papergrid
Date: / /

Eg

$f(n) = 2n+8$

$c_1 g(n_1) = 5n$

$c_2 g(n) = 7n$

$c > 0$

$c_2 g(n) \leq f(n) \leq c_1 g(n)$

If $n=1$

$f(n) = 2(1)+8 = 10$

$c_1 g(n_1) = 5n = 5$

$c_2 g(n) = 7n = 7$

$7 \leq 10 \leq 5$

$n=2$

$f(n) = 2(2)+8 = 12$

$c_1 g(n) = 5n = 5 \times 2 = 10$

$c_2 g(n) = 7n = 7 \times 2 = 14$

$14 \leq 12 \leq 10$ ×

$n=3$

$f(n) = 2(3)+8 = 14$

$c_1 (g(n)) = 5n = 15$

$c_2 (g(n)) = 7n = 21$

$21 \leq 14 \leq 15$ ×

$n=4$

$f(n) = 2(4)+8 = 16$

$c_1 g(n) = 5n = 20$

$c_2 g(n) = 7n = 28$

$n=5$

$f(n) = 2(5)+8 = 18$

$c_1 g(n) = 5n = 25$

$c_2 (g(n)) = 7n = 35$

$n=6$

$f(n) = 2(6)+8 = 20$

$c_1 g(n) = 5(6) = 30$

$c_2 g(n) = 7(6) = 42$

big oh → it is less than equal to
$f(n) \leq c \cdot g(n)$

small oh → it is only less than
$f(n) < c \cdot g(n)$

papergrid
Date: / /

- **O (small oh)**

→ It is same as big O.
that is $f(n) < og(n)$

- **ω (small/little omega)**

→ Similar to omega
$f(n) > \omega(g(n))$.

* **Mathematical analysis of recursive and non-recursive algorithm.**

- **Mathematical analysis of non-recursive algorithm.**

- **General conditions / plan**

1) Decide the input size based on the n value
2) Identify the basic operations
3) Check how many times basic operation is executed, determine the best, average, worst case.
4) Setup sum for the number of ~~types~~ times basic operation is executed.
5) Simplify the sum using standard formula

Formulas

1) $\sum_{i=1}^{n} 1 = 1+1+1+\cdots+1 = n \in \theta(n)$

2) $\sum_{i=1}^{n} i = 1+2+3+\cdots+n = \frac{n(n+1)}{2} \in \theta(n^2)$

3) $\sum_{i=1}^{n} i^K = 1+2^K+3^K+\cdots+n^K = \frac{n^{K+1}}{K+1} \in \theta(n^{K+1})$

4) $\sum_{i=1}^{n} a^i = 1+a+\cdots+a^n = \frac{a^{n+1}-1}{a-1} \in \theta(a^n)$

5) $\sum_{i=1}^{n} (a_i \pm b_i) = \sum_{i=1}^{n} a_i \pm \sum_{i=1}^{n} b_i$

6) $\sum_{i=1}^{n} (a_i) = c \sum_{i=1}^{n} a_i$

7) $\sum_{i=K}^{n} 1 = n - K + 1$

(Q) Finding the Element with max value

Problem description:
i/p:
o/p:

```
max_value ← A[0]
for i ← 1 to n-1 do
{
  if (a[i] > max_value) then
    max_value ← a[i]
}
return max_value
```

inp ⟹ 0 1 2 3 / 10 3 5 15 ; n=4 ?

a[i] > 10
i=1: 3 > 10 ×
i=2: 5 > 10 ×
i=3: 15 > 10 ✓
a[3] = 15

$$C(n) = \sum_{i=1}^{n-1} 1 \qquad \text{using formula 1}$$

$$= n - 1$$

$$= \theta(n)$$

(Q) Find wether all elements in an array are differ

Problem description:
i/p:
o/p:

```
for i ← 0 to n-2 do
{
  for j ← i+1 to n-1 do
  {
    if (a[i] = A[j]) then
      return false
  }
}
return true
```

0 1 2 3 / 10 20 10 30
a[0] = a[3]
10 = = 20 ×

outer loop $\sum_{i=0}^{n-2}$ · [inner loop $\sum_{j=i+1}^{n-1} 1$ · 1]

$$\sum_{j=i+1}^{n-1} 1 = (n-1) - (i+1) + 1 \Rightarrow n-1-i-1+1$$

$$= (n-1-i)$$

$$\sum_{i=0}^{n-2} (n-1-i)$$

~~$\sum_{i=0}^{n-2} (n-1) - \frac{n(n+1)}{2}$~~

$$\sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i \Rightarrow (n-1)\sum_{i=0}^{n-2} 1 - \sum_{i=0}^{n-2} i$$

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

$$(n-1)\sum_{i=0}^{n-2} 1 \cdot 1 - \frac{(n-2)(n-2+1)}{2}$$

$$(n-1)(n-2-0+1) - \frac{(n-2)(n-1)}{2}$$

$$(n-1)(n-1) - \frac{(n-2)(n-1)}{2}$$

$$\frac{2(n-1)(n-1) - (n-2)(n-1)}{2}$$

$$= \frac{2(n^2-2n+1) - (n^2-3n+2)}{2}$$

$$= \frac{(n^2-n)}{2} = \frac{1}{2} n^2 \qquad T(C) = \theta(n^2)$$

(Q) Matrix multiplication

```
for (int i=0; i<n; i++)
{
  for (int j=0; j<n; j++)
  {
    C[i][j] = 0;
    for (int k=0; k<n; k++)
    {
      C[i][j] = A[i][k] * B[k][j];
    }
  }
}
```

outer loop = n
middle = n
inner = n
∴ total = n × n × n = n^3
multiplication: A[i][k] * B[k][j]
addition: C[i][j] += (log)

1(mul) + 1(add) = 2
∴ total = $2n^3$

$C(n) = O(n^3)$

Formula

$t(n) = ?$

$\sum_{i=1}^{n} 1 = n$ (Formula 1)

$\sum_{j=1}^{n} n = n^2$

$T(n) = \sum_{i=1}^{n} n^2$

$\sum_{i=1}^{n} n^2 = n^3 \quad (\because \sum_{i=1}^{n} c = cn)$

$T(n) = O(n^3)$

- Mathematical Analysis of Recursive algorithms

- General condition

1) Decide the input size based on the n value
2) Identify the basic operation.
3) Check how many times basic operation is executed, determine the best, average and worst case.
4) Set up the recurrence relation with some initial condition of basic operation.

5) Solve the recurrence relation using backward substitution and forward substitution mechanism.

(a) Factorial of an algorithm: $n! = n.(n-1)!$

$5! = 5 \times 4!$

```
if (n == 0)
    return 1
else
    return F(n-1) * n.
```

$F(n) = F(n-1) * n$ → Equation of the factorial of an number.

