B. TECH.

FIFTH SEMESTER EXAMINATION 2010-11

DESIGN AND ANALYSIS OF ALGORITHMS

Time: 3 Hours *Total Marks: 100*

Note: (1) *Attempt all questions.*
(2) *All questions carry equal marks.*
(3) *Be precise in your answer.*

1. Attempt any four of the following: **(5 × 4 = 20)**

(a) Determine the asymptotic order of the following functions:

(i) $f(n) = 3x^2 + 5$

Ans. $f(n) = 3x^2 + 5$

order $= \theta(x^2)$

(ii) $f(n) = 2^n + 5n + 3$

Ans. $f(n) = 2^n + 5n + 3$

order $= \theta(n^n)$

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

$= i^2 + i^2 + ...i^2$

$=$ order $= \theta(n^3)$

(iv) $f(n) = 5$

order $= \theta(1)$

(v) $f(n) = n + 7$

order $= \theta(n)$

(b) Why do we use asymptotic notation in the study of algorithm? Explain in brief various asymptotic notations and give their significance.

Ans. Asymptotic notations define the asymptotic running time of an algorithm and their domain are set of real numbers.

(i) Theta Notation (θ): Average case (Tight Bound)

$f(n) = \theta(g(n))$

$\boxed{C_1(g(n)) \le f(n) \le C_2(g(n))}$ for all $n \ge n_0$.

(ii) Omega Notation (Ω) : (Lower Bound) (Best Case)

$f(n) = \Omega(g(n))$

$\boxed{0 \le C(g(n)) \le f(n)}$ for all $n \ge n_0$.

(iii) Big oh Notation: (O) (Upper Bound) (Worst Case)

$f(n) = O(g(n))$

$\boxed{0 \le f(n) \le C(g(n))}$ for all $n \ge n_0$

(c) Solve the following recurrence using Master method:

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

Ans. Solve by Master method:

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

$a = 4;$ $b = 3;$ $f(n) = n^2$

$n^{\log_b^a} = n^{\log_3^4} = n^{1 \cdot 2}$

$f(n) = n^2$

$\therefore f(n) > n^{\log_b^a}$

$\therefore T(n) = \theta(f(n)) = \theta(n^2)$

(d) Discuss any one sorting algorithm for quick sort.

Ans. Counting sort: It assumes that each of n input elements is an integer in the range $0 - k$.

we use 3 arrays:

(a) Input array $A[1...n]$

(b) Output array $B[1...n]$

(c) Temporary array $C[0...k]$

```
Counting sort $(A,B,k)$:
for $i \leftarrow 0$ to $k$
    do $c[i] \leftarrow 0$
for $j \leftarrow 1$ to length $[A]$
    do $C[A[j]] \leftarrow [A[j]] + 1$
for $i \leftarrow 1$ to $k$
    do $[i] \leftarrow c[i] + c[i-1]$
for $j \leftarrow$ length $[A]$ down to 1
    do $B[C[A[j]]] \leftarrow A[j]$
    $c[A[j]] \leftarrow c[A[j]] - 1$
```

(e) Explain and write partitioning algorithm for quick sort.

Ans. Partition (A, P, r)

```
$x \leftarrow A[r]$
$l \leftarrow P - 1$
for $j \leftarrow P$ to $r - 1$
    do if $[A[j]] \le x$
        then $i \leftarrow i + 1$
        exchange $A[i] \leftrightarrow A[j]$
exchange $A[i + 1] \leftrightarrow A[r]$
return $i + 1$
```

(f) Write algorithm to count the number of nodes in a given circular linked list.

Ans. List visited;

```
   List tovisit;
to visit. add (A) || add first node
while (tovisit is not empty)
{
   Node current = visited remove( )
   Array <Node> links = current get Links ();
   for (int i = 0; i < links size ( ); i ++)
   {
   if (! visited. contains (links [i]))
   {
   tovisit. add (links [i]);
}
visited. add (current);
}
}
return visited. size ( );
```

2. Attempt any two of the following:

(10 × 2 = 20)

(a) Explain red-black tree. Prove that red-black tree with n internal nodes has height at most $2 \log_2 (n + 1)$

Ans. A red black tree is a binary tree with over extra bit of storage per node and its the colour which can be either red or black. A binary search tree is a red black tree if it satisfies following properties:

1. Every node is either red or black
2. Root is black
3. Every leaf is black
4. If a node is red, both of its children are black.
5. For each node all the path from the node to descendent leaves contain the same number of black nodes (Black Height).

