1) Describe a heap and its purpose in Analysis of Algorithm.

A heap is a specialized binary tree based data structures where each node satisfies the heap property.

The Main purpose of a heap in the analysis of algorithms is to efficiently maintain and retrieve the maximum element

2) Explain how a binary max-heap and a binary min-heap vary from one-another

A binary max-heap ensures that the value of each node is greater that or equal to the values of its children, with the maximum val out the root, conversely a binary min-heap guarantees that the value of each node is less than or equal to the values of its children, with the minimum value at the root

3) Compare and contrast a heap with an ordinary binary tree in terms of structures and operations.

A heap is a specialized binary tree where each node satisfies the heap property, ensuring efficient retrieval of the maximum element

In ordinary binary tree doesn't enforce any specific ordering among the nodes and offers more flexibility in terms of structure.

4) Explain the concept of the activity selection problem.

The activity selection problem involves selecting a maximum number of activities that can be performed by a single person or resource, given a set of activities. that, each with a start time and finish time..

5) Describe the way in which the problem of activity selection might be represented as an optimization problem.

The problem of activity selection can be represented as an optimization problem. by an objective function and constraints

Objective function : maximize the number of activities Selected.

Constraints: Activities Selected must not overlap in time.

Compare and contrast the greedy algorithm and dynamic programming approaches to solving the Activity selection.

Greedy Algorithm:-

Greedy algorithm make locally optimal choices at each step with the hope of finding a globally optimal solution.

=> The greedy algorithm has a time Complexity of O(nlogn), where n is the number of activities, making it efficient for most practical purposes.

Discuss the importance of sorting activities based on the finish time in the activity selection problem.

Sorting activities based on their finish times is crucial in the activity selection problem for several reasons.

=) Ensuring Compatibility

=> optimizing Solution Quality

=) Efficiency

=) Simplicity

practical compression

Huffman coding is widely used in practical areas, including image and text compression, due to its effectiveness in achieving lossless compression.

Text compression:-

File compression

Data compression

Image compression:

=) Loseless compression

=) Lossy compression.

) Talk about how important it is to solve the TSP optimally for real, world applications

Solving the TSP optimally is crucial for real-world application due to its direct impact on resource efficiency and cost reduction. By finding the shortest route to visit multiple locations.

Give an explanation of permutations and Combination as well as their function in algorithm analysis.

Permutations represent arrangements of elements where the order matter. While combination represent selection where the order doesn't matter.

=> They help analyze the time complexity of algorithms by determining the number of operative required for different input sizes.

16 Marks:

) Implement a priority queue using a binary heap. A priority queue is a data structure that store elements along with their associated priorities, where elements with higher priorities are dequeued before elements with lower priorities. Your task is create a program utilizes a binary heap to achieve efficient insertion, deletion of the maximum element, and retrieval of the maximum element operations.

```
#Include <iostream>
using namespace std;
int H[50];
int size = -1;
  int parent (int i) { return (i-1)/2; }
```

```
void shiftup (int i) {
while (i>0 && H[parent(i)] < H[i]) {
    Swap (H[parent(i)], H[i]);
    i= parent(i); }
void shiftdown. (int i) {
    int    maxindex = i;
    int l = leftchild(i);
if (i < size && H[l] > H[maxindex])
        maxIndex = l;
    if (i != maxIndex) {
    Swap (H[i], H[maxIndex])
    shiftDown (maxIndex); }
void  insert (int p) {
    size = size + 1;
    H[size] = p;
    shiftup = (size);
}
int  getmax() { return H[0]; }
int main() {
    insert (45);
    insert (80);
}
```

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std (vector<int>& nums, int k) {
    if (k<=0 || k > nums.size()) {
        return -1; }

    priority_queue<int> maxHeap;
    for (int i=0; i<k; i++) {
        maxHeap.push(nums[i]); }
    for (int i = k; i<nums.size(); i++) {
        if (nums[i]< maxHeap.pop()) {
            maxHeap.pop();
            maxHeap.push(nums[i]); }
    } return maxHeap.pop();
int main() {
    Stand (time(NULL));
    Vector<int> num1={3,1,4,2,5};
    int k1=2;
    Cout << "Array :";
    for (int num : nums1) {
        Cout<< num << " "; }.
    insert(14);
    insert(31);
    Insert(11);      Insert(13); Insert(7);
```