$M(n) = M(n-1) + 1$

→ To multiply factorial n-1 by n.
→ The basic operation is multiplication, or else the multiplication should be performed atleast one time for this multiplications are required to compute factorial of (n-1).

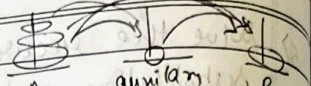
Forward substitution	Backward substitution
$M(0) = 1$	$M(n) = M(n+1)+1$
$M(1) = M(1-1)+1$	$M(n-1) = M((n-1)-1)+1$
$= M(0)+1$	$= M(n-2)+1$
$= 1+1 = 2$	~~$M(n-2) = M((n-2)-1)+1$~~
$M(2) = M(2-1)+1$	$M(n) = M(n-2)+1+1$
$= M(1)+1$	$= M(n-2)+2$
$= 2+1 = 3.$	$M(n-2) = M(n-2-1)+1$
	$= M(n-3)+1$
	$M(n) = M(n-3)+3$
	$M(n) = M(n-i)+i$
	$i = n$
	$M(n) = M(n-n)+n$
	$= M(0)+n$
	$= 1+n$
	$= n$
	$O(n)$

$M(n) = M(n-1) + M(n-1) + 1$

$M(n) = 2M(n-1) + 1$

2 recursive call + 1 move

Eg: Towers of Hanoi prob



(forward sub)

$M(n) = M(n-1) + M(n-1) + 1$

$M(n) = 2M(n-1) + 1$

Forward substitution

$M(0) = 0$

$M(1) = 1 \quad n \geq 1$

$M(2) = 3$

$M(3) = 7$

$M(4) = 15$

Backward substitution

$M(n) = 2M(n-1) + 1$

$= 2[2M(n-2) + 1] + 1$

$= 4M(n-2) + 3$

$= 4M[2M(n-3) + 1] + 3$

$= 8M(n-3) + 7$

$= 2^3 M(n-3) + 2^3 - 1$

$= 2^i M(n-i) + 2^i - 1$

$n = i$

$= 2^n M(n-n) + 2^n - 1$

$= 2^n M(0) + 2^n - 1$

$= 2^n (0) + 2^n - 1$

$= \theta(2^n)$

* Brute force method:

Examples of Brute force Mechanism

1) Selection sort & Bubble sort
2) Sequential search & String matching

(•) Selection sort Algorithm

```
for i ← 0 to n-2 do
{
  min ← i
  for j ← i+1 to n-1 do
  {
    if a[j] < a[min] then
      min ← j
  }
  temp ← a[i]
  a[i] ← a[min]
  a[min] ← temp
}
```

a[0] = min

a[1] < a[0]

Time complexity of swapping of numbers = $\theta(n)$

outer loop : inner loop

$\sum_{i=0}^{n-2} \left[\sum_{j=i+1}^{n-1} \cdot 1 \right]$

$T(C) = \theta(n^2)$

Time complexity of swapping of numbers = $\theta(n)$

(•) Bubble sort Algorithm

```
For i ← 0 to n-2 do
{
  For j ← 0 to n-2-i do
  {
    if a[j] > a[j+1] then
    {
      temp ← a[i]
      a[i] ← a[j+1]
      a[j+1] ← temp
    }
  }
```

$\theta(n)$

$\sum_{i=0}^{n-2} \cdot \sum_{j=0}^{n-2-i} \cdot 1$

$\sum_{j=0}^{n-2-i} \cdot 1 \Rightarrow (n-2-i-0+1)$

$= (n-1-i)$

→ Difference b/w selection sort & Bubble sort

→ Applications of Brute Force Algorithm

papergrid
Date: / /

$$\sum_{i=0}^{n-2} (n-1) - i$$

$$(n-1)\sum_{i=0}^{n-2} 1 \cdot 1 - \sum_{i=0}^{n-2} i$$

$$n\frac{(n+1)}{2} = n-2\ \frac{(n-2+1)}{2}$$

$$T(c) = \theta(n^2)$$

(•) Sequential search (Linear search) — also known as

Refer to Time complexity

$C_{best} = 1$

$C_{worst} = n$

$C_{avg} = \theta\left(\frac{n+1}{2}\right)$ (or) $\theta(\log n)$

```
for( i=0; i<n; i++)
{
if( key == a[i])
return i
}
```

$C(avg) = P(n+1) + n(1-p)$ if $p=1$

$C(avg) = \frac{1(n+1)}{2} + n(1-p)$

$= \frac{(n+1)}{2} + 0$

$C(avg) = \frac{n+1}{2}$ //

(•) Brute force string matching algorithm.

```
for i ← 0 to n-m
{
  j ← 0
  while j < m and p[i] = T[i+j] do
    j ← j+1
  if j = m
    return i
}
return -1
```

$T(c) = \theta(nm)$ → worst case time complexity

Average time ⇒ $\theta(n)$

papergrid
Date: / /

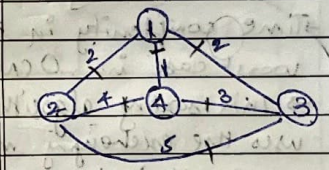
* Exhaustive Searching :

1) Travelling sales person problem
2) Assignment method (out of syllabus)
3) ~~Hoffman's problem~~ Knapsack problem.

(•) Travelling sale person problem :

Conditions :

1) The starting and ending vertex should be same. [It follows Hamiltonian circuit]
2) Has to travel all the cities.
3) Find the minimum cost



	1	2	3	4
1	0	2	2	1
2	2	0	5	4
3	2	5	0	3
4	1	4	8	0

	Tour	Path length
1.	1-3-4-2-1	2+3+4+2 = 11
2.	1-2-4-3-1	2+4+3+2 = 11
3.	1-3-2-4-1	2+5+4+1 = 12
4.	1-2-4-3-1	2+4+3+2 = 11
5.	1-4-2-3-1	1+4+5+2 = 12
6.	1-4-3-2-1	1+3+5+2 = 11

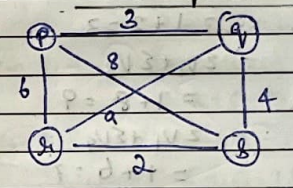
(•) Difference b/w Selection sort and Bubble sort

Selection Sort	Bubble sort
→ Selection sorting is a sorting algorithm where we select the minimum element from the array and put that at its correct position.	→ Bubble sorting is a sorting algorithm where we check two elements and swap them at their correct positions.
→ It time complexity in the Best Case is $O(n^2)$	→ It's time complexity in the best case is $O(n)$
→ Selection sort performs minimum no of swaps to sort the array	→ Bubble sort performs maximum no. of swaps to sort the array
→ Its Time complexity in the worst case is $O(n^2)$	→ Time complexity in the worst case is $O(n^2)$
→ This sorting algorithm uses the selection method	→ This sorting algorithm uses the exchanging method
→ efficient sorting technique	→ not efficient
→ not a stable algorithm	→ It is a stable algorithm
→ Faster	→ Slower.

