SRN | | | | | | | | | | |

PES University, Bengaluru

(Established under Karnataka Act 16 of 2013)

END SEMESTER ASSESSMENT(ESA) B. TECH 4th SEMESTER CSE

UE18CS251

Design and Analysis of Algorithms

Time: 3 Hrs | Answer All Questions | Max Marks: 100

1	a	Prove that if $t1(n) \in \Omega(g1(n))$ and $t2(n) \in \Omega(g2(n))$ then $t1(n) + t2(n) \in \Omega(\max\{g1(n), g2(n)\})$ Solution Since $t1(n) \in \Omega(g1(n))$, ⇨ $t1(n) \geq c1g1(n)$ for all $n \geq n1$. Since $t2(n) \in \Omega(g2(n))$, ⇨ $t2(n) \geq c2g2(n)$ for all $n \geq n2$. Let $c = \min\{c1, c2\}$ and consider $n \geq \max\{n1, n2\}$ $t1(n) + t2(n) \geq c1g1(n) + c2g2(n)$ $\geq cg1(n) + cg2(n) = c[g1(n) + g2(n)]$ $\geq c\max\{g1(n), g2(n)\}$. Hence $t1(n) + t2(n) \in \Omega(\max\{g1(n), g2(n)\})$, $c= \min\{c1, c2\}$ and $n_0=\max\{n1, n2\}$, respectively	4
	b	Explain the method of comparing the order of the growth of 2 functions using limits. Compare order of growth of (i) $\log_2 n$ and sqrt(n) ii) $(\log_2 n)^2$ and $\log_2 n^2$ Solution $$\lim T(n)/g(n) = \begin{cases} 0 & \text{order of growth of } T(n) < \text{order of growth of } g(n) \\ c > 0 & \text{order of growth of } T(n) = \text{order of growth of } g(n) \\ \infty & \text{order of growth of } T(n) > \text{order of growth of } g(n) \end{cases}$$ $\log_2 n \quad =(O(\text{ sqrt}(n)))$ $(\log_2 n)^2 = \Omega(\log_2 n^2)$	2+4

SRN

	c	Solve the following recurrence relations using substitution method f(n) = $\begin{cases} f(n-1) + n & \text{for } n > 0 \\ 0 & \text{for } n = 0 \end{cases}$ x(n) = 3x(n-1) for n>1, x(1) = 4 x(n)= x(n/2) + n for n>1 , x(1) = 1, n = 2^k Solution i) O(n2) ii) O(3n) iii) O(n)	6
	d	Rank the following functions in the order of increasing asymptotic growth(log base is 2) n^2, n!, (log n)!, nlogn, 2logn, e^n, 5 Solution 5 < 2 logn < nlogn, < n^2 < e^n < (log n)! < n!	4
2	a	Design a Θ(n) algorithm to count the number of substrings that start with an A and end with a B in the given text. (For example, there are 9 such substrings in DAAXBABAGBD) Solution number o substrings that end with a B at a given position i (0 < i ≤ n-1) in the text is equal to the number of A's to the left of that position. ⇨ Initialize the number of A's encountered and the number of desired substrings encountered to 0. ⇨ Scan the text from left to right. When an A is encountered, increment the number of A's encountered. When a B is encountered, increment the number of desired substrings encountered by the current value of the number of A's encountered. When the text is exhausted, return the last value of the number of substrings encountered. Since, we do a linear pass on the text and spends constant time on each of its character, Time complexity is Θ(n)	6

SRN

b Apply Insertion Sort to sort the list A L G O R I T H M S in alphabetical order. **4**

Solution

A L G O R I T H M S
A | L G O R I T H M S
A L | G O R I T H M S
A G L | O R I T H M S
A G L O | R I T H M S
A G L O R | I T H M S
A G I L O R | T H M S
A G I L O R T | H M S
A G H I L O R T | M S
A G H I L M O R T | S
A G H I L M O R S T

c Analyze the best-case and worst-case time complexity of Insertion sort. **4**

Solution

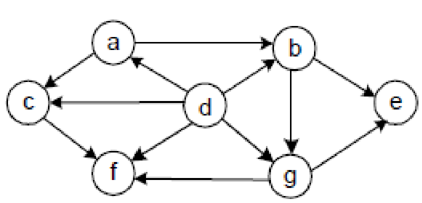
Best Case (if the array is already sorted): the element v at A[i] will be just compared with A[i-1] and since A[i-1] ≤ A[i] = v, we retain v at A[i] itself and do not scan the rest of the sequence A[0…i-1]. There is only one comparison for each value of index i.

