B.E.

Fifth Semester Examination, MAY 2009

Analysis And Design of Algorithms (CSE-305-E)

Note : Attempt any five questions. All questions carry equal marks.

Q. 1. (a) Write and explain different types of asymptotic notations with suitable example.

Ans. The notations we use to describe the asymptotic running time of an algorithm are defined in terms of functions whose domains are the set of natural number $N = \{0, 1, 2, \ldots\}$. Such notations are convenient for describing the worst-case running time function $T(n)$, which is usually defined only on integer input sizes.

It is sometimes convenient, however, to abuse asymptotic notation in a variety of ways.

Example : The notation is easily extended to the domain of real number, or alternatively, restricted to a subset of the natural numbers.

$O(g(n)) = \{f(n)$: there exist positive constants c_1 and c_2 and number such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0\}$

Q. 1. (b) What is the recurrence relation? Solve the following relation by recursive method

$T(n) = 2T(n/2 + 17) + n$

Ans. When an algorithm contains a recursive case to itself, its running time can often be described by a recurrence equation or recurrence which describes the over all running time on a problem of size n in terms of the running time on smaller inputs.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le c \\ aT(n/b) + D(n) + C(n) & \text{otherwise.} \end{cases}$$

$$T(n) \le 3T(\lfloor n/4 \rfloor) + cn^2$$

$$\le 3d\lfloor n/4 \rfloor^2 + cn^2$$

$$\le 3d(n/4)^2 + cn^2$$

$$= \frac{3}{16} dn^2 + cn^2$$

$$= dn^2$$

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

Q. 2. (a) Explain the algorithm of Quick Sort and compare time complexity with Heap sort.

Ans. Quick sort, like merge sort, is based on divide 2 conquer paradigm introduced. Here is the three step divide and conquer process for storing a typical subarray $A[P \ldots\ldots\ldots\ldots r]$

Divide : Partition (rearrange) the array $A[P \ldots\ldots r]$ into 2 subarrays $A[P \ldots\ldots q-1]$ & $A[q+1 \ldots\ldots r]$

Such that each element of $A[P \ldots\ldots q-1]$ is less than or equal to $A[q]$, which is in turn less than or equal to each element of $A[q+1 \ldots\ldots r]$

Conquer : Sort the 2 subarrays $A[P.....q-1]$ & $A[q+1.....r]$ by recursive calls to quick sort.

Combine : Since the subarrays are sorted in place, no work in needed to combine them : the entire array $A[P.....r]$ is now sorted.

Quick Sort : (A, p, r).

If $p<r$

Then $q \leftarrow$ partition (A, p, r)

Quick sort $(A, p, q-1)$

Quick sort $(A, q+1, r)$

Q. 2. (b) What is Divide and Conquer strategy? Design Merge sort algorithm using Divide and Conquer strategy. Also give its recurrence relation.

Ans. Many useful algorithms are recursive in structure. to solve a given problem, they call themselves recurseilvely over or more times to deal with closely related sub problems. The divide and conquer paradigm involves three steps at each level of the recurrsion.

Divide : The problem into a number of subproblems conquer the subproblems by solving them recurseively combine the solutions to the sub problems into the selection for the original problem.

Merge Sort :

$A(A, p, q, r)$

$n_1 \leftarrow q-p+1$

$n_2 \leftarrow r-q$

Create arrays $L[1.....n_1+1]$ and $R[1.....n_2+1]$

For $i \leftarrow 1$ to n_1

Do $L[p] \leftarrow A[p+i-1]$

For $j \leftarrow 1$ to n_2

Do $R[i] \leftarrow A[q+j]$

$L[n_1+1] \leftarrow \infty$

$R[n_2+1] \leftarrow \infty$

$i \leftarrow 1$

$j \leftarrow 1$

For $k \leftarrow p$ to r

Do if $[L(i)] \le R[j]$

Then $A[k] \leftarrow L[i]$

$i \leftarrow i+1$

Else $A[k] \leftarrow R[j]$

$j \leftarrow j+1$

Q. 3. (a) Write and explain Kruskal's algorithm to find the minimum spanning tree from a graph with example and derive its time complexity.

Ans Kruskal's algorithm is based directly or the generic minimum spanning tree algorithm. It finds a safe edge to add to the growing forest by finding of all the edges that connect any 2 tree s in the forest, an edge (u, v) of least weight. Let c_1 & c_2 denote 2 trees that are connected by (u, v).

Since(u, v) must be a light edge connecting c_1 to some other tree.