(•) Application of Brute force Algorithm

1) String Matching: Used to search for a pattern within a text by checking each possible starting point
2) Password Cracking (Cybersecurity): Brute force attempts all possible characters combinations until the correct password is found.
Common in penetration testing & security analysis
3) Combinatorial Problems: Solving Problems with multiple possible combination
4) Sudoku Solver: Uses recursion & backtracking to try each possible number in empty cell.
5) Cryptanalysis: Breaking encryption keys by trying all possible keys until the correct one decrypts the message.
6) Finding Maximum/ Minimum in Array: Comparing every element to identify the maximum & or minimum value
7) Generating Subsets or Permutations: Common in set theory problems and combinational optimization.
8) Pathfinding in Graph: Used in exhaustive search strategies where all paths are explored.

(•) Travel sales person problem



	Tour	Path length
1	p-q-s-r-p	3+4+2+6 = 15
2	p-q-r-s-p	3+9+2+8 = 220
3	p-r-q-s-p	6+9+4+8 = 27
4	p-r-s-q-p	6+2+4+3 = 15
5	p-s-r-q-p	8+2+9+3 = 22
6	p-s-q-r-p	8+4+9+6 = 27
7		
8		
9		
10		
11		
12		

(*) Knapsack Problem

①

i	w_i (weight)	v_i (value)	capacity.
1	2	$1	W = 8.
2	3	$2	
3	4	$8	
4	5	$6	

Item 1 $\Sigma w_1 = 2$ $\Sigma v_1 = 1$

Item 2 $\Sigma w_2 = 3$ $\Sigma v_2 = 2$

Item 3 $\Sigma w_3 = 4$ $\Sigma v_3 = 8$

Item 4 $\Sigma w_4 = 5$ $\Sigma v_4 = 6$

Item 1, 2 $\Sigma w_1 + w_2 = 2+3 = 5$ $\Sigma v_1 + v_2 = 1+2 = 3$

Item 1, 3 $\Sigma w_1 + w_3 = 2+4 = 6$ $\Sigma v_1 + \Sigma v_3 = 1+8 = 9$

Item 1, 4 $\Sigma w_1 + w_4 = 2+5 = 7$ $\Sigma v_1 + \Sigma v_4 = 1+6 = 7$

Item 2, 3 $\Sigma w_2 + w_3 = 3+4 = 7$ $\Sigma v_2 + v_3 = 2+8 = 10$

Item 2, 4 $\Sigma w_2 + w_4 = 3+5 = 8$ (feasible) $\Sigma v_2 + v_4 = 2+6 = 8$

Item 3, 4 $= \Sigma w_3 + w_4 = 4+5 = 9$ (not feasible) $\Sigma v_3 + v_4 = 8+6 = 14$

Item 1, 2, 3 $= \Sigma w_1 + w_2 + w_3 = 2+3+4 = 9$ $\Sigma v_1 + v_2 + v_3 = 1+2+8 = 11$

Item 1, 2, 4 $= \Sigma w_1 + w_2 + w_4 = 2+3+5 = 10$ $\Sigma v_1 + v_2 + v_4 = 1+2+6 = 9$

Item 2, 3, 4 $= \Sigma w_2 + w_3 + w_4 = 3+4+5 = 12$ $= \Sigma v_2 + v_3 + v_4 = 2+8+6 = 16$.

The item 2, 4 satisfies the solution for a given knapsack with the given capacity 8.

Time complexity $= \theta(2^n)$.

②

i	w_i	v_i	capacity
1	2	$20	16.
2	5	$30	
3	10	$50	
4	5	$10	

Item 1 $\Sigma w_1 = 2$ $\Sigma v_1 = 20$

Item 2 $\Sigma w_2 = 5$ $\Sigma v_2 = 30$

Item 3 $\Sigma w_3 = 10$ $\Sigma v_3 = 50$

Item 4 $\Sigma w_4 = 5$ $\Sigma v_4 = 10$

Item 1, 2 $\Sigma w_1 + w_2 = 2+5 = 7$ $\Sigma v_1 + v_2 = 20+30 = 50$

Item 1, 3 $\Sigma w_1 + w_3 = 2+10 = 12$ $\Sigma w_1 + v_3 = 20+50 = 70$

Item 1, 4 $\Sigma w_1 + w_4 = 2+5 = 7$ $\Sigma v_1 + v_4 = 20+10 = 30$

Item 2, 3 $\Sigma w_2 + w_3 = 5+10 = 15$ $\Sigma v_2 + v_3 = 30+50 = 80$

Item 2, 4 $\Sigma w_2 + w_4 = 5+5 = 10$ $\Sigma v_2 + v_4 = 30+10 = 40$

Item 3 4 $\Sigma w_3 + w_4 = 10+5 = 10$ $\Sigma v_3 + v_4 = 50+10 = 60$

Item 1, 2, 3 $\Sigma w_1 + w_2 + w_3 = 2+5+10 = 17$ × $\Sigma v_1 + v_2 + v_3 = 20+30+50 = 100$

Item 1, 2, 4 $\Sigma w_1 + w_2 + w_4 = 2+5+5 = 12$ $\Sigma v_1 + v_2 + v_4 = 20+30+10 = 60$

Item 2, 3, 4 $\Sigma w_2 + w_3 + w_4 = 5+10+5 = 20$ $\Sigma v_2 + v_3 + v_4 = 30+50+10 = 90$

The item 2, 3 satisfies the solution for a given knapsack with the given capacity 16.

2, 3 is the optimal solution

Time complexity $= \theta(2^n)$

* Decrease and Conquer:

→ It is an approach for solving a problem
→ Changing an instance into one smaller instant of the problem.
→ ~~to~~ solve the smaller instance
→ Convert the solution of a smaller instance into a solution for the larger instance.
→ It follows top to bottom approach if its recursive process. non-recursive - bottom to top approach

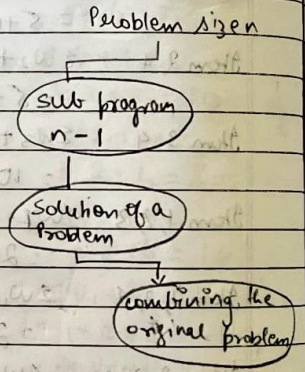
(•) Types of ~~Divide~~ decrease and conquer:

1) Decrease by ~~to~~ constant
2) Decrease by a constant factor
3) Variable size decrease.