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

Worst Case (if the array is reverse-sorted): the element v at A[i] has to be moved all the way to index 0, by scanning through the entire sequence A[0…i-1].

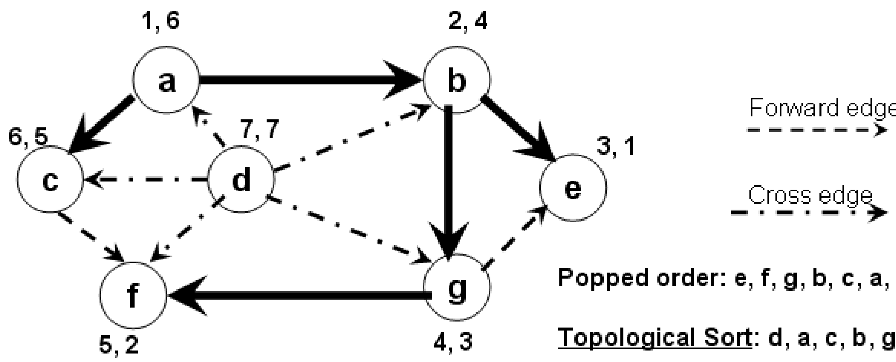
$$\boxed{C(n) = \sum_{i=1}^{n-1} \sum_{j=i-1}^{0} 1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \Theta(n^2)}$$

d Explain how to use DFS to solve topological sorting problem. Apply DFS solve the topological sorting problem for the following directed graph **3+3**



SRN

Solution



Popped order: e, f, g, b, c, a, d

<u>Topological Sort</u>: d, a, c, b, g, f, e

---

3 a **Develop a divide and conquer algorithm to find the position of the largest element in an array of *n* integers. Write the recurrence equation for the number of comparisons and hence derive the time complexity of the algorithm.** 4+4

Solution

- divide an array of size n into two sub-arrays of size n/2 each
- Find the index of the maximum element within the sub-arrays using a recursive approach.
- compare the values of the elements that are the largest in the two sub-arrays and return the largest

Call **Algorithm** $MaxIndex(A, 0, n-1)$ where

Algorithm $MaxIndex(A, l, r)$
//Input: A portion of array $A[0..n-1]$ between indices l and r $(l \leq r)$
//Output: The index of the largest element in $A[l..r]$
if $l = r$ **return** l
else $temp1 \leftarrow MaxIndex(A, l, \lfloor (l+r)/2 \rfloor)$
    $temp2 \leftarrow MaxIndex(A, \lfloor (l+r)/2 \rfloor + 1, r)$
    if $A[temp1] \geq A[temp2]$
        return $temp1$
    else return $temp2$

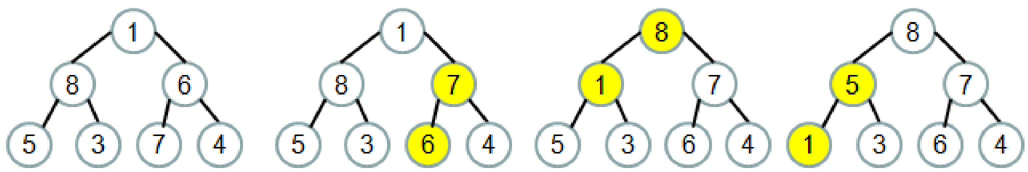
Recurrence Equation

$C(n) = 2\,C(n/2) + 1$ for $n > 1$ and $C(1) = 0$

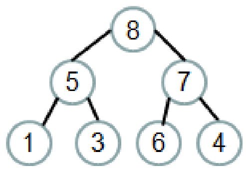
using Master Theorem,
$a = 2$; $b = 2$; $d = 0$
$a > b^d$.
Hence, $C(n) = \Theta(n^{\log_2 2}) = \Theta(n)$.

