UNIVERSITY OF THE PUNJAB

B.S. in Computer Science / Fifth Semester – Fall 2023

Roll No.

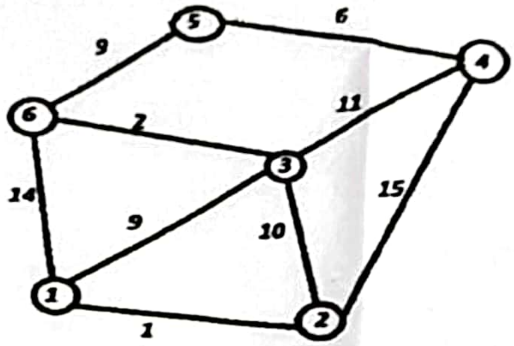
Subject: Design & Analysis of Algorithms **Paper: DC-321** **Time: 3 Hrs. Marks: 60**

THE ANSWERS MUST BE ATTEMPTED ON THE ANSWER SHEET PROVIDED

Q.1. Answer the following short questions: (6x5=30)

A) What is hashing? What does it mean by quadratic probing?

B) Show step-by-step execution of Kruskal's minimum spanning tree Algorithm for the graph given below where start vertex is vertex-1.



C) Fill-in the required information about following sorting algorithms.

Algorithm	In-place	Stable	Best Case	Worst Case
Insertion Sort				
Heap Sort				
Merge Sort				
Quick Sort				
Bubble Sort				

D) Discuss about small-omega (ω) asymptotic notation. What this notation represents?

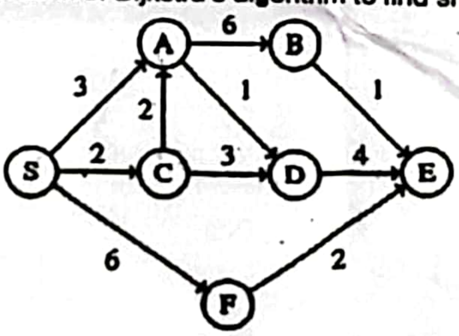
E) Sort the following data using quick-sort. Show detailed working to get full credit.

240, 6, 13, 45, 56, 8, 66, 44, 5, 23

F) What is meant by priority queue? Which data structures can be used for implementations of priority queues?

Q.2. Answer the following questions. (3x10=30)

A) Show step-by-step execution of Dijkstra's algorithm to find shortest paths starting from vertex S.



B) What are the three cases of master theorem? Give examples of solving recurrence relation for each case.

C) Write pseudo-code or C/C++ code to find strongly-connected-components in a given graph?

Q1.

A) What is hashing? What does it mean by quadratic probing?

Hashing :-

Hashing refers to the process of generating a fixed-size output from an variable-size input using the mathematical formulas known as hash functions.

This technique determines an index or location for the storage of an item in a data structure.

For example :-

List = [11, 12, 13, 14, 15]

Hash function = [x % 10]

	11 % 10	12 % 10	13 % 10	14 % 10	15 % 10
0	1	2	3	4	5
	11	12	13	14	15

Hash Table

Quadratic Probing :-

$h(k) = k \bmod 10$

keys = 42, 16, 91, 33, 18, 27, 36, 62

→ for 36 mod 10 = 6

index 6 is already filled.

In case of collision :-

$h'(k,i) = (h(k) + i^2) \bmod 10$

$6 + 1^2 \bmod 10 = 7$ already filled

$6 + 2^2 \bmod 10 = 0$

0	36	
1	91	91 % 10
2	42	42 % 10
3	33	33 % 10
4		
5		
6	16	16 % 10
7	27	27 % 10
8	18	18 % 10

