Daryl Fernandes, 72
AOA, Sem 4

Assignment 1
(11/04/2021)

1)

- It is also known as the table-lookup method.
- It solves the recurrance relation for algorithms based on divide & conquer strategy.
- In this algorithmic strategy, the problem of size n is recursively divided into small sub-problem of size, say, 'n/b'.
- Suppose the number of such sub-problems is 'a'. Then the solution to the orginal problem is obtained by combining the solutions of these 'a' sub-problems which are solved individually by using the same algorithm
- The general recurrance formula for divide & conquer algorithm is given as:

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

where $f(n)$ is the time required for dividing the problem into sub-problems & combining their solutions to obtain the solutions to the orginal problem

a) $T(n) = 3T(n/4) + n \log n$

$\therefore a = 3, b = 4, f(n) = n \cdot \log n$

$n^{\log_b a} = n^{\log_4 3} \approx n^{0.79}$

$n^{\log_b a} < f(n)$

$n^{0.79} < n \log n$

$af(n/b) \leq c * f(n)$

$af(n/4) \leq 3f(n/4) = 3\left(\frac{n}{4} \log \frac{n}{4}\right)$

$= \frac{3}{4} n \log \frac{n}{4} \approx \frac{3}{4} n \log n$

$$c * f(n) = c * n \log n$$
$$\frac{3}{4} n \log n \leq c * n \log n$$

$$\therefore c = \frac{3}{4}$$

Hence, case III of the Master theorem is asatisfied

$$T(n) = O(f(n)) = O(n \log n)$$

b) $T(n) = 0.7T(n) + \log n$

Since, $T(n) = aT(n/b) + f(n)$ where $a \geq 1$ & $b > 1$, this problem has $a \leq 1$ & hence cannot be solved

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

$a = 4$, $b = 3$, $k = 2$, $p = 0$

$a < b^k \Rightarrow$ Case 3;

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

$a = 4$, $b = 2$, $k = 2$, $p = 0$

$\therefore a < b^k \Rightarrow$ Case 3

$\therefore T(n) = \Theta(n^k \log^p n)$

(By Master theorem case 3)

$= \Theta(n^2 \cdot \log^0 n)$

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

2) Theta(Θ)

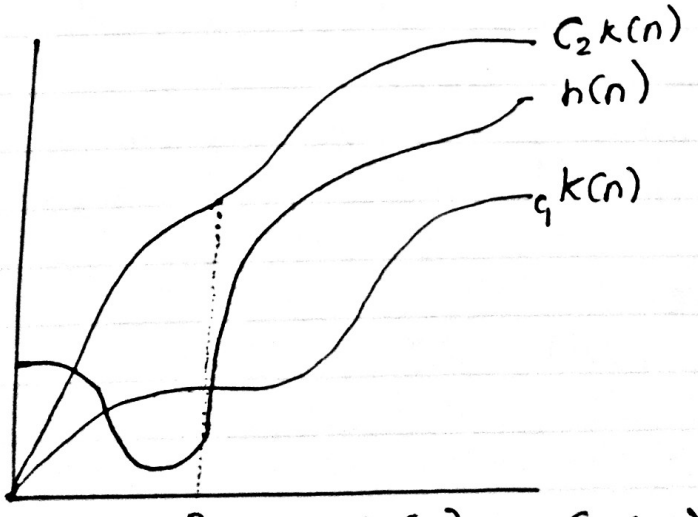
- Let $h(n)$ & $k(n)$ are two funtions. The funtions $h(n) = \Theta(k(n))$ if there exists some positive constants c_1, c_2 & n_0 such that $c_1 * k(n) \leq h(n) \leq c_2 * (k(n))$, $\forall n \geq n_0$

- In other words, the function $h(n) = \Theta(k(n))$ if $h(n) = O(k(n))$ & $h(n) = \Omega(k(n))$, $\forall n \geq n_0$.
- Here $k(n)$ defines both an upper & lower bound on $h(n)$. it defines a tight bound on $h(n)$.

Examples :

i) The function $5n + 16 \neq \Theta(1)$ as $5n + 16 \neq \Theta(1)$ though $5n + 16 = \Omega(1)$.

ii) The function $5n + 16 = \Theta(n)$ as $5n + 16 \leq 21n$ for all $n \geq 1$ & $5n + 16 \geq n$ for all $n \geq 1$
Here $c_1 = 1$, $c_2 = 21$



$h(n) = \Theta(k(n))$

Tight Bound (Θ Notation)

3) Substitution method

It is also known as an Iteration method or guess-and-verify method.

i) Guessing the form of the solution: Here, different values of n are substituted in a recurrence equation to get a sequence.

ii) Verification of the solution: The solution sequence is then verified by mathematical induction.

