DAA_Lab_Experiments

October 16, 2025

0.0.1 Experiment 1A: Bubble Sort

[1]:
```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

print(bubble_sort([5, 2, 9, 1, 5, 6]))
```

```
[1, 2, 5, 5, 6, 9]
```

0.0.2 Experiment 1B: Selection Sort

[2]:
```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr

print(selection_sort([64, 25, 12, 22, 11]))
```

```
[11, 12, 22, 25, 64]
```

0.0.3 Experiment 1C: Insertion Sort

[3]:
```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
```

```
        arr[j + 1] = key
    return arr

print(insertion_sort([12, 11, 13, 5, 6]))
```

```
[5, 6, 11, 12, 13]
```

0.0.4 Experiment 2A: Linear Search

[4]:
```
def linear_search(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i
    return -1

print(linear_search([2, 3, 4, 10, 40], 10))
```

```
3
```

0.0.5 Experiment 2B: Binary Search

[5]:
```
def binary_search(arr, x):
    low, high = 0, len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            low = mid + 1
        else:
            high = mid - 1
    return -1

print(binary_search([2, 3, 4, 10, 40], 10))
```

```
3
```

0.0.6 Experiment 3A: Merge Sort

[6]:
```
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
    return arr

print(merge_sort([12, 11, 13, 5, 6, 7]))
```

```
[5, 6, 7, 11, 12, 13]
```

0.0.7 Experiment 3B: Quick Sort

[7]:
```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr)//2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)

print(quick_sort([3,6,8,10,1,2,1]))
```

```
[1, 1, 2, 3, 6, 8, 10]
```

0.0.8 Experiment 3C: Heap Sort

[8]:
```
def heapify(arr, n, i):
    largest = i
    l = 2*i + 1
    r = 2*i + 2

    if l < n and arr[l] > arr[largest]:
        largest = l
```

```
    if r < n and arr[r] > arr[largest]:
        largest = r
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)

def heap_sort(arr):
    n = len(arr)
    for i in range(n//2 - 1, -1, -1):
        heapify(arr, n, i)
    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)
    return arr

print(heap_sort([12, 11, 13, 5, 6, 7]))
```

```
[5, 6, 7, 11, 12, 13]
```

0.0.9 Experiment 9A: N-Queens Problem

[9]:
```
def is_safe(board, row, col, n):
    for i in range(row):
        if board[i][col] == 1:
            return False
    for i, j in zip(range(row-1, -1, -1), range(col-1, -1, -1)):
        if board[i][j] == 1:
            return False
    for i, j in zip(range(row-1, -1, -1), range(col+1, n)):
        if board[i][j] == 1:
            return False
    return True

def solve(board, row, n):
    if row == n:
        for r in board:
            print(r)
        print()
        return True
    for col in range(n):
        if is_safe(board, row, col, n):
            board[row][col] = 1
            if solve(board, row+1, n):
                return True
            board[row][col] = 0
    return False
```

```
n = 4
board = [[0]*n for _ in range(n)]
solve(board, 0, n)
```

```
[0, 1, 0, 0]
[0, 0, 0, 1]
[1, 0, 0, 0]
[0, 0, 1, 0]
```

[9]: True

0.0.10 Experiment 9B: Sum of Subsets Problem

[10]:
```
def subset_sum(arr, n, index, target, subset):
    if target == 0:
        print(subset)
        return
    if index == n or target < 0:
        return
    subset_sum(arr, n, index+1, target - arr[index], subset + [arr[index]])
    subset_sum(arr, n, index+1, target, subset)

arr = [3, 4, 5, 2]
target = 7
subset_sum(arr, len(arr), 0, target, [])
```

```
[3, 4]
[5, 2]
```