Lab Assignment 1: Algorithm Foundations

```
import time, random
import matplotlib.pyplot as plt
```

Task 1: Fibonacci (Recursive vs Dynamic Programming)

Recursive Fibonacci calls itself again and again → very slow (O(2^n)).

DP Fibonacci stores results → fast (O(n)).

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

def fib_dp(n):
    f=[0,1]
    for i in range(2,n+1):
        f.append(f[i-1]+f[i-2])
    return f[n]

print("Fib Rec(10):", fib_rec(10))
print("Fib DP(10):", fib_dp(10))
```

```
Fib Rec(10): 55
Fib DP(10): 55
```

Task 2: Sorting Algorithms

Bubble Sort: Swap repeatedly → $O(n^2)$

Insertion Sort: Insert into correct position → $O(n^2)$

Selection Sort: Pick smallest each time → $O(n^2)$

Merge Sort: Divide and merge → O(n log n)

Quick Sort: Divide by pivot → O(n log n) average, $O(n^2)$ worst

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n-1):
        for j in range(n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

arr = [5, 3, 1, 4, 2]
print("Original:", arr)
print("Sorted:", bubble_sort(arr))
```

```
Original: [5, 3, 1, 4, 2]
Sorted: [1, 2, 3, 4, 5]
```

```
def insertion_sort(a):
    for i in range(1, len(a)):
        key = a[i]; j = i-1
        while j >= 0 and a[j] > key:
            a[j+1] = a[j]; j -= 1
        a[j+1] = key
    return a

a = [5,3,1,4,2]
print("sorted : ", insertion_sort(a))
```

```
sorted :  [1, 2, 3, 4, 5]
```

```
def selection_sort(a):
    for i in range(len(a)):
        min_idx = i
        for j in range(i+1, len(a)):
            if a[j] < a[min_idx]:
                min_idx = j
        a[i], a[min_idx] = a[min_idx], a[i]
    return a

a = [5,3,1,4,2]
print("sorted : ", selection_sort(a))
```

```
sorted :  [1, 2, 3, 4, 5]
```

```
def merge_sort(a):
    if len(a) <= 1:
        return a
    mid = len(a)//2
    left = merge_sort(a[:mid])
    right = merge_sort(a[mid:])
    return merge(left, right)

a = [5,3,1,4,2]
print("sorted : ", merge_sort(a))
```

```
sorted :  [1, 2, 3, 4, 5]
```

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

```
a = [5,3,1,4,2]
print("sorted : ", merge_sort(a))
```

```
sorted :  [1, 2, 3, 4, 5]
```

Task 3: Binary Search

Works on sorted arrays

Cuts search space in half each time

O(log n) time

```
def binary_search(a, x):
    l, h = 0, len(a)-1
    while l <= h:
        mid = (l+h)//2
        if a[mid] == x:
            return mid
        elif a[mid] < x:
            l = mid+1
        else:
            h = mid-1
    return -1

print("Binary Search on [1,2,3,4,5], 3 →", binary_search([1,2,3,4,5],
3))
```

```
Binary Search on [1,2,3,4,5], 3 → 2
```

Task 4: Experimental Profiling & Graph

Bubble Sort grows very slow for large inputs.

Merge Sort grows much faster.

Graph shows Bubble rising steeply, Merge rising slowly.

```
import time, random, matplotlib.pyplot as plt

sizes = [100, 500, 1000, 2000]
times_bubble = []
times_merge = []

for n in sizes:
    arr = [random.randint(0,10000) for _ in range(n)]
    # Bubble
    a = arr[:]; start = time.time(); bubble_sort(a);
times_bubble.append(time.time()-start)
    # Merge
    a = arr[:]; start = time.time(); merge_sort(a);
```

```
times_merge.append(time.time()-start)

plt.plot(sizes, times_bubble, label="Bubble Sort")
plt.plot(sizes, times_merge, label="Merge Sort")
plt.xlabel("Input Size"); plt.ylabel("Time (s)")
plt.title("Sorting Performance")
plt.legend()
plt.show()
```