MST-KRUSKAL (G, ω)

$A \leftarrow \varphi$

For each vertex $v \in V[G]$

do MAKE-SET (v)

Sort the edges of E into non decreasing over by weight w.

For each edge (u, v) and E taken in non-decreasing order by weight.

Do if FIND-SET $(u) \neq$ FIND$-$SET(v)

Then $A \leftarrow A \cup \{(u, v)\}$

UNION (u, v)

Return A

Q. 3. (b) Explain the greedy approach for algorithm design? Devise a solution for fractional Knapsack using greedy approach? Give its time complexity.

Ans. Algorithms for optimization problems typically go through a sequence of steps with a set of choices at each step. For many optimization problems, using dynamic programming to determine the best choices is over kill; simpler, more efficient algorithm will do. A greedy algorithm always makes the choice that looks best at the moment.

Greedy algorithms donot always yield optimal solutions, but for many problems they do. The greedy method is quite powerful and works well for and wide range of problems.

A Recursive Solution : The second step in developing a dynamic-programming selection is to recursively define the value of an optimal selection. For the activity selection problem, we let $c(i, j)$ be the number of activities is a maximum size subset of mutually compatible activities in S_{ij}.

$$C[i, j] = c[i, k] + c[k, j] + 1$$

$$c[i, j] = \begin{cases} 0 & \text{if } S_{ij} = \varphi \\ \max\{c[i,k] + c[k,j] + 1 & \text{if } S_{ij} \neq \varphi\, i < k < j \end{cases}$$

Q. 4. (a) Describe Dynamic Programming method with a suitable example.

Ans. Dynamic programming like the divide and conquer method. Solves problems by combining the solutions to subproblems.

A dynamic programming algorithm solves every subproblem just once and theis saves its answer in a table, thereby avoiding the work of recomputing the answer every time the sub subproblem is encountered.

The development of a dynamic programming algorithm can be broken into a sequence of 4 steps :

(i) Characterize the structure of an optimal selection.

(ii) Recursively define the value of an optimal selection.

(iii) Compute the value of an optimal selection in a bottom-up-fashion.

(iv) Construct an optimal selection from computed information.

Q. 4. (b) Write and explain algorithm for finding Transitive Closure using Dynamic approach with suitable example and derive its time complexity.

Ans. Given a directed graph $G = (V, E)$ with vertex. Set $V = \{1, 2, \ldots\ldots\ldots n\}$ we may wish to find out whetner there us a parts in G from i to j for all vertex pairs $i, j \in V$.

The transitive closure of G is defined as the graph $G^* = (V, E)^*$, where

$$E^* = \{(i, j)^*; \text{there is a parts from vertex } i \text{ to vertex } j \text{ in } G\}$$

$$t_{ij}^{(0)} = \begin{cases} 0 \text{ if } i \neq j \text{ and } (i,j) \notin E \\ 1 \text{ if } i = j \text{ or } (j,j) \in E \end{cases}$$

& for $k \geq 1$

Transitive colsure (G)

$n \leftarrow |V[G]|$

For $i \leftarrow 1$ to n

Do for $j \leftarrow 1$ to n

Do if $i = j$ or $(j, i) \in E(G)$

Then $t_{ij}^{(0)} \leftarrow 1$

Else $t_{ij}^{(0)} \leftarrow 0$

For $K \leftarrow 1$ to n

Do for $i \leftarrow 1$ to n

Do $t_{ij}^{(K)} \leftarrow t_{ij}^{(k-1)} \vee (t_{jk}^{(k-1)} \wedge t_{kj}^{(k-1)})$

Return $T^{(n)}$

Q. 5. Discuss Branch and Bound strategy by solving Travelling Salesman problem for any graph and analyze the time complexity for the same.

Ans. In the travelling-salesman problem, we are given a complete undirected graph $G = (V, E)$ that has ω non-negative integer cost $c(u, v)$ associate with each edge $(u, v) \in E'$ and we must find a hamiltonian cycle(a four) of G with minimum cost. As an extension of the notation, let $c(A)$ denote the total cost of the edges in the subset $A \subseteq E$

$$C(A) = \sum_{(u,v) \in A} c(u, v)$$

APPROX-TSP-TOVR (G, c)

(i) Select a vertex $r \in V[G]$ to be a root vertex.

(ii) Compute a minimum spanning tree T for G from root or using MST-PR/M (G, c, r).

(iii) Let L be the list of vertices visited in a preceded tree walk of J.

(iv) Return the hamiltonian cycle H that visits the vertices in the order L.

