CMR Institute of Technology
Department of Computer Science & Engineering
IAT 1
14SCS23 – Advanced Algorithms

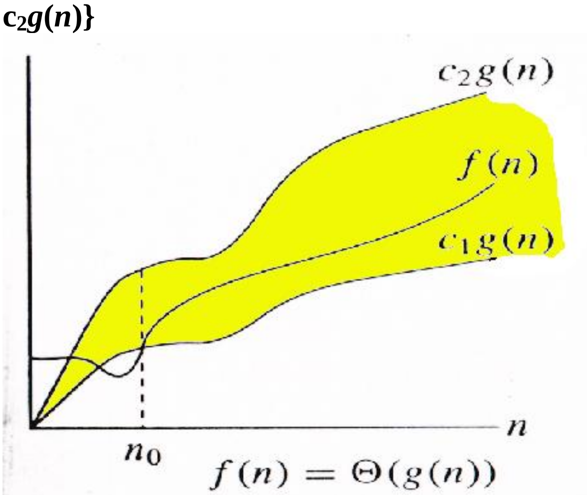
1. **a)Define and explain the various asymptotic notations with related graphs and examples.**
   Definition 1, each 3

- Running time of an algorithm as a function of input size n **for large n**.
- Expressed using only the **highest-order term** in the expression for the exact running time.
  - Instead of exact running time, say Q(n^2).
- Describes behavior of function in the limit.
- Written using **Asymptotic Notation**.

- **Q, *O*, W, *o*, w**
- Defined for functions over the natural numbers.
  - **Ex:** $f(n)$ = Q(n^2).
  - Describes how $f(n)$ grows in comparison to n^2.
- Define a ***set*** of functions; in practice used to compare two function sizes.
- The notations describe different rate-of-growth relations between the defining function and the defined set of functions.

Θ-notation

n(**$g(n)$**) = {**$f(n)$** : A **positive constants c_1, c_2, and n_0, such that** ?**n** ñ **n_0**, **0** á **$c_1g(n)$** á **$f(n)$** á **$c_2g(n)$**}

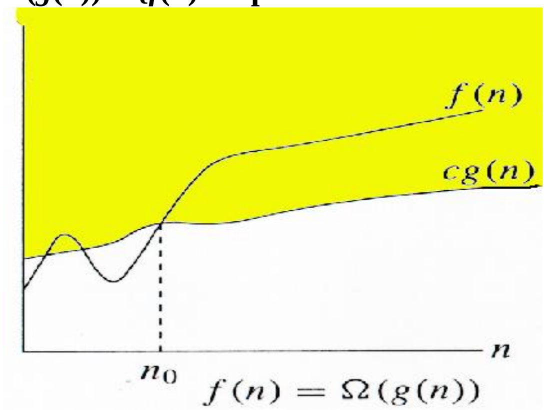


Ω -notation

$\Omega(g(n)) = \{f(n) : \exists$ positive constants c and n_0, such that $\forall n \geq n_0$, we have $0 \leq cg(n) \leq f(n)\}$

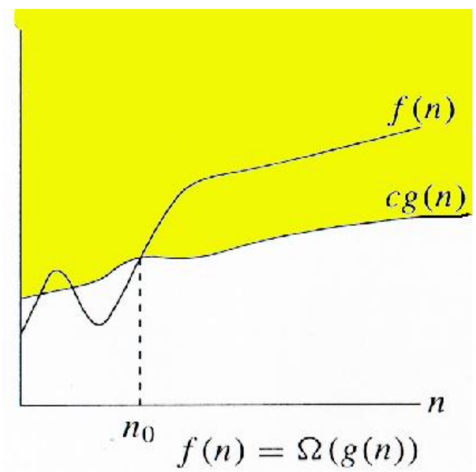
<u>*O*-notation</u>

$O(g(n))$ = {$f(n)$: A positive constants c and n_0, such that ?n ñ n_0, we have 0 á $f(n)$ á c$g(n)$ }



<u>Ω -notation</u>

$\Omega(g(n)) = \{f(n) : \exists$ positive constants c and n_0, such that $\forall n \geq n_0$, we have $0 \leq cg(n) \leq f(n)\}$



Q3.AIllustrate the aggregate analysis of amortized analysis on the operation INCREMENT in a binary counter. (definition 2, explanation-6, derivation 2) ex: 2 Incrementing a binary counter.