→ for 62 mod 10 = 2

Index 2 is already filled

$2 + 1^2 \bmod 10 = 3$ already filled

$2 + 2^2 \bmod 10 = 6$ already filled

$2 + 3^2 \bmod 10 = 1$ already filled

$2 + 4^2 \bmod 10 = 8$ already filled

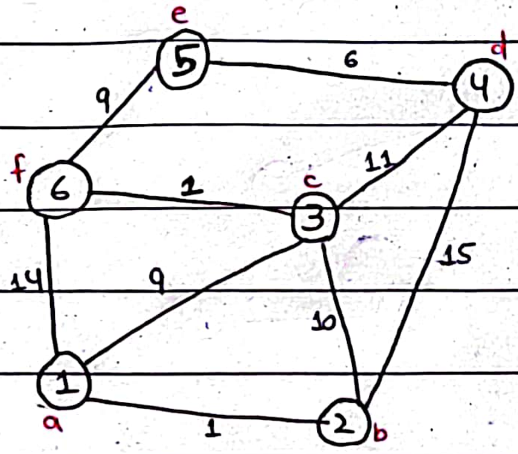
⋮ ⋮ ⋮ ⋮

⇒ Sometimes there is no guarantee to find an index.

Date:

M T W T F S

B) Show step-by-step execution of Kruskal's minimum spanning tree algorithm for the graph given below where start vertex is 1



list of sorted edges :

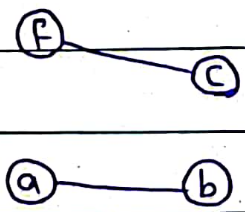
ab	cf	de	ac	ef	bc	cd	fa	bd
1	2	6	9	9	10	11	14	15

Number of edges in subtree = 6 - 1 = 5 edges

Tree edges Subtree

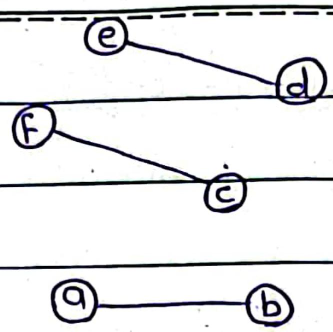
ab
1

cf
2

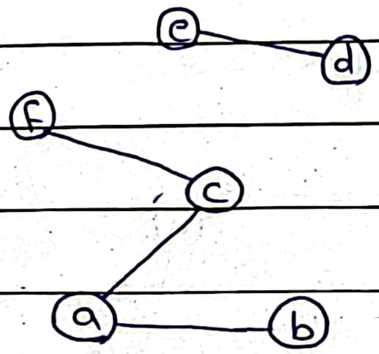


Date: M T W T F S

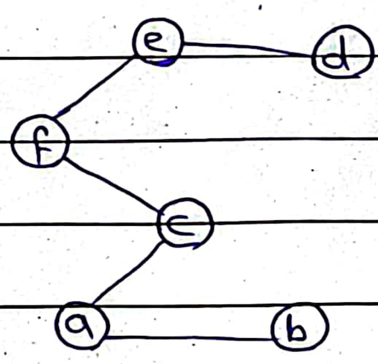
de
6



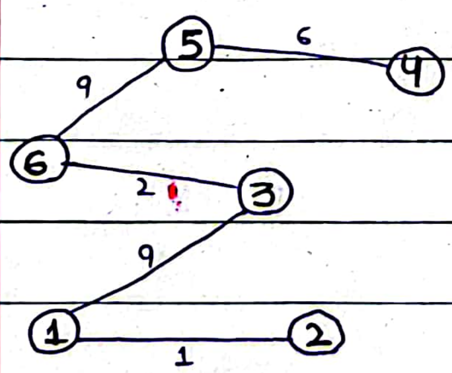
ac
9



ef
9



SUBTREE:-



Total edges = 5

Total vertics = 6

Date: ____________ M T W T F S

c) Fill in the required information:

Algorithm	In-place	Stable	Best Case	Worst Case
Insertion sort	Yes	Yes	O(n)	O(n²)
Heap sort	Yes	No	O(n log n)	O(n log n)
Merge sort	No	Yes	O(n log n)	O(n log n)
Quick sort	Yes	No	O(n log n)	O(n log n)
Bubble sort	Yes	Yes	O(n)	O(n²)

D) Discuss about small-omega (ω). What this notation represents?

Small omega (ω), is an asymptotic notation to denote the lower bound that is not asymptotically tight.