Based on the way of repeated substitutions it is further divided into ~~by~~ two types.

Recursion Tree

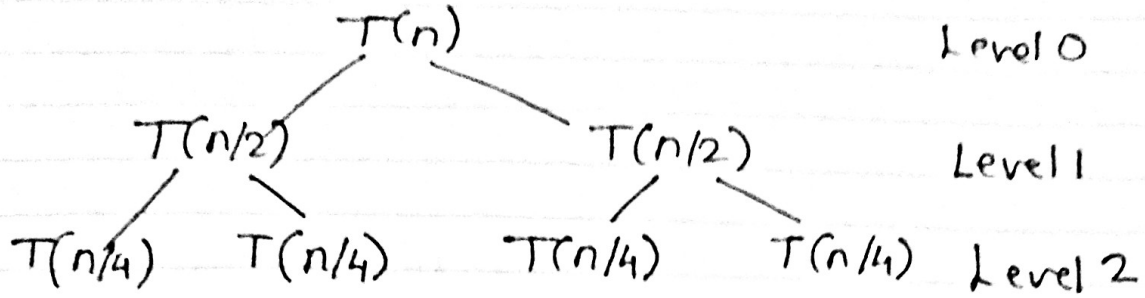
- Like Master's Theorem, Recursion tree is another method for solving the recursion relations.
- A recursion tree is a tree where each node represents the cost of a certain recursive sub-problem
- We sum up the values in each node to get the cost of the entire algorithm.

Example

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

Step 1

Draw a recursion tree based on the given recurrence relation



Step 2 :

Determine cost of each level:

Cost of level 0 = n

Cost of level 1 = $n/2 + n/2 = n$

Cost of level 2 = $n/4 + n/4 + n/4 = n$

Step 3 :

total number of levels

Sub problem at level 0 = $n/2^0$

Sub problem at level 1 = $n/2^1$

Sub problem at level 2 = $n/2^2$

level x = $n/2^x = 1$

$2^x = n$

Take log on b.s.

$x \log 2 = \log n$

$x = \log_2 n$

$\therefore$ Total number of levels in the recursion tree = $\log_2 n + 1$

Step 4 :

level 0 has 2^0 nodes $\therefore$ 1 node

level 1 has 2^1 nodes $\therefore$ 2 node

level 2 has 2^2 nodes $\therefore$ 4 nodes

level-$\log_2 n$ has $2^{\log_2 n}$ nodes ie. n nodes

Step 5

Cost of last level = $n \times T(1) = \Theta(n)$

Step 6

Add costs of all levels

$$T(n) = \{n + n + n + \cdots\} + \Theta(n)$$
$$= n \times \log_2 n + \Theta(n)$$
$$= n \log_2 n + \Theta(n)$$
$$= \Theta(n \log_2 n)$$

4) Selection sort

Algorithm

1 = Set MIN to location 0
2 = Search the minimum element in the list
3 = Swap with value at location MIN
4 = Increment MIN to point to next element
5 = Repeat until list is sorted

Complexity

Iteration number	Number of comparisions	Number of sorted elements	Number of swap
1	(n-1)	1	1
2	(n-2)	1	1
3	(n-3)	1	1
⋮	⋮	⋮	⋮
n-2	2	1	1
n-1	1	2	1

∴ Total number of comparisions $= (n-1) + (n-2) + (n-3) + \cdots + 2 + 1$

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

Total number of swaps $= (n-1) = O(n)$

∴ Total complexity of ~~ste~~ selection sort is $O(n + n^2) = O(n^2)$

5) Algorithm for finding Min-Max

Max-Min-DnC (first, last, max, min)

Input: X[0, n-1] is a global array with n elements are to be found. first and last are two integers such that $1 \leq$ first $\leq$ last $\leq n$.

Output: max is the max number and min is the min number of an array X

```
if (first = last)
    return max := min := X[first];
else if (first = last-1) {
    if X[first] < X[last] {
        max := X[last];
        min := X[first];
    }
    else {
        max := X[first];
        min := X[last];
    }
else {
```

```
        mid := [ (first + last) /2 ];
        Max_Min_DnC (fist, Mid, max, min);
        Max_Min_DnC (mid+1, last, max1, min1);
    if (max1 > max)
        max := max1;
    if (min1 < min)
        min := min1;
    }
}
```

Complexity

$$T(n) = \begin{cases} 2T(n/2) + 2 & ; \text{ if } n>2 \\ 1 & ; \text{ if } n=2 \\ 0 & ; \text{ if } n=1 \end{cases}$$