→ Implement a k-bit binary counter, that counts upward from 0.

→ An array A [0 .... k-1] of bits, where A.length = k, as the counter.

→ x = binary number ; lowest-order bit = A [0] ; highest-order bit = A [k-1].

So, $x = \sum_{i=0}^{k-1} A[i] \cdot 2^i < n \sum_{i=0}^{\infty} 1/2^i = n\left(\frac{1}{1-1/2}\right) = 2n.$

$\therefore$ Increment cost = $O(n)$.

Avg cost per operation = $O(1)$.

pseudocode.

INCREMENT (A)

1. i = 0
2. while i < A.length and A[i] == 1
3. A[i] = 0
4. i = i+1
5. if i < A.length
6. A[i] = 1.

→ The single execution of INCREMENT ...

(k) in the worst case.

→ So, a sequence of n Increment operations on an initially zero counter takes time $O(nk)$ in the worst case.

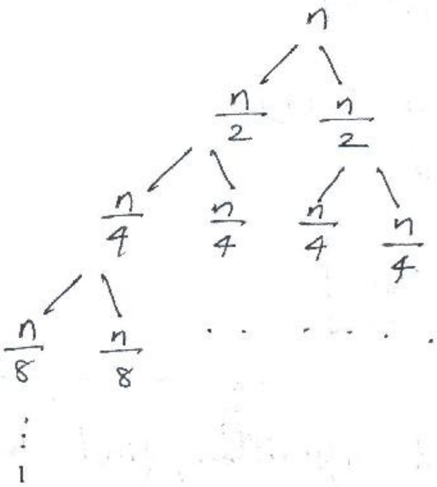
Counter Value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
.	0	0	0	0	0	1	0	1	8
.	0	0	0	0	0	1	1	0	10
.	0	0	0	0	0	1	1	1	11
.	0	0	0	0	1	0	0	0	14
.	0	0	0	0	1	0	0	1	15
.	0	0	0	0	1	0	1	0	
.	0	0	0	0	1	0	1	1	
	0	0	0	0	1	1	0	0	
	0	0	0	0	1	1	0	1	
	0	0	0	0	1	1	1	0	
	0	0	0	0	1	1	1	1	
	0	0	0	1	0	0	0	0	

cost of increment = Θ(# of bits flipped)

2. a) Use a recursion tree to determine a good asymptotic upper bound on the recurrence relation

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

Soln.



Height = $\lg n$.

cost = $\theta(n \lg n)$.

b) State the master theorem and solve the following recurrence relations using Master theorem.

i) $T(n) = 9 \cdot T\left(\frac{n}{3}\right) + n$

Soln. gn: $a = 9$; $b = 3$; $f(n) = n$.

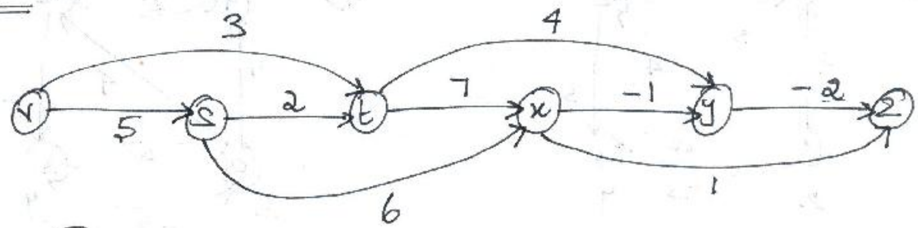
$$n^{\log_b a} = n^{\log_3 9} = n^{\log_3 3^2} = n^2 = \theta(n^2).$$

$f(n) = O(n^{\log_3 9 - \epsilon})$; where $\epsilon = 1$.

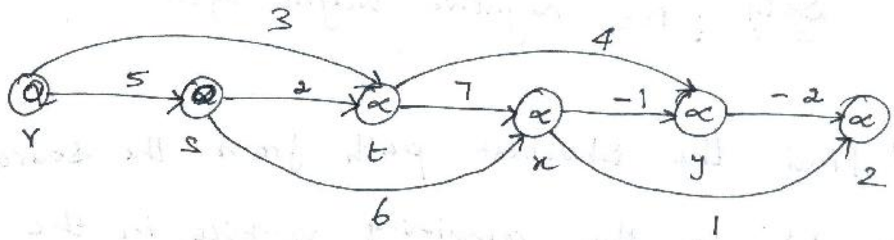
So, case 1 is applied.