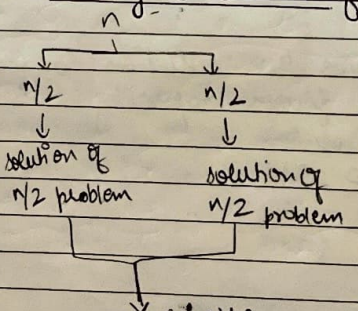
1) Decrease by constant.

Applications:
Insertion sort, Graph searching Algorithm (BFS, DFS), Topological sorting.

Eg- $a^{10} \Rightarrow a^9 \cdot a$



2) Decrease by a constant factor



Eg:- $a^{10} = a^5 \cdot a^5$

This is an example for binary search.

3) Variable size decrease.

We perform iteration and decrease the size of variables.
Eg:- GCD of number using Euclead process

(•) Decrease by constant

→ Insertion sort
→ ~~Topological~~ Topological sorting.

⇒ Insertion sort

- The given list is divided in 2 parts sorted array & unsorted array
- For every ~~list~~ of array 1st element we are taking as sorted array all remaining elements are unsorted.
- The first element in the unsorted array is taken as the temporary variable.
- We check the temp variable with sorted array if we found greater value we are shifting to one right position. [i.e if sorted array is greater than USA then the SA is shifted to one position to the right]
- The sorted list is checked from right to left & the US list is checked from left to right
- In this method the element is inserted at its appropriate position.

→ Algorithm.

```
For i ← 1 to n-1 do
{
  temp ← A[i]
  j = i-1 // set j at previous element of a[i]
  while (j >= 0) AND (A[j] > temp) do
```

```
{
  A[j+1] ← A[j]
  j = j-1
}
A[j+1] ← temp
}
```

$C_{Best} = \sum_{i=1}^{n-1} 1$

$= (n-1-1+1)$

$= n-1$

$= \theta(n)$

$C_{avg} = \frac{(n-1)n}{2} \times \frac{1}{2}$

$= \frac{n^2-n}{4} = \frac{n^2}{4}$

$C_{avg}(n) = \theta(n^2)$

$C_{worst} = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1$

$= \sum_{i=1}^{n-1} (i-1-0+1)$

$= \sum_{i=1}^{n-1} (i)$

$= \theta(n^2)$

(8). 5 | 4 10 1 6 2

SA | USA

5 ☐ 10 1 6 1 2 temp [4]

5 > 4 ✓

So the sorted element is moved to the right

4 5 | ☐ 1 6 1 2 temp [10]

SA | USA

5 > 10 × No movement in the array

4 > 10 × No movement in the array

∴ No change in array

4 5 10 | 1 6 12

SA | USA

Right to left ← left to right →

4 5 10 | 1 ☐ 6 12 temp [1]

SA

10 > 1 ✓ move to the right

4 5 ? 10 6 2 temp [1]

5 > 1 ✓ move to the right

4 ? 5 10 6 2 temp [1]

4 > 1 ✓ move to the right

1 4 5 10 | 6 2

SA | USA

1 4 5 10 | 1 2 temp [6]

SA | USA

10 > 6 ✓ move to the right

1 4 5 ? 10 2

5 > 6 × So no movement in the array.

1 4 5 6 10 | 2

SA | USA

1 4 5 6 10 | ☐ temp [2]

SA | USA

10 > 2 ✓ move to the right temp

1 4 5 6 ? 10 [2]

6 > 2 ✓ move to the right temp

1 4 5 ? 6 10 [2]

5 > 2 ✓ move to the right temp

1 4 ? 5 6 10 [2]

4 > 2 ✓ move to the right temp

1 ? 4 5 6 10 [2]

1 > 2 × So no movement in the array

1 2 4 5 6 10

SA

(Q) 30 ; 20 10 40 50
SA USA
temp 20
30 > 20 ✓ move to the right

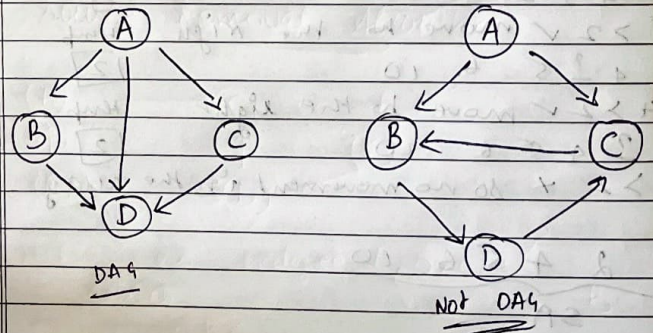
20 30 ; 10 40 50
SA USA

20 30 ; □ 40 50 temp 10
SA
30 > 10 ✓ move to the right temp 110
20 ? 30 40 50
20 > 10 ✓ move to the right temp 10
10 20 30 ; 40 50 temp 140
SA USA
30 > 40 ✗ No movement in array
10 20 30 40 ; 50 temp 50
SA USA
10 20 30 40 □
40 > 50 ✗ No movement in array.

10 20 30 40 50
SA ✓.

⇒ Topological Sorting

DAG → Direct acyclic graph



Conditions :

→ A directed acyclic graph is directed graph with no cycles
→ Based on the principle of DAG specific ordering of vertices is possible
→ This method of arranging the vertices in some specific manner is called Topological sort

Topological sort (2 types)

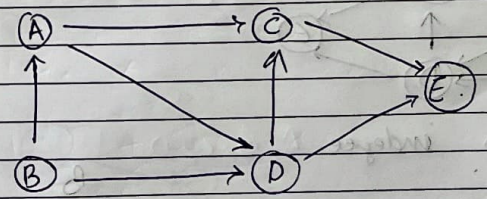
1. DFS based algorithm
2. Source removal algorithm

DFS → i → j [i to j means i will come first then j]
i j

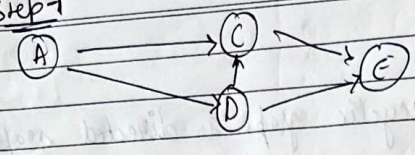
1. DFS

→ Steps:
Removing vertices & placing in stack
↓
Poping the elements in stack
↓
Reversing the string.

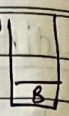
→ DFS follows stack
→ If the vertex is having no or less indegree it is removed first

(Q)


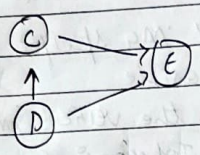
Step-1

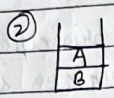


B is removed first

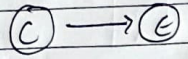


A is removed

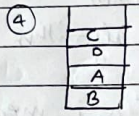




D is removed

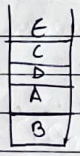


C is removed



E is pushed in the last on the stack

Stack →



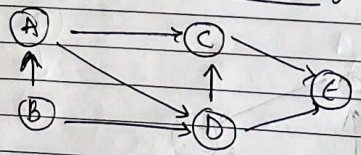
Step-2 (Poping the elements from the stack)

E C D A B.