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

By backward substitution we have:

$T(n) = 2[2T(n/4) + 2] + 2$

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

$= 2^3 T(n/2^3) + 8 + 4 + 2$

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

$\therefore T(n) = 2^k T(n/2^k) + \sum_{i=1}^{k} 2^i$

$T(n) = 2^k T(n/2^k) + 2^{k+i} - 2$ (1)

Let $\frac{n}{2^k} = 2$

$\therefore 2^k = \frac{n}{2}$

Substituting in eqn (i)

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

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

$$= \frac{n}{2} + n - 2$$

$$T(n) = \frac{3n}{2} - 2$$

$$T(n) = O(n)$$

6) Binary Search

Algorithm

i) Compare x with the middle element
ii) If x matches with the middle element we return the mid index.
iii) Else if x is greater than the mid element, then x is can only lie in the right subarray after the mid element
iv) Else (x is smaller) recur for the left half.

Complexity

$$T(n) = \begin{cases} 1 & ; \text{if } n=1 \\ T(n/2) + c & ; \text{if } n>1, c \text{ is constant} \end{cases}$$

~~Solution~~

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

By backward substitutions we get,

$T(n) = [T(n/4) + C] + C$

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

$= T(n/2^3) + C] + 2C$

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

$T(n) = T(n/2^k) + k \cdot C \quad \cdots (1)$

Let $n/2^k = 1$

$\therefore n = 2^k$

$\therefore k = \log_2 n$

Sub. in eqn (1)

$T(n) = T(1) + C \log_2 n$

$T(n) = 1 + C \log_2 n$

... (Sub. $T(1) = 1$ from base condition)

$\therefore T(n) = O(\log_2 n)$

7) $n = 4$

$(P1, P2, P3, P4) = (100, 10, 15, 27)$

$(d1, d2, d3, d4) = (2, 1, 2, 1)$

$\Rightarrow P(1:4) = (100, 27, 15, 10)$

$d(1:4) = (2, 1, 2, 1)$

J'	Assigned time slots	Job under consideration	Action	Total profit
$\emptyset$	none	J[1] = 1	assign to [1,2]	0
1	[1,2]	J[2] = 4	assign to [0,1]	100
[1,4]	[0,1], [1,2]	J[3] = 3	Not feasible reject	100+27 = 127
[1,4]	$\emptyset$, [1,2]	J[4] = 2	Not feasible reject	100+27 = 127

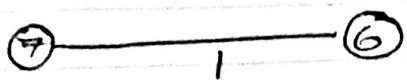
∴ The optimal solution J' = [1,4] with a max profit of 127 units

8) The given graph contains 9 vertices & 14 edges. So the MST formed will be having (9-1) = 8 edges

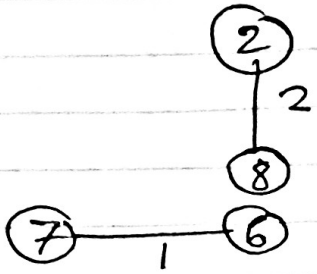
After sorting:

Weight	Src	Dest
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	7	3
7	0	8
8	1	7
8	3	2
9	5	4
10	1	4
14	3	5