$\therefore \boxed{T(n) = \theta(n^2)}$

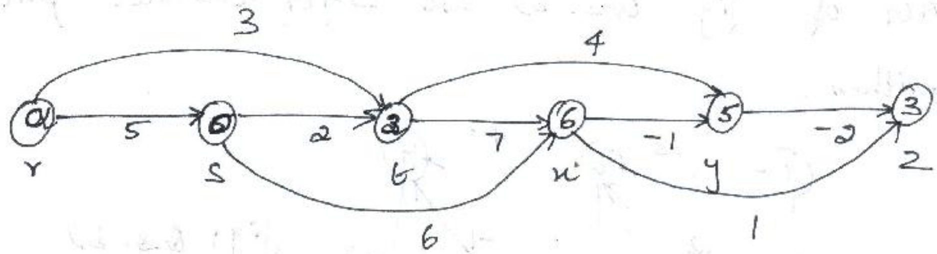
step : 1



step : 2



step : 3



2. c) Write the johnson's algorithm to solve all-pairs shortest path problem for sparse graphs.

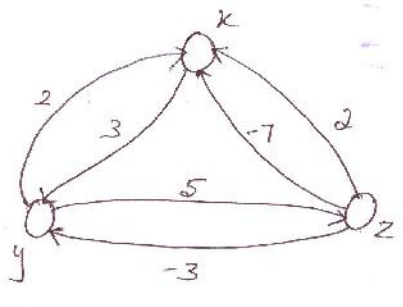
Johnson's algorithm

form G'
run BELLMAN-FORD on G' to compute $\delta(s, v)$ for all $v \in V$
if BELLMAN-FORD returns FALSE
  then G has a negative-weight cycle
  else
    compute $\widehat{w}(u, v) = w(u, v) + \delta(s, u) - \delta(s, v)$ for all $(u, v) \in E$
    for each vertex $u \in V$
      do run Dijkstra's algorithm from u using weight function $\widehat{w}$
        to compute $\widehat{\delta}(u, v)$ for all $v \in V$
      for each vertex $v \in V$
        do ▷ Compute entry d_{uv} in matrix D

$$d_{uv} = \underbrace{\widehat{\delta}(u, v) + \delta(s, v) - \delta(s, u)}$$

because if p is a path $u \rightsquigarrow v$,
then $\widehat{w}(p) = w(p) + h(u) - h(v)$

b) Write Johnsons algorithms for sparse graphs. Use same to find shortest paths between all pairs of vertices in the graph of fig. Q1 (b). [10]



Soln:

Johnsons alg:

Johnson (G, w)

Compute G', where G'.V = G.V ∪ {s}

G'.E = G.E ∪ {(s, v) ; v ∈ G.V}, and

w(s, v) = 0 for all v ∈ G.V

if BELLMAN-FORD (G', w, s) = = FALSE

print "the input graph contains a negative-weight cycle"

else for each vertex v ∈ G'.V

set h(v) to the value of δ(s, v)

computed by the Bellman ford algorithm

for each edge (u, v) ∈ G'.E

$\hat{w}(u, v) = w(u, v) + h(u) - h(v)$

let $D = (d_{uv})$ be a new n x n matrix

for each vertex u ∈ G.V

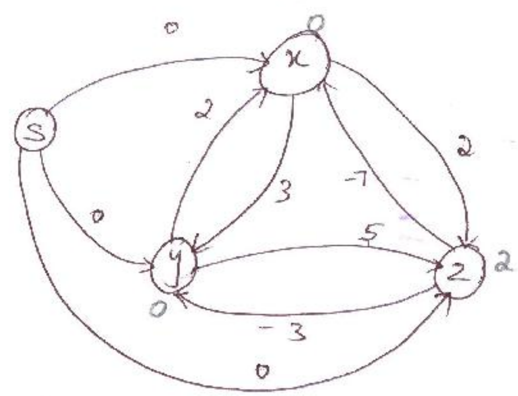
run Dijkstra (G, $\hat{w}$, u) to compute $\hat{\delta}(u, v)$ for all v ∈ G.V

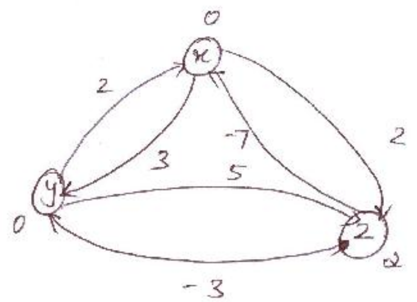
for each vertex v ∈ G.V

$d_{uv} = \hat{\delta}(u, v) + h(v) - h(u)$

return D.

