Experiment-1

Revisit and Refresh the Study of Python (SEARCHING TECHNIQES)

TASK-1

With Understanding, Identify The Scope, Advantages And Disadvantages Of The Given Searching Algorithm.

Searching Technique	Scope	Constraints	Application
1. Linear Search	Sequentially checks each element in list. Adv: simple and universal Disadv: slow on large DBs	No sorting required, works on any list	Small datasets, unsorted arrays.
2. Binary Search	Divide and conquer search on sorted arrays Adv: very fast Disadv: needs sorted array	List must be sorted first.	Searching dictionaries, large indexes.
3. Depth-First Search (DFS)	Traverse graph deeply before backtracking. Adv: memory efficient. Disadv: can miss sometimes and give longer solutions.	Require stack data structure.	Cycle detection, path finding.
4. Breadth-First Search (BFS)	Traverse graph level by level Adv: gives shortest path Disadv: high memory for wider graphs.	Needs queue data structure.	Shortest path in unweighted graphs.
5. Hash-Based Search	Direct lookups using hashkeys Adv: O(1) Disadv: collisions	Required good hash fns.	Hash tables, dictionaries.

6. Jump Search	Searching sorted list by jumping fixed steps and linear scan. Adv: faster than linear search Disadv: slow than binary search	Required sorted data	Faster than linear search on sorted arrays
7. Interpolation Search	Uses value distribution to predict search position. Adv: faster than binary in uniform data Disadv: poor in skewed data	Uniformly distributed sorted data	Numeric datasets with predictable distribution
8. Exponential Search	Find range exponentially then binary search inside it Adv: excellent when target near beginning Disadv: needs stored data	Data must be stored	Unbounded or very large sorted arrays
9. Fibonacci Search	Search sorted array using Fibonacci partitions Adv: no division Disadv: slower than binary	Sorted list needed	Systems where division is costly
10. Ternary Search	Divide into 3 parts Adv: simple for unimodal Disadv: slower than binary	Only for unimodal functions.	Optimization, min or max finding
11. Sublist Search (KMP Algorithm)	Find pattern inside text using prefix table Adv: O(n+m) Disadv: preprocessing overhead	Pattern preprocessing required	String matching, text search, compilers
12. A* Search Algorithm	Heuristic best-first search for optimal paths. Adv: finds optimal path efficiently	Needs admissible hueristics	Maps, games, robotics pathfinding

	Disadv: high memory usage		
13. Beam Search	Heuristic search that explores multiple best partial solutions at each step but keeps only the top *k* (beam width). Adv: faster and less memory Disadv: can prune away optimal solution	Quality depends heavily on beam width; may miss the globally optimal path.	NLP decoding, speech recognition.
14. Grover's Search Algorithm	Quantum alg for searching unsorted databases Adv: O(sqrt n) Disadv: not practical on classical machines	Requires quantum computer Mostly theoretical.	Cryptography, quantum optimization

TASK-2

With Understanding Write The Algorithm/ Steps To Implement The Searching Algorithms.

Searching Technique	Algorithm/ Program	Input/Output
1. Linear Search	Algorithm LinearSearch(A, x) for i from 0 to n−1 do if A[i] == x then return i return -1	Input: Array A, target x Output: Index of x or -1
2. Binary Search	Algorithm BinarySearch(A, x) low ← 0 ; high ← n−1 while low ≤ high do mid ← (low + high)/2 if A[mid] == x then return mid else if x < A[mid] then high ← mid−1 else low ← mid+1 return -1	Input: Sorted array A, target x Output: Index of x or -1
3. Depth-First Search (DFS)	Algorithm DFS(G, s) mark s as visited print s for each neighbor v of s do if v not visited then DFS(G, v)	Input: Graph G, start node s Output: DFS traversal order

4. Breadth-First Search (BFS)	Algorithm BFS(G, s)
	create queue Q
	mark s as visited ; enqueue s
	while Q not empty do
	u ← dequeue(Q)
	print u
	for each neighbor v of u do
	if v not visited then mark visited ; enqueue v
		Input: Graph G, start node s
		Output: BFS traversal order

5. Hash-Based search	Algorithm HashSearch(H, k) index ← hash(k) if H[index] contains k then return value else search collisions (chain/probe) return "not found"	Input: Hash table H, key k Output: Value or "not found"
6. Jump Search*	Algorithm JumpSearch(A, x) step ← √n ; prev ← 0 while A[min(step,n)-1] < x do	Input: Sorted array A, target x Output: Index or -1

	prev ← step ; step ← step + √n if prev ≥ n then return -1 for i from prev to step do if A[i] == x then return i return -1	
7. Interpolation Search*	Algorithm InterpolationSearch(A, x) low ← 0 ; high ← n−1 while A[low] ≤ x ≤ A[high] do pos ← low + (x-A[low])*(high-low)/(A[high]-A[low]) if A[pos] == x then return pos else if x < A[pos] then high ← pos−1 else low ← pos+1 return -1	Input: Sorted uniform array A, target x Output: Index or -1
8. Exponential Search*	Algorithm ExponentialSearch(A, x) if A[0] == x then return 0 i ← 1 while i < n AND A[i] ≤ x do i ← i * 2 return BinarySearch(A, x, i/2, min(i,n-1))	Input: Sorted array A, target x Output: Index or -1

9. Fibonacci Search*	Algorithm FibonacciSearch(A, x)
	generate Fibonacci numbers until F[k] ≥ n
	offset ← -1
	while F[k] > 1 do
	i ← min(offset + F[k−2], n−1)
	if A[i] < x then
	k ← k−1 ; offset ← i
	else if A[i] > x then
	k ← k−2
	else return i
	return -1
		Input: Sorted array A, target x
		Output: Index or -1

10. Ternary Search	Algorithm TernarySearch(f, l, r) while r − l > ε do m1 ← l + (r−l)/3 m2 ← r − (r−l)/3 if f(m1) < f(m2) then r ← m2 else l ← m1 return (l+r)/2	Input: Function f(x), range [l, r] Output: Point of min/max
11. Sublist Search (KMP Algorithm)*	Algorithm KMP(T, P) compute LPS array for P i ← 0 ; j ← 0 while i < length(T) do if P[j] == T[i] then i++, j++	Input: Pattern P, text T Output: Index of first match or -1

	if j == length(P) then return i−j else if mismatch and j > 0 then j ← LPS[j−1] else i++ return -1	

12. A* Search Algorithm*	Algorithm A*(G, s, g)
	open ← priority queue containing s
	gScore[s] ← 0
	while open not empty do
	u ← node with lowest f = gScore + h
	if u == g then return path
	for each neighbor v of u do
	temp ← gScore[u] + cost(u,v)
	if temp < gScore[v] then
	gScore[v] ← temp
	parent[v] ← u
	add v to open
		Input: Graph with costs, start s, goal g, heuristic h()
		Output: Optimal path

13. Beam Search	Algorithm BeamSearch(start, k) beam ← {start} while goal not found AND beam not empty do candidates ← generate all successors of nodes in beam beam ← select k best candidates by heuristic return best node in beam	Input: Initial state, successor function, beam width k Output: Best solution found
14. Grover's Search Algorithm*	Algorithm GroverSearch() initialize equal superposition of all states repeat O(√N) times: apply oracle to flip phase of target state apply diffusion operator (inversion about mean) measure state (gives target index with high probability)	Input: Unsorted database of N items, target condition Output: Index where condition is true

Name: P Kartikeya Raj

Reg_no: 2023001492

Date: 3rd December 2025