Proof: Let h be the height of tree. According to property 4 of red black tree, at least half of the nodes on any simple path from root to a leaf, not including the root must be black. Consequently, the black height of root must be at least $h/2$ thus;

$$n \ge 2^{h/2} - 1$$

Moving the 1 to the left hand side and taking logarithm on both sides yield $\lg (n + 1) \ge h/2$ or $h \le 2 \lg (n + 1)$

Hence proved.

(b) Explain and write an algorithm for union of two binomial heaps. Also discuss the time complexity of the same.

Ans. Following procedure unites binomial heap H_1 and H_2 returning a heap.

```
BINOMIAL_HEAP_UNION $(H_1, H_2)$
H $\leftarrow$ MAKE_BINOMIAL_HEAP ( )
head [H] $\leftarrow$ BINOMIAL_HEAP_MERGE $(H_1, H_2)$
Free the object $H_1$ and $H_2$ but not the list they point to
if head $[H]$ = nil.
```

```
then return H.
prev – x ← NiL
x ← head [H]
next – x ← sibling [x]
while next – x ≠ NiL
do if (degree (x) ≠ degree (next_x)) or.
then prev_x ← x
x ← next_x
else if key (x) ≤ key (next _x)
then sibling (x) ← sibling (next_x)
BINOMIAL_LINK (next_x, x)
else if prev_x = NiL
then head [x] ← next_x
else sibling [prev_x] ← next_x
BINOMIAL_LINK (x, next, x)
x ← next_x
next_x ← sibling (x)
return H.
```

Complexity

Total time needed for this algo. is

$$\boxed{O(\log n)}$$

(c) Write short notes on the following:

(i) B-Trees.

(ii) Fibonacci heaps.

Ans.

(i) B-Trees:

B-Tree is a rooted tree with

1. Every node x has:
   (a) $n(x)$, no. of keys
   (b) keys are in non decreasing order
2. Each internal node x contain $n(x) + 1$ pointers.
3. The key key$i[x]$ separate range of keys stored in each subtree
   $k_1 \leq \text{key}_1\ [x] \leq k_2\ \text{key}_2\ [x] \leq \ldots \leq \text{key}_{n(x)}[x] \leq \text{key}\ n[x] + 1$
4. All leaves have same depth.
5. There are upper and lower bounds to number of keys a node can contain.
6. If $n \geq 1$ and then for any n-key B-tree T of height h and min. degree t ≥ 2

$$h \leq \log_t \frac{n+1}{2}.$$

(ii) Fibonacci heap: Like a binomial heap, a fibonacci heap is a collection of min-heap ordered trees. The trees in fibonacci heap are not considered to be binomial trees.

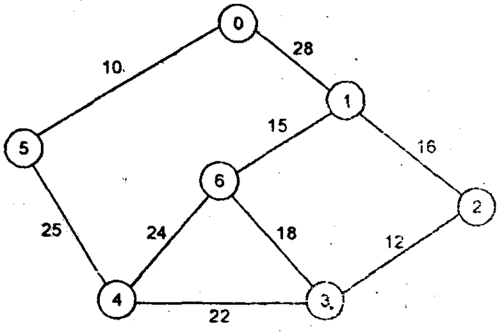
In fibonacci heaps, are rooted but unordered each node (x) contains a pointer to any one of its children. The children of x are linked together in a circular ,doubly circular linked list. Each child y has pointers left (y) and right (y) that point to y's left and right sibling.

The order in which sibling appear in child's list is arbitrary.

3. Attempt any two of the following:

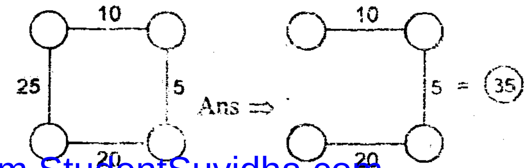
(10 × 2 = 20)

(a) Define minimum cost spanning tree. Write Prim's algorithm to generate a minimum cost spanning tree for any given weighted graph. Generate minimum cost spanning tree for the following graph using Prim's Algorithm.