3

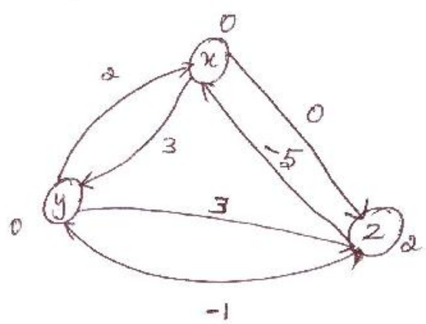


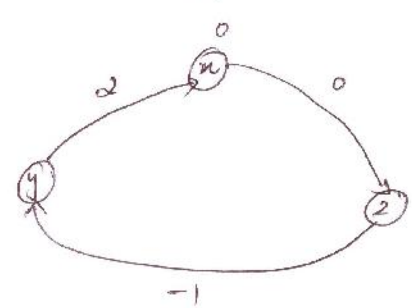


For each edge $e = (u, v)$

$$c'_e = c_e + p_u - p_v.$$

So,





2. a) Define master method. Use master method to give tight asymptotic bounds for the following recurrences. [7]

i) $T(n) = 4T(n/2) + n^2$. ii) $T(n) = 4T(n/2) + n^3$.

Soln.

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

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

$$\boxed{T(n) = \Theta(n^2 \lg n)}$$

$$T(n) = 4T(n/2) + n^3$$

$$\therefore n^3 = \Omega(n^{2+\epsilon}) \text{ and}$$

$$4(n/2)^3 = \frac{1}{2}n^3 \leq cn^3 \;;\; c < 1$$

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

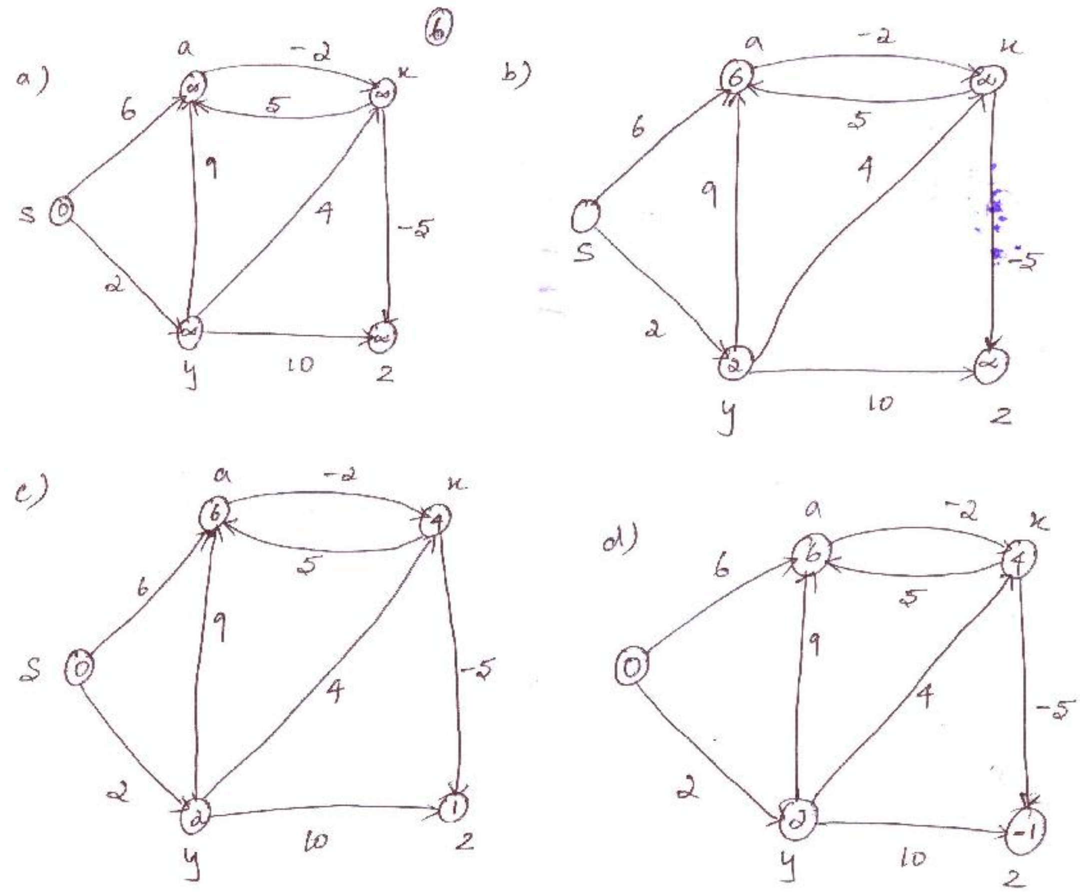
b) Define O, Θ, Ω notation. [3].

