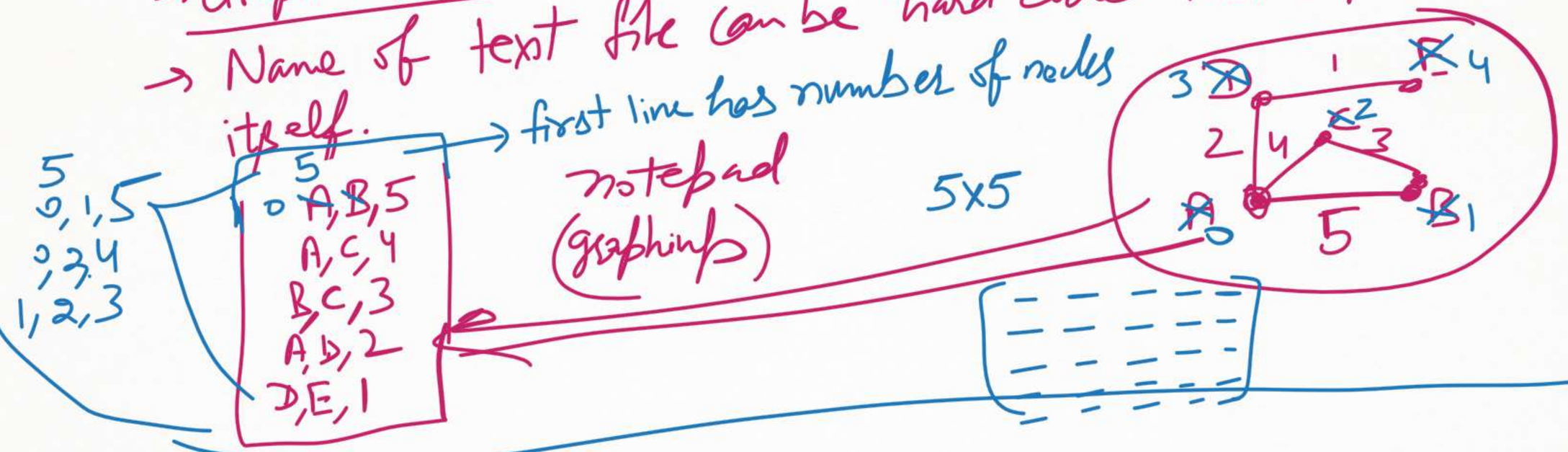
STL

Home Assignment →

Develop C/C++/Java/Python code for the Prim's Algorithm for some graph.

→ Graph Input will be read from a text file

→ Name of text file can be hard coded in the program itself.



Kruskal's Algorithm

Earlier we studied Prim's Method / Greedy Approach

we picked edges based on minimum weight criterion w.r.t. a vertex selected. The edges are picked in such a manner that new edges is connected to the already growing up tree.

→ gradually expands the edge into a tree

Kruskal Algorithm

Spanning tree

- Weighted undirected graph, G = (V,E,w)
  - Assume G is connected
- Identify a spanning tree with minimum weight
  - Tree connecting all vertices in V
- **Strategy 2:**
  - Order edges in ascending order by weight
  - Keep adding edges to combine components

N vertices
N-1 edges

no. of edges = m
no of vertices = n

1st step - Sort all the edges by weight. It keeps on trying edges from smallest to largest weight and for every edge, it can add to set of edges that will form a MST finally, only if, tree property is not violated (no cycle formation)

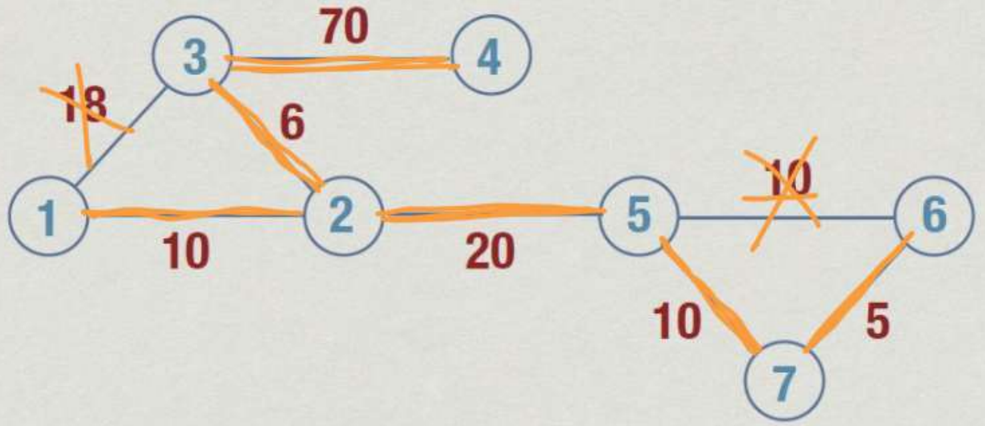
Kruskal's algorithm

```
algorithm Kruskal_V1

Let E = [e1,e2,…,em] be edges sorted by weight

TE = [] // List of edges added so far
i = 1   // Index of edge to try next

while TE.length() < n-1 //n-1 edges form a tree
  if adding E[i] to TE does not form a cycle
    TE.append(E[i])
    i = i+1
```