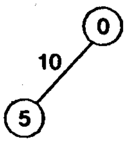
Ans. Let $G = (v, e)$ be on one undirected graph then subgraph $t = (v, E^1)$ is called spanning tree of G if t is a tree

eg;

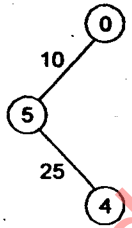


```
MST_PRIM $(G, w, r)$
for each $u \in v(G)$
    do key $[u] \leftarrow \infty$
    $\pi[u] \leftarrow$ NiL
key $[r] \leftarrow 0$
$Q \leftarrow v(G)$
while $Q \neq \phi$
    do $u \leftarrow$ EXTRACT_MIN $(Q)$.
    for each $v \in$ Adj $[u]$
    do if $v \in Q$ and $w(v, u) <$ key $(v)$
        then $\pi[v] \leftarrow u$
        key $[v] \leftarrow w(u, v)$
```

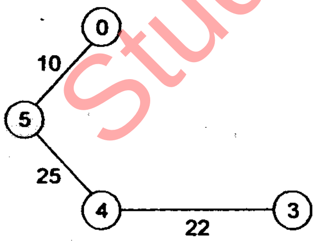
Step 1:



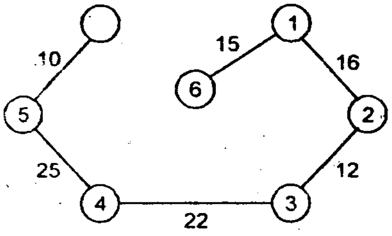
Step 2:



Step 3:



Step 4:



Total cost $= 10 + 25 + 22 + 12 + 16 + 15$
$= 100$

(b) Write an algorithm to find minimum and maximum elements simultaneously from a given list of elements. You are also required to discuss its running time.

Ans. We can devise an algo. that can find both min and max. in $O(n)$ time.

Strategy here is to maintain max. and min. elements seen thus far. Rather than processing each elements of input and comparing it with current max. and min. at cost of 32 comparison on 1 element we do comparisons in pairs.

We compare pairs of elements from input first with each other, then smaller with current min and larger with current max. at a rate of 3 comparisons on 2 elements.

Initial min. and max. depend on n being even or odd. If n is odd first element is both even and odd. If n is even 1 comparison is needed to find min. & max.

$\therefore$ n odd → we do $\lfloor 3n/2 \rfloor$ Comparisons.

n even → we do $3\left\lfloor \frac{n-2}{2} \right\rfloor$ Comparisons.

Complexity: $O(n)$

(c) Explain and write an algorithm for Greedy method of algorithm design. Given 10 activities along with their start and finish time as

$S = \{A_1, A_2, A_3, A_5, A_6, A_7, A_8, A_9, A_{10}\}$
$S_1 = \{1, 2, 3, 4, 7, 8, 9, 9, 11, 12\}$
$F_1 = \{3, 5, 4, 7, 10, 9, 11, 13, 12, 14\}$

Compute a schedule where the largest numbers of activities take place.

Ans. A greedy algorithm makes the choice that looks best at the moment. That is, it makes a locally optimal choice in hope that it will lead to optimal solution.

```
GREEDY_ACTIVITY_SELECTOR $(S, f)$
$n \leftarrow$ length $(S)$
$A \leftarrow \{a_1\}$
```

```
$i \leftarrow 1$
for $m \leftarrow 2$ to $n$
    do if $S_m \ge f_i$
        then $A \leftarrow A \; U \; \{a_m\}$
            $i \leftarrow m$
return $A$.
```

Activity 1 is selected 1 – 3

Activity 2 can't be selected

Activity 3 is selected 3 – 4

Activity 4 is selected 4 – 7

Activity 5 is selected 7 – 10

Activity 6 can't be selected

Activity 7 can't be selected

Activity 8 can't be selected

Activity 9 is selected 11 – 12

Activity 10 is selected 12 – 14

∴ 1 – 3 – 4 – 5 – 9 – 10 is the selection

4. (a) Discuss the dynamic programming solution to longest common subsequence (LCS) problem. Write an algorithm to compute an LCS of two given strings.