Step-3 Reversing the string

B A D C E

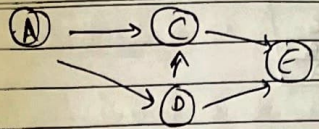
(2) Source Removal algorithm



B has no indegree

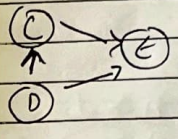
delete B.

B (1)



Next delete A

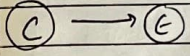
B, A



B

Next delete D

B, A, D



Next delete C

B, A, D, C

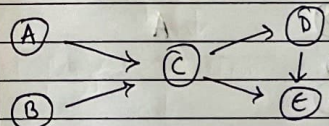
Next → B, A, D, C, E

In source removal algorithm

we do not place the elements in a stack or pop or reverse the string

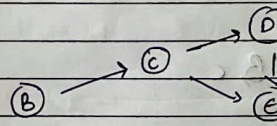
we just remove the elements in order

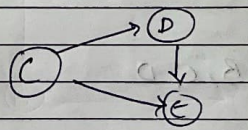
(3)

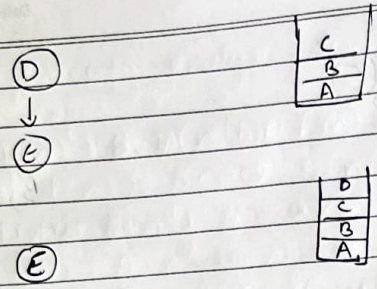


DFS

Step-1



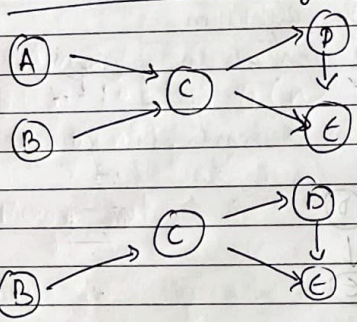




Step-2 E D C B A

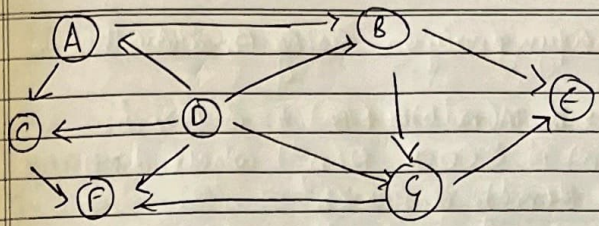
Step-3 A B C D E

Source Removal Algorithm:

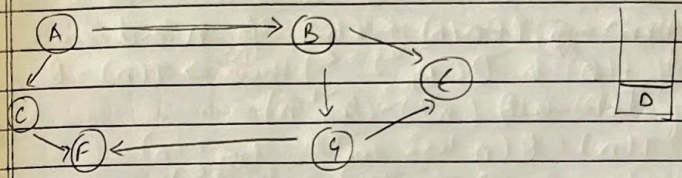


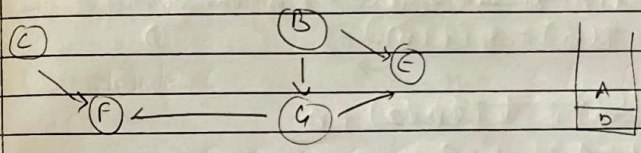
A, B, C, D, E

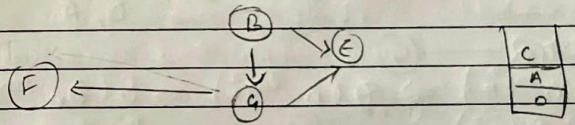
(9).

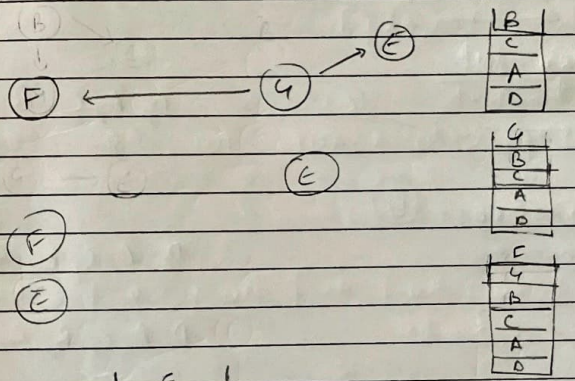


Step1

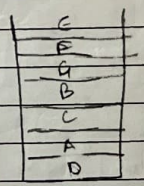








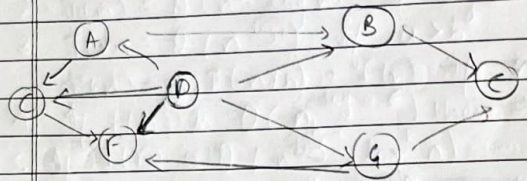
Stack →

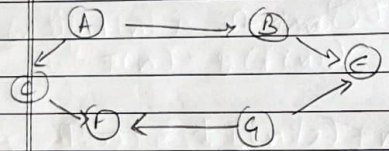


Step-2 Poping ⇒ E F G B C A D

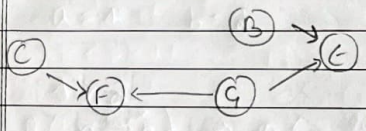
Step-3 → D A C B G F E

Source Removal Algorithm

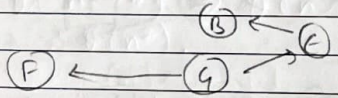




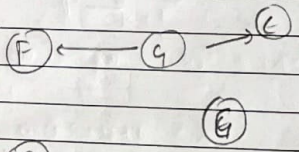
D



D, A



D, A, C



D, A, C, B

D, A, C, B, G

D, A, C, B, G, E

D, A, C, B, G, F, E

Eg:- Binary search, merge sort, quick sort, traversal techniques.