SRN

b **Sort the array [1, 8, 6, 5, 3, 7, 4] using Heap sort (Use bottom up Heap construction and show all steps). What is time complexity of Heap Sort?** 4+2



Proper (Initial) Heap



Sorting the Array

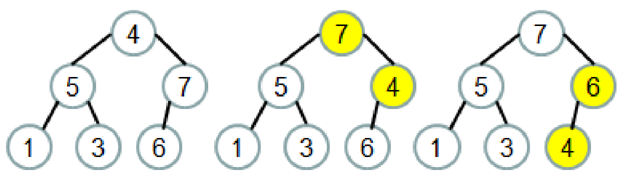
Initial Array (satisfying the heap property)

-10000 8 5 7 1 3 6 4

Array sorting in progress

-10000 7 5 6 1 3 4 8

Iteration # 1: Remove key 8



Time complexity

THeapsort(n) = THeap(n) + TSort(n)

THeapsort(n) ∈ max{**ϴ**(n), **ϴ**(n logn)}

THeapsort(n) ∈ **ϴ**(n logn)

c Answer the following with respect to Quick Sort algorithm justify your answer 6

a**.** Are strictly decreasing arrays the worst-case input, the best-case input, or neither?

b. if pivot element is chosen as the median of the first, last, and middle, are increasing arrays the worst-case input, the best-case input, or neither?

c. Is quicksort inplace sorting algorithm

Solution

a. Strictly decreasing arrays constitute the worst case because all the splits will yield one empty subarray.

b. For either an increasing or decreasing subarray, the median of the first, last, and middle values will be the median of the entire subarray. Using it as a pivot will split the subarray in the middle which results in smallest number of key comparisons

c. Yes quicksort doesn't use any auxiliary space

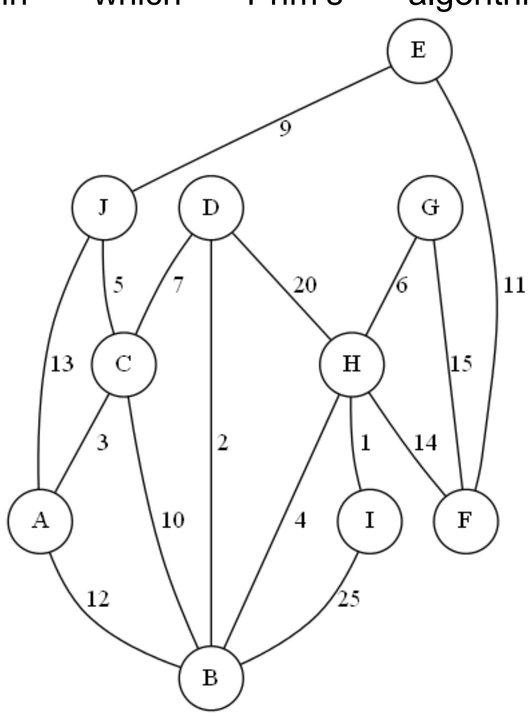
SRN

4 a Explain Greedy technique based algorithm to solve single source shortest path problem. Analyse the run-time complexity of the algorithm **8**

```
Begin Algorithm Dijkstra (G, s)
    For each vertex v ∈ V                                                      } O(V) time
        d [v] ← ∞ // an estimate of the min-weight path from s to v          }
    End For
    d [s] ← 0                                                                    O(V) time to Construct a Min-heap
    S ← Φ // set of nodes for which we know the min-weight path from s
    Q ← V // set of nodes for which we know estimate of min-weight path from s
    While Q ≠ Φ  ←——— done |V| times = O(V) time
        u ← EXTRACT-MIN(Q)  ←—— Each extraction takes O(logV) time
        S ← S U {u}
        For each vertex v such that (u, v) ∈ E  } done O(E) times totally
            If v ∈ Q and d [v] > d [u] + w (u, v) then  }
                d [v] ← d [u] + w (u, v)                }  It takes O(logV) time when done once
                Predecessor (v) = u                     }
            End If
        End For
    End While
End Dijkstra
```

Overall Complexity: O(V) + O(V) + O(VlogV) + O(ElogV)
Since |V| = Ω(|E|), the VlogV term is dominated by the ElogV term. Hence, overall complexity = O(|E|*log|V|)

b Use Prim's algorithm starting at node A to compute the Minimum Spanning Tree (MST) of the given graph. Write down the edges of the MST in the order in which Prim's algorithm adds them to the MST. **4**