Date:

M T W T F S

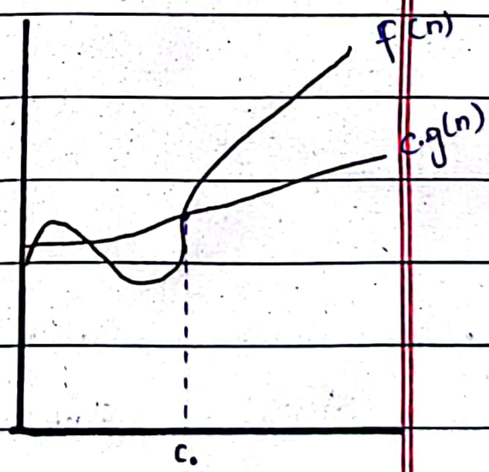
Little omega is a rough estimate of the order of growth not exact.

→ $f(n) \in \omega(g(n))$ iff $g(n) \in o(f(n))$

★ $g(n)$ is an asymptotic lower bound on $f(n)$

★ $f(n)$ and $g(n)$ grow at different rates.

$f(n) \geq c.g(n)$



Asymptotic notations are essential tools in algorithm analysis that allow us to compare the growth rate of functions in a precise manner.

Date:__________ M T W T F S

E) Sort the following data using quick-sort

240, 6, 13, 45, 56, 8, 66, 44, 5, 23

p

240 6 13 45 56 8 66 44 5 23

i i i i i i i i i $\boxed{j}$ $\boxed{i}$

swap 'j' and pivot

p

23 6 13 45 56 8 66 44 5 240

i i i $\boxed{i}$ $\boxed{j}$

p

23 6 13 5 56 8 66 44 45 240 swap i and j

i $\boxed{i}$ $\boxed{j}$ j j j

p

23 6 13 5 8 56 66 44 45 240 swap i and j

i j

$\boxed{j}$ $\boxed{i}$

swap 'j' with pivot

p p

8 6 13 5 23 56 66 44 45 240

i i $\boxed{i}$ $\boxed{j}$ i $\boxed{i}$ $\boxed{j}$

swap 'j' and 'i' swap 'i' and 'j'

p

8 6 5 13 23 56 45 44 66 240

p i j i i j

$\boxed{j}$ $\boxed{i}$ $\boxed{j}$ $\boxed{i}$

swap 'j' with pivot swap 'j' with pivot

5 6 8 13 23 44 45 56 66 240

Therefore, array in sorted form is:-

5 6 8 13 23 44 45 56 66 240

Date:

M T W T F S

F) What is meant by a priority queue? Which data structures can be used for implementation of pqueue?

Priority queue :-

A priority queue is a type of queue that arranges elements based on their priority level.

Elements with higher priority values are retrieved first. In a priority queue, each element has a priority value associated with it. When you add an element to the queue, it is inserted in a position based on its priority value.

Date:

M T W T F S

How to implement priority queue?

Priority queue can be implemented using the following data structures :

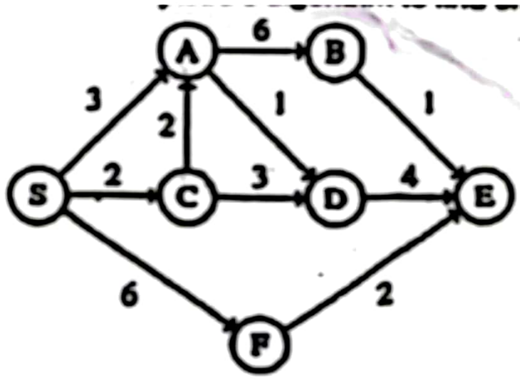
- Arrays
- Linked lists
- Heap
- Binary Search Tree

Date:

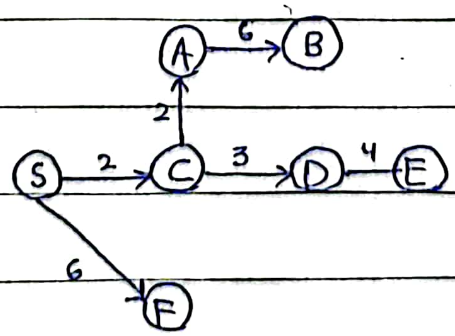
M T W T F S

Q2.

A) Show step-by-step execution of dijkstra's algorithm to find shortest path.