1) Pick edge 7-6 : No cycle formed, include it

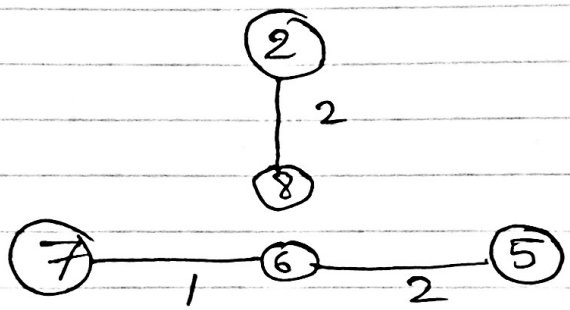


2) Pick edge 8-2 : No cycle formed, include

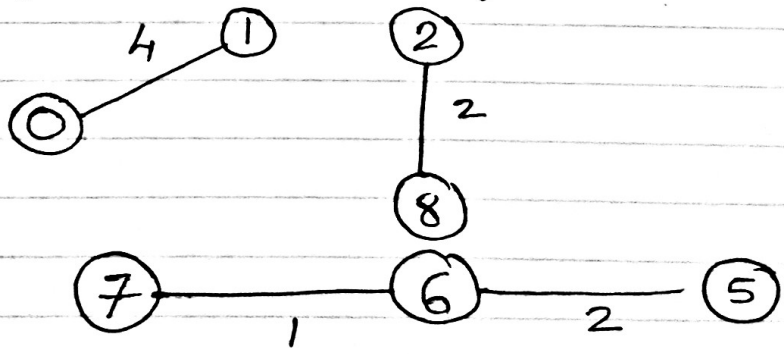


3) ~~Pick edge~~

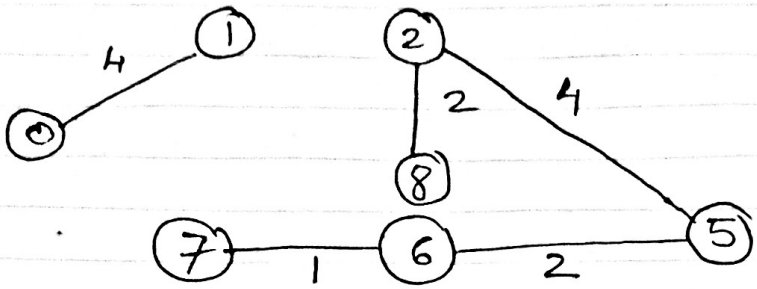
3) Pick edge 6-5 : No cycle formed, include



4) Pick edge ~~0-1~~ No cycle formed, include

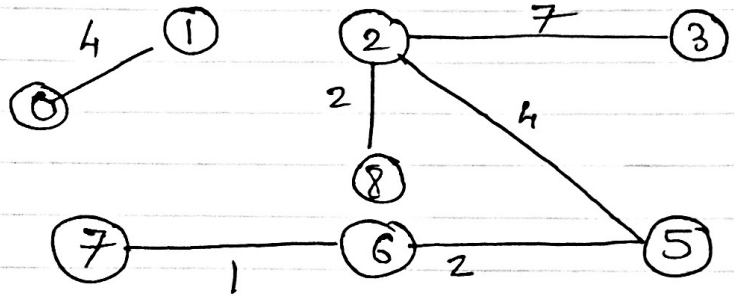


5) Pick edge 2-5 : No cycle formed, include



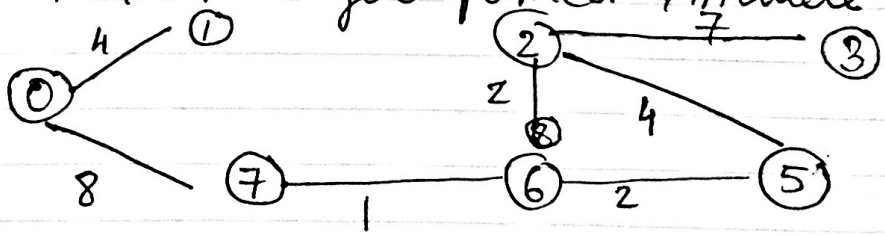
6) Pick edge 8-6 : Cycle formed, discard

7) Pick edge 2-3 : No cycle formed, include



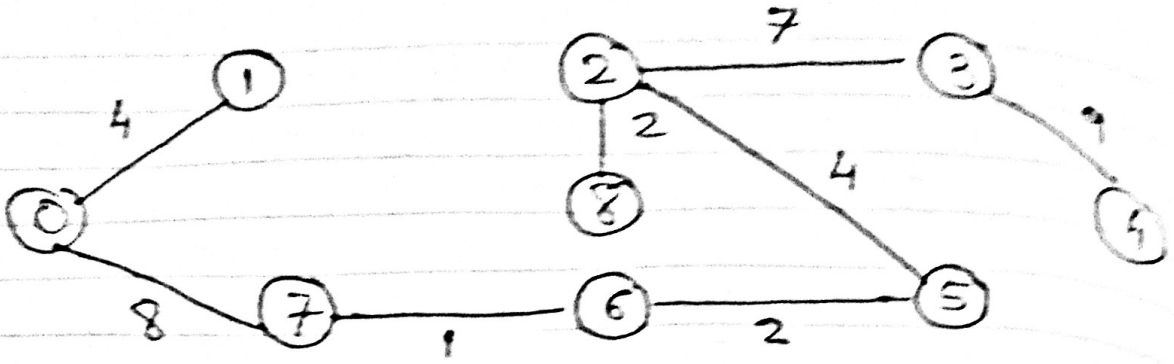
8) Pick edge 7-8 : Since including this edge results in cycle, discard.

9) Pick 0-7 : No cycle formed, include it



10) Pick edge 1-2 : Cycle formed, discard

11) Pick edge 3-4 : No cycle formed include it



Since the ~~two~~ number of edges included equals (V-1), the algorithm stops here.