Theorem : If $P \neq NP$, then for any constant $P \geq 1$, there is no polynomial time approximation algorithm with approximation rated and for the general travelling salesman problem.

Q. 6. (a) Differentiate between Backtracking and Branch and Bound method.

Ans. Another inversing property of depth first search is that the search call be used to classify the edges of the input graph $G = (V, E)$. This edge classifications can be used to glean important information about a graph.

We can define graph four edges in terms of depth first forest G_π produced by a depth first search on G.

(i) Tree edges are edges in the depth first forest G_π. Edge (u, v) is a tree edge if v was first discovered by exploring edge (u, v).

(ii) Back edges are those edges (u, v) connecting a vertex u to an ancestor v in a depth first tree.

(iii) Forward edges are those non tree edges (u, v) connecting a vertex u to a descendent v in a depth first tree.

(iv) **Cross edges** are all other edges. Theyt can go between vertices in the same depth first tree, as long as one vertex is not an ancestor of the other or they can go between verticles in different depth first trees.

Q. 6. (b) Give an algorithm for graph coloing problem using Backtracking and analyze the time complexity of the same.

Ans. A K colouring of an udirected graph $G = (V, E)$ is a function $c : V \rightarrow \{1, 2, \ldots\ldots K\}$ such that $c(u) \neq c(v)$ for every edge $(u, v) \in E$. In other words, the numbers 1, 2, $\ldots\ldots K$ represent the K colors and adjacent vertices must have different colors.

(i) Give an efficient algorithm to determine the 2 coloring of a graph if one exists.

(ii) Cost the graph-coloring problem as a decision problem.

(iii) Let the language 3 color be the set of graphs that can be 3-colored.

Graph caloring problem is to determine the minimum number of color needed to color a given graph.

Q. 7. (a) Explain heap sort with its complexity? What is running time of heap sort when list is already sorted? Sort the following list using heap sort :

14, 8, 15, 6, 3,. 20, 22, 28, 10, 40

Ans. The heap sort algorithm starts by using BUILD-MAX-HEAP to build a max heap on the input array

$$A[1\ldots\ldots n]$$

Where, n = length $[A]$.

Since the maximum element of an array is stored at the root A[1], it can be put into its correct final position by exchanging it with $A[n]$.

HEAP SORT $[A]$

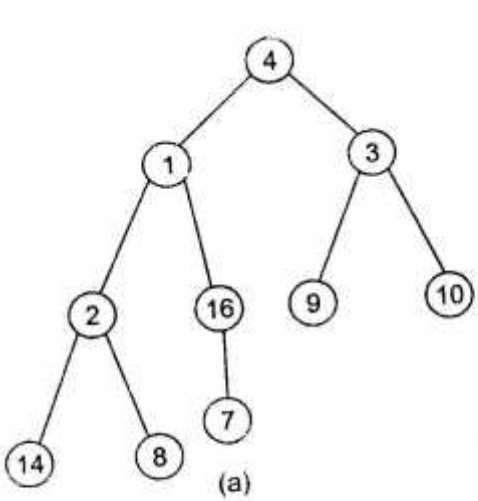
BUILD-MAX-HEAP(A)