0, a[n-1]

* Divide and conquere :

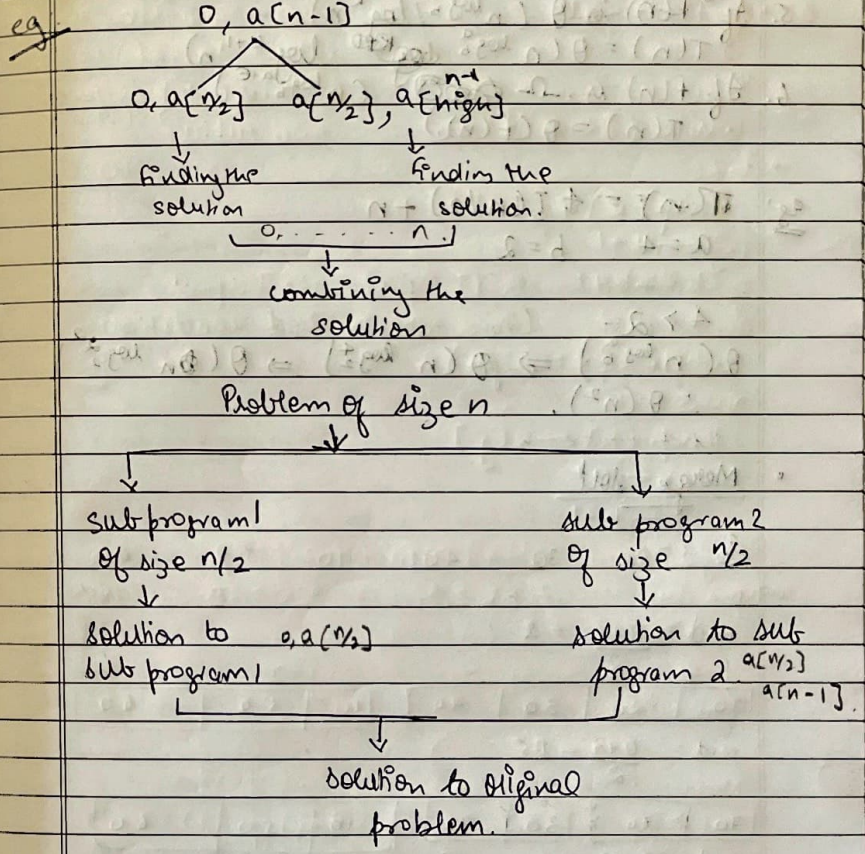
→ In divide and conquer method a given problem is :

i) Divide into smaller subprograms

ii) These subproblems are solved independantly

iii) If necessary the solution of subproblems are combined to get a solution to the original problem

eg:



- Time complexity :- → n/2 · n/2

$$T(n) = a \cdot T(n/b) + f(n)$$

$$T(n) = 2 \cdot T(n/b) + f(n)$$

T(n) = time for size n

a = no of subproblems

T(n/b) = time for size (n/b)

f(n) = Time required for dividing the problem into subproblems.

ϵ → epsilon

→ Efficiency analysis using masters theorem.

$T(n) = a \cdot T(n/b) + f(n)$ $\quad n \geq d$

If $f(n)$ is $\Theta(n^d)$ where $d \geq 0$

1. $T(n) = \Theta(n^d)$ if $a < b^d$
2. $T(n) = \Theta(n^d \log n)$ if $a = b^d$
3. $T(n) = \Theta(n^{\log_b a})$ if $a > b^d$

$T(n) = a \cdot T(n/b) + f(n)$ if $n \geq d$

4. If $f(n)$ is $O(n^{\log_b a - \epsilon})$ then $\Rightarrow T(n) = \Theta(n^{\log_b a})$
5. If $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$
   $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
6. If $f(n)$ is $\Omega(n^{\log_b a + \epsilon})$ → epsilon
   $T(n) = \Theta(f(n))$

eg: $T(n) = 4T(n/2) + n$

$a = 4 \quad b = 2$

$4 > 2$ (This satisfies 3rd condition)

$\Theta(n^{\log_b a}) \Rightarrow \Theta(n^{\log_2 4}) \Rightarrow \Theta(n^{\log_2 2^2})$

$\therefore \Theta(n^2)$.

- Merge sort

70	20	30	40	10	50	60
low 0	1	2	3	4	5	high 6

$mid = \frac{0+6}{2} = 3$

70	20	30	40	10	50	60

$mid = \frac{0+3}{2} = 1.5$

70	20	30	40	10	50	60

$mid = \frac{0+2}{2} = 1$

70	20	30	40	10	50	60

$\frac{0+1}{2} = 0.5$

70	20	30	40	10	50	60

$\frac{0+1}{2} - 0.5$

70	20	30	40	10	50	60

$\frac{0+1}{2} = 0.5$

70	20	30	40	10	50	60

20	70	30	40	10	50	60

20	30	40	70	10	50	60

10	20	30	40	50	60	70

Algorithm

~~merge sort first~~

```
merge sort (int, a[0, n-1), low, high)
{
  if (low < high) then
  {
    mid = (low + high) / 2
    merge sort (a, low, mid)
    merge sort (a, mid+1, high)
    combine (a, low, mid, high)
  }
```

Algorithm combine

```
Alg combine (a[0....n-1], low, mid, high)
{
  k ← low
  i ← low
  j ← mid+1
  while (i<=mid and j <= high) do
  {
    if (a[i] <= a[j]) then
```

```
{
  temp[k] ← a[i]
  i ← i+1
  k ← k+1
}
else
{
  temp[k] ← a[j]
  j ← j+1
  k ← k+1
}
while (i<=mid) do
{
  temp[k] ← a[i]
  i ← i+1
  k ← k+1
}
while (j<=high)
{
  temp[k] ← a[j]
  j ← j+1
  k ← k+1
}
```

Time complexity of Merge sort

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

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

$a = 2$

$b = 2 \quad a = b^d, \; d = 1$

$\theta(n^d \log_2 n)$

$= \theta(n \log n)$.

$T(n/2)$ → Time taken to left sublist to get sorted

$T(n/2)$ → Time taken to right sublist to get sorted

cn → Time taken for combining two sublist

Eg: Merge sort

10 5 7 6 1 4 8 3 2 9

low: 0 1 2 3 4 5 6 7 8 9 high

$mid = \frac{0+9}{2} = 4.5$

10 5 7 6 1 4 | 8 3 2 9

$mid = \frac{0+5}{2} = 2.5$

10 5 7 6 | 1 4 | 8 3 2 9

$\frac{0+3}{2} = 1.5$

10 5 | 7 6 | 1 4 | 8 3 | 2 9

$\frac{0+1}{2} = 0.5$

10 | 5 | 7 | 6 | 1 | 4 | 8 | 3 | 2 | 9

5 10 | 6 7 | 1 4 | 3 8 | 2 9

5 6 7 10 | 1 3 4 8 | 2 9

1 3 4 5 6 7 8 10 | 2 9

1 2 3 4 5 6 7 8 9 10

HW

* Difference b/w Divide & Conquer and Brute force mechanism

→ Applications of Divide & Conque mechanism
→ advantages, disadvantages.
→ difference b/w merge sort and quick sort.