1 to m

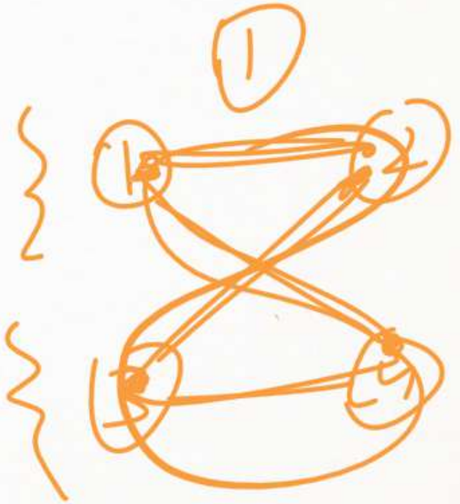
how to ensure that cycle is not formed??

Kruskal's algorithm



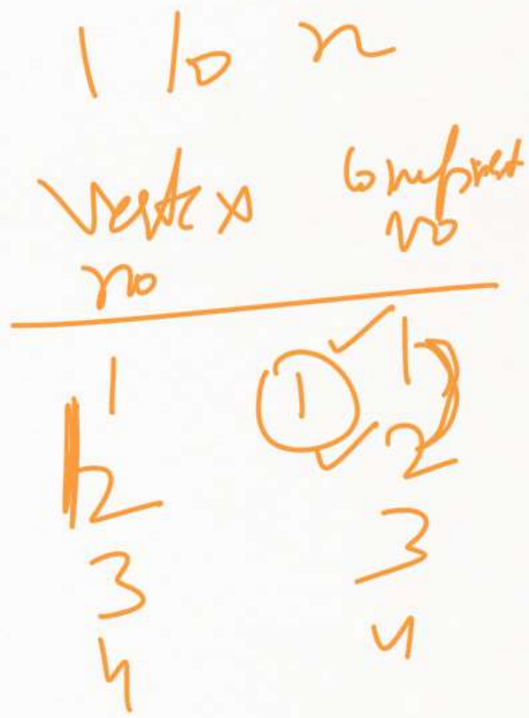
Correctness of Kruskal's algorithm …

- Unlike Prim's algorithm, at intermediate stages TE is not a tree
- Edges in TE partition vertices into connected components
  - Initially, each vertex is a separate component
  - Adding e = (u,v) merges components of u and v
    - If u and v are already in same component, e forms a cycle, hence discarded