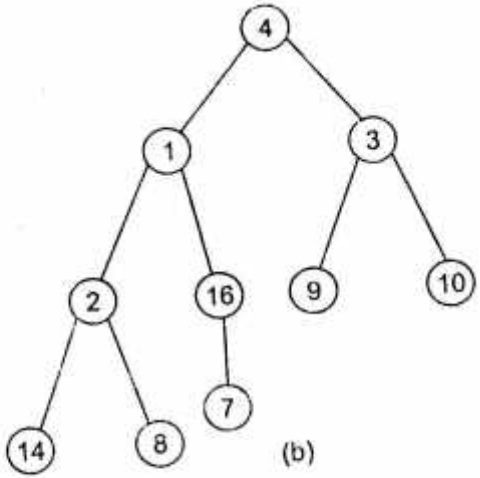
For $P \leftarrow$ length [A] down to 2

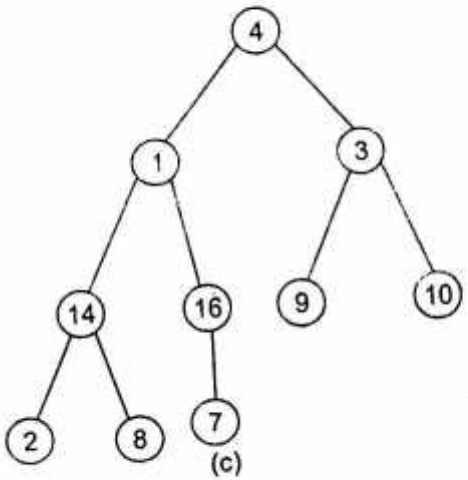
Do exchange $A[1] \leftrightarrow A[i]$

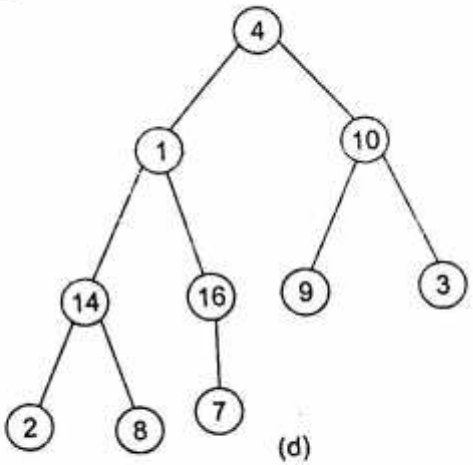
Heap-size $[A] \leftarrow$ heap-size $[A] - 1$

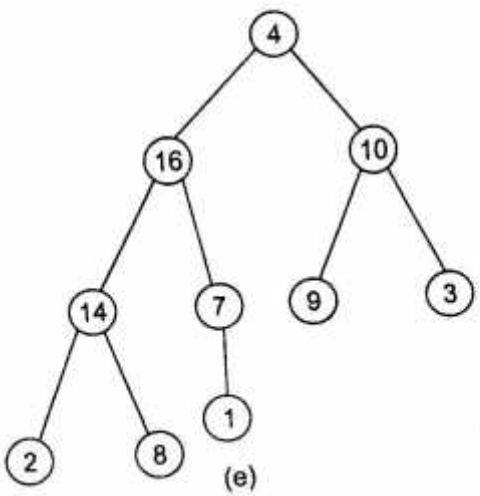
MAX-HEAPIFY $(A, 1)$

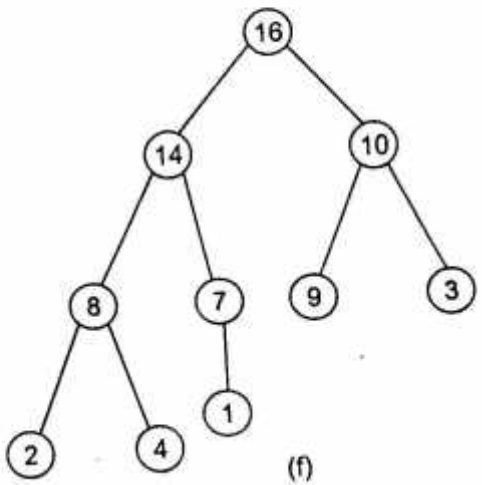

4
1
3
2
16
9
10
14
8
7
(a)


4
1
3
2
16
9
10
14
8
7
(b)


4
1
3
14
16
9
10
2
8
7
(c)


4
1
10
14
16
9
3
2
8
7
(d)


4
16
10
14
7
9
3
2
8
1
(e)


16
14
10
8
7
9
3
1
2
4
(f)

Q. 7. (b) Explain the implementation of Priority queue using max-heap? Give the procedure for HEAP, INSERT and HEAP-EXTRACT-MAX for implementation of Priority Queue? Give the time complexity also.

Ans. A priority queue is a data structure for maintainty a set S of element, each with an associated value called a key. A max priority queue supports the following operation.

— INSERT

— MAXIMUM (S)

— EXTRACT-MAX (S)

— INCREASE KEY (S, x, k)

HEAP-MAXIMUM (A)

Return A[1]

HEAP EXTRACT-MAX (A)

if heap-size [A] < 1

then error "heap under flow"

max ← A[1]

$A[1] \leftarrow A$[heap-size[A]]

heap size [A] ← heap size [A] − 1

MAX HEAPIFY (A, 1)

Return MAX

Q. 8. Explain any four of the following :

(a) Huffman codes

(b) Adjacency List and Adjacency Matrix Representation of graph

(c) Optimal Binary Search Trees

(d) NP-Complete Problems

(e) Fixed and variable sized tuple formation state space trees

Ans. (a) Huffman Codes : Huffman codes are a widely used and very effective technique for compressing data: savings for 20% to 90% are typical, depending on the characteristics of the data being compressed.

