# DESIGN AND ANALYSIS OF ALGORITHMS LAB
# TOPIC 3: DIVIDE AND CONQUER (Q1–Q16)

## Q1. Binary Search

**Aim:** To search an element in a sorted array using divide and conquer.

**Procedure:** Repeatedly divide array and compare middle element.

**Code:**

```
def binary_search(arr, low, high, key):
    if low <= high:
        mid = (low + high) // 2
        if arr[mid] == key:
            return mid
        elif key < arr[mid]:
            return binary_search(arr, low, mid-1, key)
        else:
            return binary_search(arr, mid+1, high, key)
    return -1

print(binary_search([2,4,6,8,10], 0, 4, 8))
```

**Output:** Index of element printed.

**Result:** Binary search executed successfully.

## Q2. Merge Sort

**Aim:** To sort an array using merge sort.

**Procedure:** Divide array, sort subarrays and merge them.

**Code:**

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        L = arr[:mid]
        R = arr[mid:]
        merge_sort(L)
        merge_sort(R)
        i=j=k=0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k]=L[i]; i+=1
            else:
                arr[k]=R[j]; j+=1
            k+=1
        while i < len(L):
            arr[k]=L[i]; i+=1; k+=1
        while j < len(R):
            arr[k]=R[j]; j+=1; k+=1

arr=[38,27,43,3,9]
merge_sort(arr)
print(arr)
```

**Output:** Sorted array displayed.

**Result:** Merge sort completed successfully.

## Q3. Quick Sort

**Aim:** To sort array using quick sort.

**Procedure:** Choose pivot and partition array.

**Code:**

```
def quick_sort(arr):
    if len(arr)<=1:
        return arr
    pivot=arr[0]
    left=[x for x in arr[1:] if x<=pivot]
    right=[x for x in arr[1:] if x>pivot]
    return quick_sort(left)+[pivot]+quick_sort(right)

print(quick_sort([10,7,8,9,1,5]))
```

**Output:** Sorted array printed.

**Result:** Quick sort works correctly.

## Q4. Find Maximum and Minimum

**Aim:** To find max and min using divide and conquer.

**Procedure:** Divide array and compare values.

**Code:**

```
def max_min(arr,l,r):
    if l==r:
        return arr[l],arr[l]
    if r==l+1:
        return max(arr[l],arr[r]),min(arr[l],arr[r])
    m=(l+r)//2
    max1,min1=max_min(arr,l,m)
    max2,min2=max_min(arr,m+1,r)
    return max(max1,max2),min(min1,min2)

print(max_min([3,5,1,8,2],0,4))
```

**Output:** Max and Min printed.

**Result:** Maximum and minimum found.

## Q5. Strassen’s Matrix Multiplication

**Aim:** To multiply matrices using Strassen’s method.

**Procedure:** Divide matrices and compute 7 products.

**Code:**

```
import numpy as np
def strassen(A,B):
    if len(A)==1:
        return A*B
    mid=len(A)//2
    A11,A12,A21,A22=A[:mid,:mid],A[:mid,mid:],A[mid:,:mid],A[mid:,mid:]
    B11,B12,B21,B22=B[:mid,:mid],B[:mid,mid:],B[mid:,:mid],B[mid:,mid:]
    M1=strassen(A11+A22,B11+B22)
    M2=strassen(A21+A22,B11)
    M3=strassen(A11,B12-B22)
    M4=strassen(A22,B21-B11)
    M5=strassen(A11+A12,B22)
    M6=strassen(A21-A11,B11+B12)
    M7=strassen(A12-A22,B21+B22)
    C11=M1+M4-M5+M7
    C12=M3+M5
    C21=M2+M4
    C22=M1-M2+M3+M6
    return np.vstack((np.hstack((C11,C12)),np.hstack((C21,C22))))

A=np.array([[1,2],[3,4]])
B=np.array([[5,6],[7,8]])
print(strassen(A,B))
```

**Output:** Product matrix printed.

**Result:** Matrix multiplication successful.

## Q6. Power using Divide and Conquer

**Aim:** To compute x^n efficiently.

**Procedure:** Reduce power by half recursively.

**Code:**

```
def power(x,n):
    if n==0:
        return 1
    temp=power(x,n//2)
    if n%2==0:
        return temp*temp
    else:
        return x*temp*temp

print(power(2,10))
```

**Output:** Power value printed.

**Result:** Power computed correctly.

## Q7. Majority Element

**Aim:** To find majority element in array.

**Procedure:** Divide array and compare counts.

**Code:**