papergrid
Date: / /

- **Quick Sort**

1) → Consider the first element as pivot element
2) → Initialize i to low, j to high.
3) → Repeat the following steps until i < j
   i) Keep on incrementing i while a[i] ≤ pivot
   ii) Keep on decrementing j while a[j] ≥ pivot
   iii) If i < j then swap (a[i], a[j])
   iv) If i ≥ j then swap (a[j], pivot)
4) → j is the position of pivot value.

eg:

30	20	10	50	60	40
0	1	2	3	4	5
30 (pivot)	20	10	50	60	40 (high)

→ i < j

0 < 5 (T)	a[1] ≤ pivot	a[2] ≤ pivot	a[3] ≤ pivot
a[i] ≤ pivot	20 ≤ 30 (T)	10 ≤ 30 (T)	50 ≤ 30 (F)
a[0] ≤ 30	i++	i++	∴ i = 3
30 ≤ 30 (T)	i = 2	i = 3	
i is incremented (i++)			
i = 1			

a[j] ≥ pivot	a[4] ≥ Pivot	a[j] ≥ pivot	a[j] ≥ pivot
j = 5	60 ≥ 30 (T)	50 ≥ 30 (T)	10 ≥ 30 (F)
40 ≥ 30 (T)	j--	j--	∴ j = 2
j is decremented (j--)	j = 3	j = 2	
j = 4			

Now i = 3 & j = 2
i ≥ j ⇒ 3 ≥ 2 swap (a[j], pivot)
(a[2], 30)
(10, 30)

10	20	30	50	60	40
LS		Pivot	RS.		

low		high.
0	1	2
50 (Pivot)	60	20
	i	j

i < j ⇒ 0 < 2 (T) | 60 ≤ 50 (F)
a[i] ≤ pivot | ∴ i = 1
50 ≤ 50
i = 1 (i++)

a[j] ≥ pivot
a[ ] ≥ 50
40 ≥ 50 (F)
j = 2

∴ Now i = 1 & j = 2 swap (a[i], a[j])
swap (60, 40)

∴ 50 40 60

low		high.
0	1	2
50 (pivot)	40	60 (high)

i < j ⇒ 0 < 2 (True) | a[i] ≤ pivot
a[i] ≤ pivot | a[1] ≤ 50
a[0] ≤ pivot | 40 ≤ 50 (T)
50 ≤ 50 (T) | ∴ i = 2. (i++)
i++
i = 1

a[i] ≤ pivot	a[j] ≥ pivot
a[2] ≤ 50	a[2] ≥ 50
60 ≤ 50 (F)	60 ≥ 50 (T)
∴ i = 2	j = 1 (j decrements (j--))

a[j] ≥ pivot | Now i = 2 & j = 1
a[ ] ≥ 50 | i ≥ j ⇒ 2 ≥ 1 swap (a[j], pivot)
40 ≥ 50 (F) | (a[1], 50)
∴ j = 1 | (40, 50)

40	50 (pivot)	60

10	20	30	40	50	60 //

If pivot is the minimum or ...
worst case Time complexity.

papergrid
Date: / /

Time complexity

Best case
$T(n) = T(n/2) + T(n/2) + C$
$= 2T(n/2) + C$
$\theta(n \log n)$

worst case
$T(n) = T(n-1) + C$
$\theta(n^2)$

Eg: Perform the quick sort 50, 30, 10, 90, 80, 20, 40, 70.

low	0	1	2	3	4	5	6	7
	50	30	10	90	80	20	40	70
	pivot			i=3			j=6	high

i<j	a[i]≤pivot	a[i]≤pivot	a[i]≤pivot
0<7 (T)	a[1]≤50	a[2]≤50	a[3]≤50
a[i]≤pivot	30≤50(T)	10≤50(T)	90≤50(False)
a[0]≤50	i++	i++	i=3
50≤50(T)	i=2	i=3	
i++			
i=1			

a[j]≥pivot	a[6]≥50
j=7	40≥50(F)
a[7]≥50	j=6
70≥50(T)	
j--	
j=6	

Now i=3 & j=6
i<j => 3<6 swap (a[i], a[j])
(a[3], a[6])
(90, 40)

pivot							
50	30	10	40	80	20	90	70
0	1	2	3	4	5	6	7
			i			j	

papergrid
Date: / /

i≤j	a[i]≤pivot	a[j]≥pivot
3<6 (T)	a[4]≤50	j=6
a[i]≤pivot	80≤50(F)	a[6]≥50
a[3]≤50	i=4	90≥50(T)
40≤50(T)		j--
i++		j=5
i=4		

a[j]≥pivot	Now => i=4 & j=5
j=5	i<j => 4<5
a[5]≥50	swap (a[i], a[j])
20≥50(F)	(a[4], a[5])
j=5	(80, 20)

50	30	10	40	20	80	90	70
0	1	2	3	4	5	6	7
				i	j		

i<j	a[i]≤pivot	a[j]≥pivot	a[4]≥pivot
4<5(T)	a[5]≤pivot	j=5	20≥50 (F)
a[i]≤pivot	80≤50(F)	a[5]≥50	j=4
a[4]≤50	i=5	80≥50(T)	
20≤50(T)		j--	
i++		j=4	
i=5			

Now i=5 & j=4
i>j => 5>4
swap (a[j], pivot)
(a[4], 50)
(20, 50)

low	0	1	2	3	pivot 4	5	6	7 high
	20	30	10	40	50	80	90	70
	LS					RS		

20 30 10 40
i pivot 1 2 3 j

i<j	a[i]≤pivot	a[j]≥pivot	a[j]≥pivot
0<3	a[1]≤20	j=3	j=2
a[i]≤pivot	30≤20(F)	a[3]≥20	a[j]≥20
a[0]≤20	i=1	40≥20(T)	10≥20
20≤20(T)		j--	j=2
i++		j=2	
i=1			

Now ⇒ i=1 & j=2
i<j ⇒ 1<2 swap(a[i], a[j])
(a[1], a[2])
(30, 10)

Pivot 20 10 30 40
0 1 2 3
i j

i<j	a[i]≤pivot	a[j]≥pivot	a[j]≥pivot
1<2(T)	a[2]≤20	j=2	j=1
a[i]≤pivot	30≤20(F)	a[j]≥pivot	a[j]≥pivot
a[1]≤20	i=2	30≥20(T)	10≥20(F)
10≤20(T)		j--	j=1
i++		j=1	
i=2			

Now i=2, j=1
i>j
2>1 swap(a[j], pivot)
(a[1], 20)
(10, 20)

10 20 30 40

10 20 30 40 50 80 90 70
LS RS

low 0 high 2
80 90 70
(pivot) i j

i<j	a[i]≤pivot	a[j]≥pivot
0<2	a[1]≤80	j=2
a[i]≤pivot	90≤80(F)	a[j]≥80
80≤80(T)	i=1	70≥80(False)
i++		j=2
i=1		

