**Quick Sort**

**Task 1: Quick Sort with First Element as Pivot**

1. **Unsorted array**: 10,16,8,12,15,6,3,9,5
2. **Pivot**: The first element, 10.

Let's write the Quick Sort algorithm and show the array after each partition and recursive call.

python

Copy code

def quick\_sort(arr):

if len(arr) <= 1:

return arr

pivot = arr[0]

less = [x for x in arr[1:] if x <= pivot]

greater = [x for x in arr[1:] if x > pivot]

return quick\_sort(less) + [pivot] + quick\_sort(greater)

# Initial array

arr = [10, 16, 8, 12, 15, 6, 3, 9, 5]

sorted\_arr = quick\_sort(arr)

print("Sorted array:", sorted\_arr)

**Task 2: Quick Sort with Middle Element as Pivot**

1. **Unsorted array**: 19,72,35,46,58,91,22,31
2. **Pivot**: The middle element.

python

Copy code

def quick\_sort\_with\_middle\_pivot(arr):

if len(arr) <= 1:

return arr

pivot = arr[len(arr) // 2]

less = [x for x in arr if x < pivot]

equal = [x for x in arr if x == pivot]

greater = [x for x in arr if x > pivot]

return quick\_sort\_with\_middle\_pivot(less) + equal + quick\_sort\_with\_middle\_pivot(greater)

# Initial array

arr = [19, 72, 35, 46, 58, 91, 22, 31]

sorted\_arr = quick\_sort\_with\_middle\_pivot(arr)

print("Sorted array:", sorted\_arr)

**Binary Search**

**Task 1: Binary Search with Comparisons Count**

1. **Array**: 5,10,15,20,25,30,35,40,45
2. **Search key**: 20

python

Copy code

def binary\_search(arr, x):

left, right = 0, len(arr) - 1

comparisons = 0

while left <= right:

mid = (left + right) // 2

comparisons += 1

if arr[mid] == x:

return mid, comparisons

elif arr[mid] < x:

left = mid + 1

else:

right = mid - 1

return -1, comparisons

# Initial array

arr = [5, 10, 15, 20, 25, 30, 35, 40, 45]

index, comparisons = binary\_search(arr, 20)

print("Index of 20:", index)

print("Comparisons made:", comparisons)

**Task 2: Binary Search with Steps**

1. **Array**: 3,9,14,19,25,31,42,47,53
2. **Search key**: 31

python

Copy code

def binary\_search\_steps(arr, x):

left, right = 0, len(arr) - 1

steps = []

while left <= right:

mid = (left + right) // 2

steps.append((left, mid, right))

if arr[mid] == x:

return mid, steps

elif arr[mid] < x:

left = mid + 1

else:

right = mid - 1

return -1, steps

# Initial array

arr = [3, 9, 14, 19, 25, 31, 42, 47, 53]

index, steps = binary\_search\_steps(arr, 31)

print("Index of 31:", index)

print("Steps taken:", steps)

**Optimal Binary Search Trees (OBST)**

**Task 1: OBST with Given Keys and Frequencies**

python

Copy code

import numpy as np

def optimal\_bst(keys, freq, n):

cost = np.zeros((n, n))

root = np.zeros((n, n))

for i in range(n):

cost[i][i] = freq[i]

for L in range(2, n+1):

for i in range(n - L + 1):

j = i + L - 1

cost[i][j] = float('inf')

for r in range(i, j + 1):

c = (0 if r == i else cost[i][r-1]) + (0 if r == j else cost[r+1][j]) + sum(freq[i:j+1])

if c < cost[i][j]:

cost[i][j] = c

root[i][j] = r + 1

return cost, root

# Initial keys and frequencies

keys = ['A', 'B', 'C', 'D']

freq = [0.1, 0.2, 0.4, 0.3]

n = len(keys)

cost, root = optimal\_bst(keys, freq, n)

print("Cost Table:")

print(cost)

print("Root Table:")

print(root)