There are many ways to represent such a file of information code, wherein each character is represented by a unique binary string. If we use a fixed length code, we need 3 bits to represent six characters

$a = 000$

$b = 001$

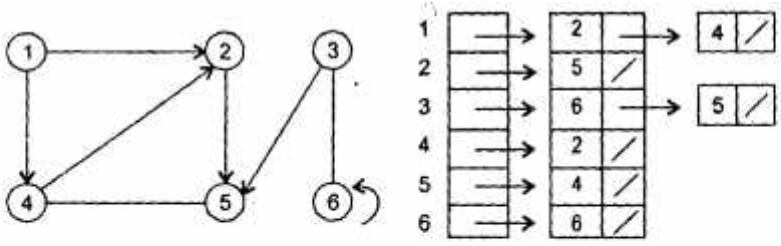
.....

$f = 101$

A variable length code can do considerably better than fixed length code.

(b) Adjacency List and Adjacency Matrix Representation of Graph : The adjacency list representation of graph $G = (V, E)$ consists of an array Adj of $|V|$ lists, one for each vertex in V.

For each $u \in V$, the adjacent list $Ad_j(u)$ list contains the vertices v such that there is an edge $(u, v) \in E$. That is, $Ad_j(u)$ consists of all the vertices adjacent to u in G.



(a) (b)

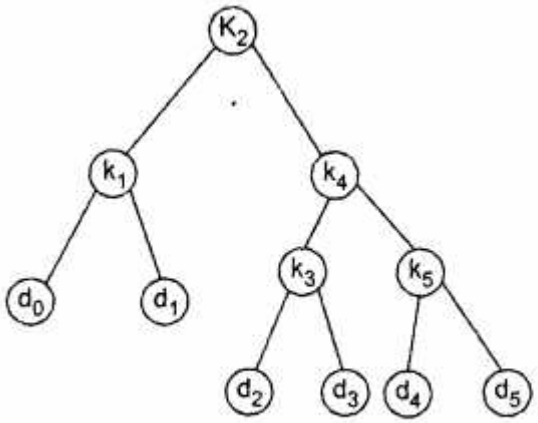
	0	1	2	3	4	5	6
1	0	1	0	0	0	0	1
2	0	0	0	0	1	1	0
3	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0

(c)

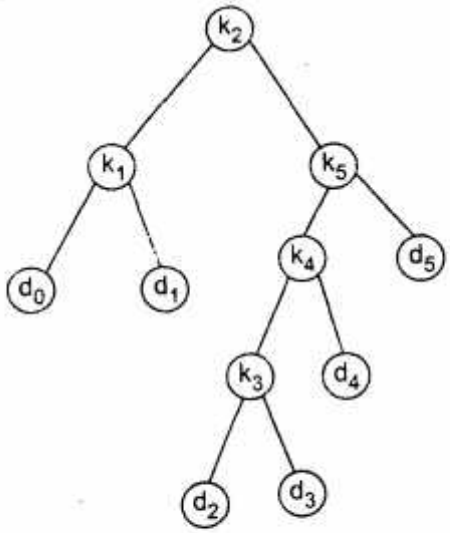
Adjacency matrix representation of graph $G=(V,E)$, we assume that the vertices are numbers 1, 2, ...$|V|$ in some arbitrary manner. Then the adjacency matrix representation consists of a $|V|\times|V|$ matrix $A=(a_{ij})$ such that

$$a_{ij}=\begin{cases}1 & \text{if } (i,j)\in E\\ 0 & \text{otherwise}\end{cases}$$

(c) Optimal Binary Search Trees :



(a)



(b)

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$$

Node	Depth	Probability	Contribution
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	2	0.05	0.15
k_4	1	0.10	0.20
k_5	2	0.20	0.60
d_0	2	0.05	0.15
d_1	2	0.10	0.30
d_2	3	0.05	0.20
d_3	3	0.05	0.20
d_4	3	0.05	0.20
d_5	3	0.10	0.40
Total			**2.80**

(d) NP-Complete Problems : NP completeness proofs are somewhat alike. Many problems of interest are optimization problem. in which each feasible, solution has an associated value and all wish to find the feasible solution with the best value. For example, in a problem that we call shorted path, all are given an undirected graph G and vertices G and vertices *u* and *v* wish to find the path from *u* to *v*.

(e) Fixed & Variable Sized Tuple Formation State Space Trees : The space needed by the algorithm is seen to be the sums of following components.

A fixed part that is independence of the characteristics of inputs and outputs. This part typically includes the instruction space, space for simple variables and fixed size component variables, space for constants and so on.

A variable part that consists of the space needed by component variables whose size is dependent on the particular problems instance being solved, the space needed by reference variables and the recursion stack space.