```
cout<< "Max priority item :
int k=0;
while (k<=size) {
    cout<< H[k] <<" ";  k++; }

remove(3);
return 0;
}
```

create a program to efficiently find the kth smallest element in an array of integers using a binary heap. Compose the methodology behind determing the kth smallest element with a binary heap. and discuss its efficiency. Your task is implementation a program to Conveing varias scenarios such as different array sizes, random element values. and varying values of k. Ensure you program handles edge cases effectively and accurately identifies the kth smallest element

```
cout<< "kth smallest element : " << kthSmallest
                    elements (nums, k1).
                    << endl;

int k2=3;
for (int num : num2) {
    cout<<num<<" "; }
```

cout << "kth smallest element : " <<

kthsmallest(nums, k) << endl;

return 0; }

Imagine you're building a project management system for your a software development company. Your system helps teams manage their tasks and dependencies efficient. Each task represents a specific piece of work, and some tasks depend on others be completed before they can start.

i) Task management:

Each task should have a unique identifier, title, description, priority, estimated duration and status.

Tasks can be organized into projects or milestones for better organisation.

Task dependencies should be supported to indicate which tasks must be completed before others can start.

ii) Dependency management:

Tasks can dependencies on other tasks, forming a directed acyclic graph of dependenci

Users should be able to view task dependencies and track programs based on dependencies.

3) Team collaboration:

i) Users should be able to collaborate by commenting on tasks, sharing files and mentioning team members.

ii) Task assignments and workload distribution should be transparent and manageable.

4) Reporting and analytics:

The System should generate reports on project progress, task complete on rates and team performance.

Security and access control:

Access control mechanisms should be implemented to restrict access to sensitive information and features based on user roles and permission.

Define a concept of N-queen problem with suitable examples.

The N-Queens problem is classic combinational problem is chessboard - based puzzles. The objectives is to place N-queens on an NXN chessboard in such a way that no two queens threaten each other.

N=4

- Q - -
- - - Q
Q - - -
- - Q -

```
# include <iostream>
# include <vector>
using namespace std;
bool issafe ( vector <vector <int>> & board, int raw,
                    int col , int N) {
    for (int i=0; i< row ; i++) {
        if (board [i][col] == 1) {
            return false ; }
    }
```

```
for (int i = row; j = col ...

    if (board[i][j] == 1){
        return false; }

for (int i = row ; j = col ; i >= 0 && j < N , i--; j++

    if (board[i][j] == 1) {
        return false; }
}
bool Solve (Vector<Vector<int>> & board, int row

    { if (row == N) {
            return true; }
}
for (int col = 0; col < N; col++) {
    if (isSafe (board, row, col, N)) {
        board [row] [col] = 1;
        if (SolveN (board, row+1, N), {
            return true; }
        board [row][col] = 0; }
    } return false;
}
int main(){
    int N=4;
    if (SolveN (board, 0, N)). {
```

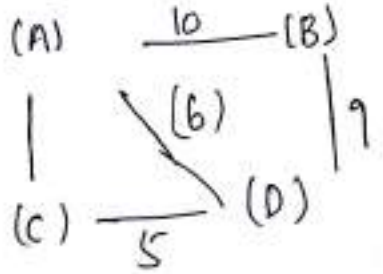
```
for (int i=0; i<N; i++) {
    for (int j=0; j<N; j++) {
        cout << board[i][j] << " ";
    }
    cout << endl;
}
return 0;
}
```

Describe dijkstra's algorithm with example.

It is a graph search algorithm used to find the shortest path from a source vertex to all other vertices in a weighted graph with non-negative edge weights. It works by iteratively exploring the vertices in the graph, updating their distances from the source vertex as it progresses.

1) Initialization:

Start with a graph represented as an adjacency list or matrix and initial it.



Set the distance of node A to 0 and
other nodes to infinity.
Set the Tentative distance of nodes B, C and D
to and infinity.

Iteration:

=) Visit node C.

=) Update the distance to node D to be 11

=) Visit node B.

=) Update the distance to node D to be 10

=) Visit node D.

=) No further updates needed.

Termination:

All nodes name been visited

Backtracking:

The shortest path from A to D is A.

=> C → D with total distance of 11.

A to B : 10

A to C : 5

A to D : 11. //