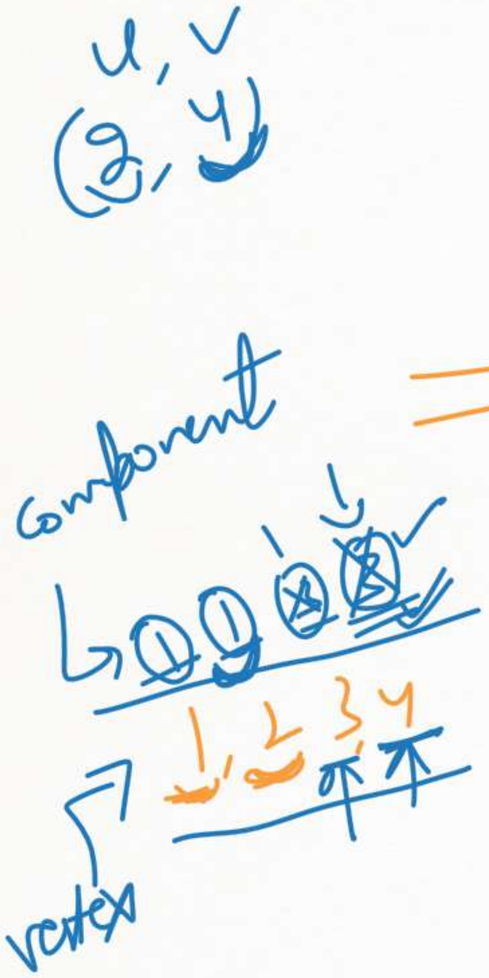
Kruskal's algorithm revisited

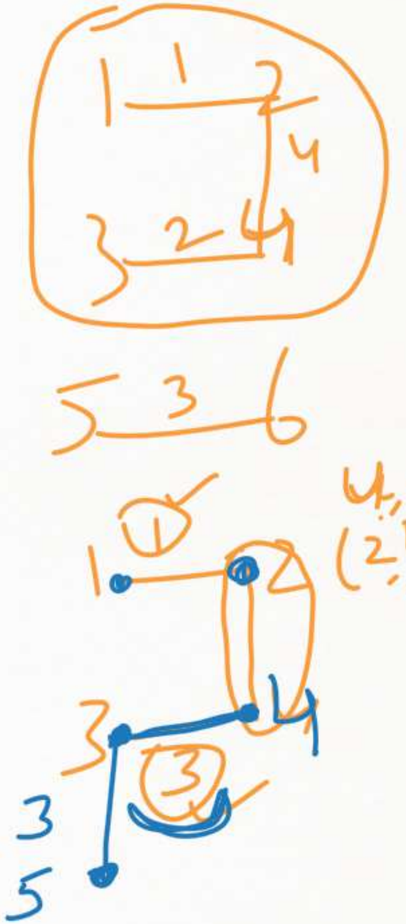
- To check if e = (u,v) forms a cycle, keep track of components
- Initially, Component[i] = i for each vertex i
- e = (u,v) can be added if Component[u] is different from Component[v]
  - Merge the two components

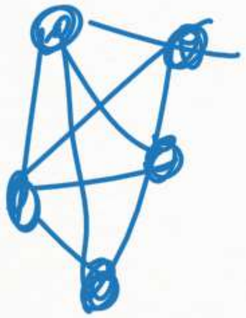


Kruskal's algorithm, refined

```
algorithm Kruskal

Let E = [e1,e2,…,em] be edges sorted by weight

for j in 1 to n          //Initially, each vertex is isolated
  Component[j] = j       //Component names are 1..n

TE = []                  //List of edges added so far
i = 1                    //Index of edge to try next

while TE.length() < n-1  //n-1 edges form a tree
  Let E[i] = (u,v)
  if Component[u] != Component[v] //E[i] does not form cycle
    TE.append(E[i])
    for j in 1 to n //Merge Component[v] into Component[u]
      if Component[j] == Component[v]
        Component[j] = Component[u]
```







Complexity

- Initially, sort edges, O(m log m)
  - m is at most n^2, so this is also O(m log n)
- Outer loop runs upto m times
  - In each iteration, we examine one edge
  - If we add the edge, we have to merge components
    - O(n) scan to update components
    - This is done once for each tree edge—O(n) times
- Overall $O(n^2)$

$\log n^2$

$2 \log n$

$O(n^2)$

Union-Find

Data Structure

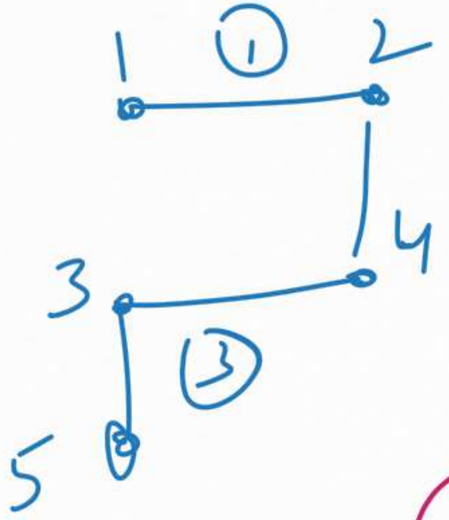
Home Algorithm

Write down code for Kruskal's Algorithm (removing the bug) as discussed in class.

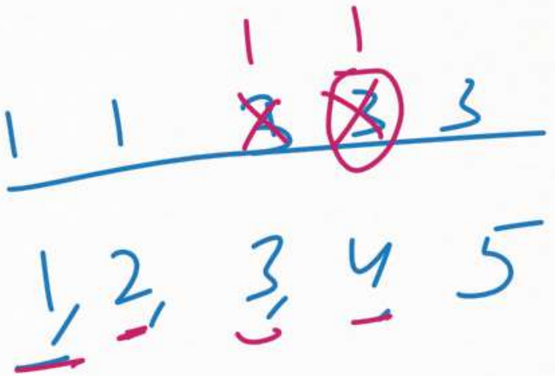
component array concept

Input of graph will be taken as in Prim's algorithm.

That's all



u=2
v=4



BUG

for j in 1 to n_5
    if Component[j] == Component[4]
        Component[j] = component[u]