Ans. If there are 2 sequences $x = \langle x_1, x_2 \ldots x_m \rangle$ and $y = \{y_1, y_2 \ldots y_n\}$. Then to find max. length for common subsequence of x and y is called LCS problem.

A sequence z is common subsequence of x & y if z is a subsequence of both x and y.

Step 1: Characterization of LCS.

Step 2: A recursive solution

Step 3: Computing length of LCS

```
LCS Length (x, y)
$m$ = length ($x$)
$n$ = length ($y$)
for $i = 1$ to $m$
    do $c[i, 0] = 0$
for $j = 0$ to $n$
    do $c[0, j] = 0$
for $i = 1$ to $m$
    do for $j = 1$ to $n$
        do if $x_i = y_i$
            then $c[i, j] = c[i-1, j-1] + 1$
                $b[i, j]$s “↖”
        else if $c[i-1, j] \ge c[i, j-1]$
            then $c[i, j] = c[i-1, j]$
                $b[i, j]$ = “↑”          0(mn)
        else $c[i, j] = c[i, j-1]$
                $b[i, j]$ = “←”
return $c$ & $b$
```

Step 4: **Constructing LCS**

```
Print_LCS ($b, x, i, j$)
if $i = 0$ or $j = o$
    then return
if $b[i, j]$ = “←”
    then PRINT_LCS ($b, x, i-1, j-1$)
    Print $x_i$
else if $b[i, j]$
    Then print_;CS ($b, x, i-1, j$)
else PRINT_LCS ($b, x, i, j-1$)
```

(b) Describe the Warshall's and Floyd's algorithm for finding all pairs shortest paths.

Ans. In floyd - warshall algorithm, we base result on observation, Under our assumption that vertices of G are $V = \{1, 2 \ldots n\}$ let us consider some subset $\{1, 2 \ldots k\}$ for some k for any pair of vertices $i, j \in v$ consider all paths from i to j whose intermediate vertices are all drawn from $\{1, 2 \ldots k)$ and let p be minimum weight path from among them. Floyd warshall algorithm exploits a relationship between path P and shorter path from i to j with all intermediate vertices in the set $\{1, 2 \ldots k-1\}$. The relationship depends on whether or not k is an intermediate vertex of path P.

```
FLOYD_WARSHALL ($w$)
$n \leftarrow$ rows $[w]$
$D^{(0)} \leftarrow w$
for $k \leftarrow 1$ to $n$
    do for $i \leftarrow 1$ to $n$
        do for $j \leftarrow 1$ to $n$
do $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
return $D^n$
```

(c) Write short notes on the following:

Ans. (i) Graph colouring: A k-colouring of an undirected graph $G = (v, E)$ is a function $c: v \rightarrow \{1, 2...k\}$ such that $c(u) \neq c(v)$ for every edge (u, v) of $\in$. In other words, no. 1, 2...k represent k colours and adjacent vertices must have different colors. Graph coloring problem is to determine minimum no. of coloures needed to color a given graph:

(a) It gives efficient algo to determine a 2 - coloring of graph

(b) Show 3 color is NP-complete

(c) Cost graph coloring as a decision problem.

(ii) Hamiltonian cycles: Formally, a Hamiltonian cycle of an undirected graph $G = (v, E)$ is a simple cycle that contains each vertex in v. A graph that contains a Hamiltonian cycle is said to be Hamiltonain; otherwise it is non-Hamiltonian.

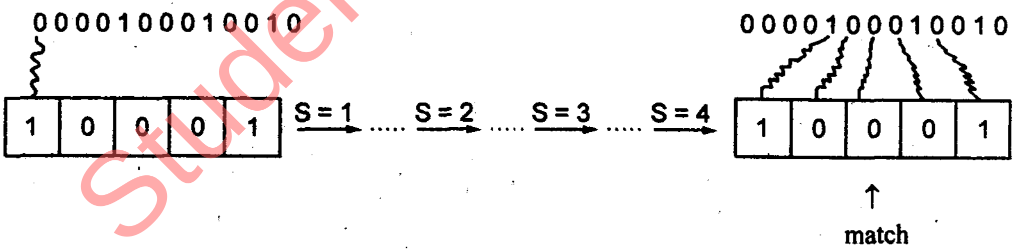
HAM-CYCLE = { <*G*>; *G* is a hamiltonain graph}

Given a problem instance <G>, one possible decision algo. lists all permutations of the vertices of G and then checks each permutation to see if its a hamiltonain path.