Solution

Edges in MST in order

(A,C), (C,J), (C,D), (B,D), (B,H),(H,I), (H,G), (E,J), (E,F)

c How many character comparisons will the Boyer-Moore algorithm make in searching for each of the following patterns in the binary text of 1000 zeros? **6**

a. 00001 **b.** 10000

Solution:

For the pattern 00001, the shift tables will be filled as follows

the bad-symbol table

c	0	1
$t_1(c)$	1	5

the good-suffix table

k	the pattern	d_2
1	00001	5
2	00001	5
3	00001	5
4	00001	5

On each of its trials, the algorithm will make one unsuccessful comparison and then shift the pattern by d1 = max{t1(0) − 0, 1} = 1 position to the right without consulting the good-suffix table:

0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1
0 0 0 0 1
etc.
0 0 0 0 1

The total number of character comparisons will be C = 1 · 996 = 996.

d What data structure would you use to keep track of live nodes in a best-first branch-and-bound algorithm? **2**

Solution

The heap and min-heap for maximization and minimization problems, respectively.

5 a What is the key difference between a problem that can be solved efficiently by dynamic programming the and one that can be solved efficiently by divide-and conquer strategy? **2+4**

What is the consequence of this difference for dynamic programming solutions?

Solution:

Dynamic programming has overlapping sub-problems while Divide-and-conquer does not.

SRN

This means if we use the naive approach and recursively solve all sub-problems in a dynamic programming solution the same sub-problem will be solved repeatedly and often this leads to exponential time complexity. So, either we have to solve the problem iteratively bottom up or cache the solutions to sub-problems (memoization) and avoid resolving them

b Compare the time complexities of the Dijkstra algorithm and the Floyd's algorithm to determine the minimum weight paths between all pairs of vertices for sparse graphs and dense graphs, and justify which algorithm you would use for each of these two types of graphs 2+4

Solution:

The Floyd's algorithm runs once on a connected graph of V-vertices and E-edges to determine the shortest paths between all pairs of vertices, at a run-time complexity of **$\Theta(V^3)$.**

The Dijkstra's algorithm (of time complexity Θ(E*logV) on a V-vertex and E-edge graph) is designed to determine the shortest path from **one vertex** to all the other vertices in a connected graph. Hence, when this algorithm is to be used to determine the shortest paths between all pairs of vertices, the algorithm has to be run V-times, each time with a particular vertex as source. Hence, the overall time complexity of using the Dijkstra's algorithm for all-pairs-shortest-paths is Θ(V*E*logV).

For sparse connected graphs, the minimum number of edges is |E| = |V| - 1. Hence, E = Θ(V). For such graphs, O(V*E*logV) = $\Theta(V^2 \cdot \log V)$.
Since, logV < V, as V -> ∞, V^2*logV < V3. Hence, it would be better to use the Dijkstra's algorithm for sparse graphs.

For dense connected graphs, the maximum number of edges is
|E| = |V|*(|V|-1)/2. Hence, E = $\Theta(V^2)$. For such graphs, Θ(V*E*logV) = $\Theta(V^3 \cdot \log V)$ > Θ(V3). Hence, it would be better to use the Floyd's algorithm for dense graphs.

d Write an algorithm to solve knapsack problem using bottom up dynamic programming. Apply the algorithm to solve the following instance of knapsack problem. 4+4

objects	weights	Profits
1	2	3
2	3	4
3	4	5
4	5	6

Capacity of knapsack=5

SRN | | | | | | | | | | |

Solution

KnapSack(v, w, n, W)
{
for ($w = 0$ to W) $V[0, w] = 0$;
for ($i = 1$ to n)
for ($w = 0$ to W)
if ($w[i] \le w$)
$V[i, w] = \max\{V[i-1, w], v[i] + V[i-1, w - w[i]]\}$;
else
$V[i, w] = V[i-1, w]$;
return $V[n, W]$;
}

	0	1	2	3	4	5
0	0	0	0	0	0	0
✓ 1	0	0	3	3	3	3
✓ 2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

Solution vector
1 1 0 0