```
def majority(arr):
    if len(arr)==1:
        return arr[0]
    mid=len(arr)//2
    left=majority(arr[:mid])
    right=majority(arr[mid:])
    if left==right:
        return left
    return left if arr.count(left)>arr.count(right) else right

print(majority([2,2,1,1,2,2,2]))
```

**Output:** Majority element printed.

**Result:** Majority element found.

## Q8. Maximum Subarray Sum

**Aim:** To find maximum subarray sum.

**Procedure:** Divide array and find crossing sum.

**Code:**

```
def max_sub(arr,l,h):
    if l==h:
        return arr[l]
    m=(l+h)//2
    return max(max_sub(arr,l,m),max_sub(arr,m+1,h),cross(arr,l,m,h))

def cross(arr,l,m,h):
    s=0; ls=-1e9
    for i in range(m,l-1,-1):
        s+=arr[i]; ls=max(ls,s)
    s=0; rs=-1e9
    for i in range(m+1,h+1):
        s+=arr[i]; rs=max(rs,s)
    return ls+rs

arr=[2,-1,3,-4,5]
print(max_sub(arr,0,len(arr)-1))
```

**Output:** Maximum sum printed.

**Result:** Maximum subarray found.

## Q9. Count Inversions

**Aim:** To count number of inversions.

**Procedure:** Use merge sort technique.

**Code:**

```
def count_inv(arr):
    return merge(arr)[1]

def merge(arr):
    if len(arr)<=1:
        return arr,0
    mid=len(arr)//2
    L,c1=merge(arr[:mid])
    R,c2=merge(arr[mid:])
    i=j=0; inv=c1+c2; res=[]
    while i<len(L) and j<len(R):
        if L[i]<=R[j]:
            res.append(L[i]); i+=1
        else:
            res.append(R[j]); inv+=len(L)-i; j+=1
    res+=L[i:]+R[j:]
    return res,inv

print(count_inv([1,20,6,4,5]))
```

**Output:** Inversion count printed.

**Result:** Inversions counted correctly.

## Q10. Missing Number

**Aim:** To find missing number in sorted array.

**Procedure:** Use binary search approach.

**Code:**

```
def missing(arr,l,h):
    if l>h:
        return l
    m=(l+h)//2
    if arr[m]==m:
        return missing(arr,m+1,h)
    return missing(arr,l,m-1)

print(missing([0,1,2,3,5,6],0,5))
```

**Output:** Missing number printed.

**Result:** Missing number found.

## Q11. Fibonacci

**Aim:** To compute Fibonacci number.

**Procedure:** Use recursive divide and conquer.

**Code:**

```
def fib(n):
    if n<=1:
        return n
    return fib(n-1)+fib(n-2)

print(fib(10))
```

**Output:** Fibonacci value printed.

**Result:** Fibonacci calculated.

## Q12. Find Maximum Element

**Aim:** To find maximum element.

**Procedure:** Divide array and compare values.

**Code:**

```
def find_max(arr,l,r):
    if l==r:
        return arr[l]
    m=(l+r)//2
    return max(find_max(arr,l,m),find_max(arr,m+1,r))

print(find_max([4,2,7,1,9],0,4))
```

**Output:** Maximum value printed.

**Result:** Maximum element found.

## Q13. Find Minimum Element

**Aim:** To find minimum element.

**Procedure:** Divide array and compare values.

**Code:**

```
def find_min(arr,l,r):
    if l==r:
        return arr[l]
    m=(l+r)//2
    return min(find_min(arr,l,m),find_min(arr,m+1,r))

print(find_min([4,2,7,1,9],0,4))
```

**Output:** Minimum value printed.

**Result:** Minimum element found.

## Q14. Binary Search Iterative

**Aim:** To search element iteratively.

**Procedure:** Use loop-based binary search.

**Code:**

```
def bsearch(arr,key):
    l,r=0,len(arr)-1
    while l<=r:
        m=(l+r)//2
        if arr[m]==key:
            return m
        elif key<arr[m]:
            r=m-1
        else:
            l=m+1
    return -1

print(bsearch([2,4,6,8,10],8))
```

**Output:** Index printed.

**Result:** Iterative binary search successful.

## Q15. Median of Two Sorted Arrays

**Aim:** To find median value.

**Procedure:** Merge arrays and compute median.

**Code:**

```
def median(a,b):
    c=sorted(a+b)
    n=len(c)
    return (c[n//2]+c[n//2-1])/2 if n%2==0 else c[n//2]

print(median([1,3],[2,4]))
```

**Output:** Median value printed.

**Result:** Median found correctly.

## Q16. Closest Pair of Points

**Aim:** To find closest pair distance.

**Procedure:** Sort points and compare neighbors.

**Code:**

```
import math
def closest(points):
    points.sort()
    return min(math.dist(points[i],points[i+1]) for i in range(len(points)-1))

print(closest([(1,2),(4,5),(3,1),(6,7)]))
```

**Output:** Closest distance printed.

**Result:** Closest pair identified.