Step	Source	D(A) p(A)	D(B) p(B)	D(C) p(C)	D(D) p(D)	D(E) p(E)	D(F) p(F)
0	S	3,S	∞	2,S	∞	∞	6,S
1	SC	3,S	∞	–	5,C	∞	6,S
2	SCA	–	9,A	–	4,A	∞	6,S
3	SCAD	–	9,A	–	–	8,D	6,S
4	SCADE	–	9,A	–	–	–	6,S
5	SCADEF	–	9,A	–	–	–	–
6	SCADEFB						



Date:______ M T W T F S

8) What are the three cases of master theorem? Give examples of solving reccurence relation for each one.

Master Theorem :-

The master theorem is used to solve recurrence relations that arise in the analysis of divide-and-conquer algorithms.

It provides a systematic way of solving recurrence relations in the form :-

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

Cases of Master Theorem :-

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Date:

M T W T F S

$$T(n) = 2T\left(\frac{n}{4}\right) + 1$$

$a = 2$; $b = 4$; $d = 0$

$2 \quad 4^0$ ∴ Comparing

$2 > 1$

$$T(n) = \Theta(n^{\log_b a})$$
$$= \Theta(n^{\log_4 2})$$
$$= \Theta(n^{1/2})$$
$$= \Theta(\sqrt{n})$$

$$T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n}$$

$a = 2$; $b = 4$; $d = \frac{1}{2}$

$2 \quad 4^{1/2}$ ∴ comparing

$2 = 2$

$$T(n) = \Theta(n^d \log n)$$
$$= \Theta(n^{1/2} \log n)$$
$$= \Theta(\sqrt{n} \log n)$$

Date:

$$T(n) = 2T\left(\frac{n}{4}\right) + n$$

$a = 2$; $b = 4$; $d = 1$

$2 \quad 4^1$ ∴ comparing

$2 < 4$

$T(n) = \Theta(n^d)$

$= \Theta(n^1)$

$= \Theta(n)$

Date:____________ M T W T F S

c) Write pseudo code or c/c++ code to find strongly connected components in a given graph?

Pseudocode for depth-first search

Here is the pseudocode for the depth-first search algorithm:

```
DFS(graph, startNode):
    Initialize an empty stack S
    Initialize an empty list Visited

    Push startNode onto S

    while S is not empty:
        currentNode = top of S

        if currentNode is not in Visited:
            Mark currentNode as visited (add it to Visited)

        foundUnvisitedNode = false
        for each node N that is adjacent to currentNode:
            if N is not in Visited:
                Push N onto S
                foundUnvisitedNode = true
                break

        if foundUnvisitedNode is false:
            Pop currentNode from S
```

```cpp
#include<iostream>
#include<list>
#include<map>
using namespace std;

class GraphStructure {
    map<int, bool> visitedNodes;
    map<int, list<int>> adjacencyList;

public:
    void addEdge(int node1, int node2);
    void DFS(int startNode);
};

void GraphStructure::addEdge(int node1, int node2) {
    adjacencyList[node1].push_back(node2);
}

void GraphStructure::DFS(int startNode) {
    visitedNodes[startNode] = true;

    cout << startNode << " ";

    for(auto nextNode : adjacencyList[startNode]) {
        if (!visitedNodes[nextNode]) {
            DFS(nextNode);
        }
    }
}

int main() {
    GraphStructure graph;
    graph.addEdge(1, 2);
    graph.addEdge(1, 3);
    graph.addEdge(1, 4);
    graph.addEdge(4, 3);
    graph.addEdge(3, 5);

    cout << "Depth First Traversal of 1st graph: ";
    graph.DFS(1);

    GraphStructure graph1;
    graph1.addEdge(3, 7);
    graph1.addEdge(3, 4);
    graph1.addEdge(4, 8);

    cout << endl;
    cout << "Depth First Traversal of 2nd graph: ";
    graph1.DFS(3);

    GraphStructure graph2;
    graph2.addEdge(9, 5);
    graph2.addEdge(5, 4);
    graph2.addEdge(5, 3);

    cout << endl;
    cout << "Depth First Traversal of 3rd graph: ";
    graph2.DFS(9);

    return 0;
}
```