5. Attempt any two of the following: **(10 × 2 = 2)**

(a) Show the comparisons the Naive-String matcher makes for the pattern P = {10001} in the text T = {0000100010010} and also show that worst cast time to find the first occurrence of a pattern in a text is $0(n - m + 1)(m - 1)$.

Ans.



Above shown is the way how the Naive string matcher would match the sequence. As shown match would occur after 4 shifts.

```
NAIVE_STRING_MATCHER (T, P)
$n \leftarrow$ length [$T$]
$m \leftarrow$ length [$P$]
for $S \leftarrow 0$ to $n - m$
    do if $P[1...m] = T[S + 1,..S + m]$
        then print "Pattern occurs with shift S".
```

Procedure NAIVE_STRING_MATCHER takes $0(n - m + 1)$ and this bound in tight is worst case.

(b) Explain and wirte Knuth-Morris-Pratt algorithm for pattern matching and also comment on its running time.

Ans. KMP_MATCHER (T, P)

```
n ← length [T]
m ← length [P]
π ← COMPUTE_PREFIX_FUNCTION (P)
q ← 0
for i ← 1 to n
do while q > 0 and P [q + 1] ≠ T [i]
    do q ← π [q]
if P [ q + 1] = T [i]
    then q ← q + 1
if q = m
    then print "Pattern occurs with shift" i − m
    q ← π[q]
```

COMPUTE_PREFIX_FUNCTION (P)

```
m ← length (P)
π (1) ← 0
k ← 0
for q ← 2 to m
    do while k > 0 and P [ k + 1] ≠ P [q]
    do k ← π [k]
    if p [ k + 1] = P [q]
        then k ← k + 1
        π [q] ← k
return π
```

Run Time Complexity:

Running Time of COMPUTE_PREFIX_FUNCTION is 0 (m). A similar analysis using q as potential function show that matching time of KMP_MATCHER is 0 (n).

(c) Write short notes on the following:

(i) Fast Fourier Transform

(ii) NP-Completeness

Ans. (i) Fast Fourier Transform: By using a method known as *FFT* which takes an advantage of special properties of complex roots of unity, we can compute DFT_n (a) in time $0(n \log n)$ as opposed to $0(n^2)$.

FFT method employs a divide and conquer strategy using even index and odd index coefficients of $A(x)$ separately to define 2 new polynomials $A^{[0]}(x)$ and $A^{[1]}(x)$ of degree bound $n/2$.

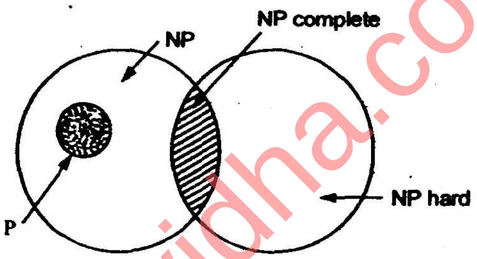
$$A^{[0]}(x) = a_0 + a_2x + a_4x^2 + \dots a_{n-1} x^{n/2-1}$$

$$A^{[1]}(x) = a_1 + a_3x + a_5x^2 + \dots a_{n-1} x^{n/2-1}$$

$$A(x) = A^{(0)} x^2 + x A^{(1)} x^2.$$

(ii) NP-completeness: A problem that is NP-Complete has the property that it can be solved in polynomia time if and only if all other NP-Complete problems can also be solved in polynomial time.

If an NP hard problem can be solved in polynomial time, then all NP_Complete problem can be solved in polynomial time. All NP-Complete problem are NP hard, but some NP hard problem are not known to be NP-Complete.


NP complete
NP
NP hard
P