Soln.

O-notation:

The function $f(n) = O(g(n))$, iff, there exist positive constant c and n_0 such that,

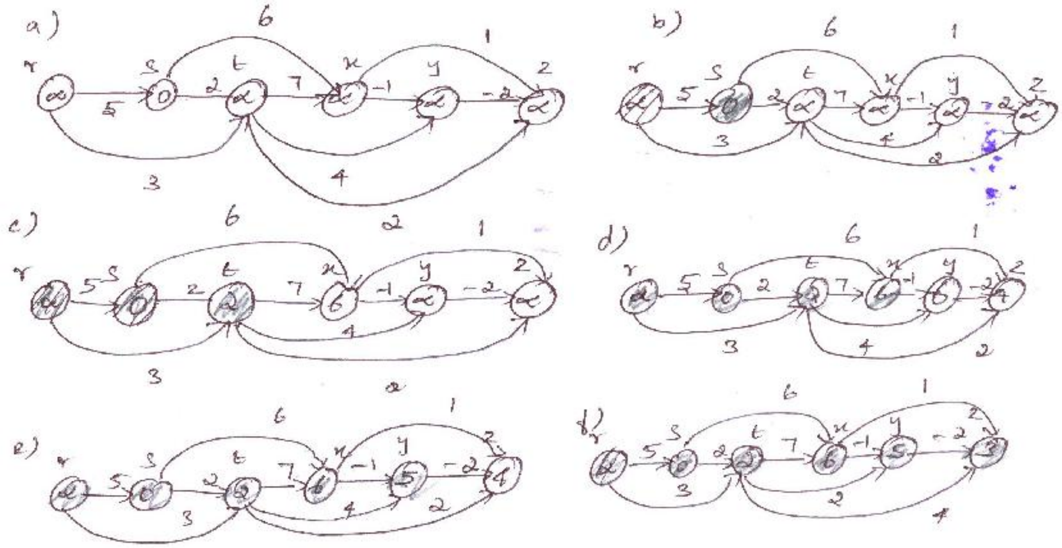
$$f(n) \leq c * g(n) \quad \forall n, n \geq n_0$$



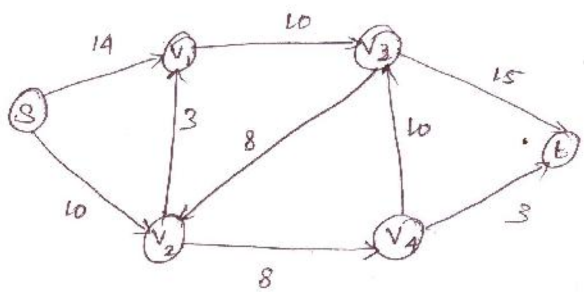
3. a) Write DAG-SHORTEST-PATHS algorithms. Run the same the directed graph of Fig. Q.3 (a). using vertex r as the source. [10].

Soln.

```
DAG-SHORTEST-PATHS (G, W, S)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE (G, S)
for each vertex u, taken in topologically sorted order
    for each vertex v ∈ G.Adj[u]
        RELAX (u, v, w)
```



3. b) Write ford - fulkerson method. Run the same to find maximum flow in graph in fig Q3. b)



Soln.

FORD - FULKERSON - METHOD (G, s, t)

initialize flow f to 0

while there exists an augmenting path p in the residual network G_f.

augment flow f along p

return f.