Now, i=1 & j=2
i<j ⇒ 1<2
70≥ swap(a[i], a[j])
j=2 (a[1], a[2])
(90, 70)

0 1 2
80 70 90 high
pivot i j

i<j	a[i]≤pivot	a[j]≥pivot	a[j]≥pivot
1<2	a[2]≤80	j=2	j=1
a[i]≤pivot	90≤80(F)	a[j]≥pivot	a[j]≥80
a[1]≤80	i=2	a[2]≥80	70≥80(F)
70≤80(T)		90≥80(T)	j=1
i++		j--	
i=2		j=1	

Now ⇒ i=2 & j=1
i>j ⇒ 2>1 swap(a[j], pivot)
(a[1], 80)
(70, 80)

70 80 90
∴ the sorted array is

10 20 30 40 50 70 80 90

* Algorithm

```
Alg Quick(a[0...n-1], low, high)
if (low < high) then
  split the array into 2 subarrays
m ← partition (A (low .... high))
Quick (a (low, m-1))
Quick (a(mid+1, ... high-1)
```

→ Algorithm for partition:

```
Alg par (A(low, high))
{
pivot ← a[low]
 i ← low
 j ← high+1
while (i == j) do
{
while (a[i] <= pivot) do
 i = i+1
while (a[j] >= pivot)
 j ← j-1
if (i <= j) then
 swap (a[i], a[j])
}
swap (a[low, a[j])
return j;
}
```

* Multiplication of large integers

```
  42
 x34
 168
126 x
1428
```

$C = a \times b$

$\underline{123} \; \underline{456} \times \underline{789} \; \underline{123}$

$A_0 \quad A_1 \quad B_0 \quad B_1$

$C = a \times b$

$\boxed{C = C_2 10^n + C_1 10^{n/2} + C_0}$

$C_2 = a_1 * b_1$

$C_0 = a_0 \times b_0$

$C_1 = (a_1 + a_0) * (b_1 + b_0) - (C_2 + C_0)$

Eg: $\underset{a \; b}{42 \times 34}$ $\quad n = 2$

$= a_1 a_0 \quad b_1 b_0$

$a_1 = 4 \quad b_1 = 3$

$a_0 = 2 \quad b_0 = 4$

$C_1 = (a_1 + a_0) * (b_1 + b_0) - (C_2 + C_0)$

$C_2 = a_1 \times b_1 \qquad C_0 = a_0 \times b_0$

$= 4 \times 3 \qquad = 2 \times 4$

$= 12 \qquad = 8$

$C_1 = (4+2) \times (3+4) - (12+8)$

$= (6) \times (7) - (20)$

$= 42 - 20$

$= 22$

$C = 12 \times 10^{2^2} + 22 \times 10^{2/2} + 8$

$= 12 \times 100 + 22 \times 10 + 8$

$= 1200 + 220 + 8$

$= 1428$ //.

~~(3) 123456 x 789123~~

(•) Time complexity:

$T(c) = 3T(n/2) + 1$

$a = 3, \; b = 2 \qquad a > b$

$T(n) = \theta(n \log_b^a)$ (using masters theorm)

$= n^{\log_2^3}$

$= (n^{1.5})$

(8) 123456 × 789123

a1 a0 b1 b0

a = 123 456 b = 789 123

$a_1 = 123$ $b_1 = 789$

$a_0 = 456$ $b_0 = 123$

$C_1 = (a_1 + a_0) * (b_1 + b_0) - (C_2 + C_0)$

$C_2 = a_1 \times b_1$ $C_0 = a_0 \times b_0$

$= 123 \times 789$ $= 456 \times 123$

$= 97,047$ $= 56,088$

$C_1 = (123 + 456) * (789 + 123) - (97047 + 56088)$

$= (579) \times (912) - (153,135)$

$= 528048 - 153135$

$= 374913$

$C = C_2 10^n + C_1 10^{n/2} + C_0$

$= 97046 \times 10^6 + 374913 \times 10^3 + 56088$

=

=

* Strassen's Matrix Multiplication

$C = A \times B$

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

$C_{11} = (A_{11} \times B_{11}) + (A_{12} \times B_{21})$

$C_{12} = (A_{11} \times B_{12}) + (A_{12} \times B_{22})$

$C_{21} = (A_{21} \times B_{11}) + (A_{22} \times B_{21})$

$C_{22} = (A_{21} \times B_{12}) + (A_{22} \times B_{22})$

Time complexity:

(of Normal matrix)

Multiplication → 8

Add → 4

$T(C) = c \times n \times n$

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

→ Divide and conquer method using Strassen's Matrix multiplication, here if the matrix is 2×2 we can accomplish seven (7) multiplications and 18 addition or subtraction. instead of normal method (8 multiplications & 4 addition).

→ Divide matrix into submatrices like a_0, a_1, a_2

→ Conquer: Use a group of matrix multiplication equations combine recursively multiply submatrix and get the final results of multiplication after performing required addition or subtraction

$S_1 = (A_{11} + A_{22})(B_{11} + B_{22})$

$S_2 = (A_{21} + A_{22}) \times B_{11}$

$S_3 = A_{11} \times (B_{12} - B_{22})$

$S_4 = A_{22} \times (B_{21} - B_{11})$

$S_5 = (A_{11} + A_{12}) \times B_{22}$

$S_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$

$S_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$

$C_{11} = S_1 + S_4 - S_5 + S_7$

$C_{12} = S_3 + S_5$

$C_{21} = S_2 + S_4$

$C_{22} = S_1 + S_3 - S_2 + S_6$

$$C = \begin{bmatrix} A_{11}: 1 & A_{12}: 2 \\ A_{21}: 3 & A_{22}: 4 \end{bmatrix} \begin{bmatrix} B_{11}: 5 & B_{12}: 6 \\ B_{21}: 7 & B_{22}: 8 \end{bmatrix}$$

$S_1 = (1 + 4)(5 + 8) = 5 \times 13 = 65$

$S_2 = (3 + 4)(5) = 7 \times 5 = 35$

$S_3 = 1 \times (6 - 8) = -2$

$S_4 = 4 \times (7 - 5) = 4 \times 2 = 8$

$S_5 = (1 + 2) \times 8 = 24$

$S_6 = (3 - 1) \times (5 + 6) = 2 \times 11 = 22$

$S_7 = (2 - 4) \times (7 + 8) = -2 \times 15 = -30$

$C_{11} = 65 + 8 - 24 + (-30) = 19$

$C_{12} = -2 + 24 = 22$

$C_{21} = 35 + 8 = 43$

$C_{22} = 65 + (-2) - (35) + 22 = 50.$