1.

```
def quicksort_first_pivot(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        print(f'Array after partitioning with pivot {arr[pi]}: {arr}')
        quicksort_first_pivot(arr, low, pi - 1)
        quicksort_first_pivot(arr, pi + 1, high)
    return arr

def partition(arr, low, high):
    pivot = arr[low]
    i = low + 1
    j = high

    while True:
        while i <= j and arr[i] <= pivot:
            i += 1
        while i <= j and arr[j] >= pivot:
            j -= 1
        if i <= j:
            arr[i], arr[j] = arr[j], arr[i]
        else:
            break

    arr[low], arr[j] = arr[j], arr[low]
    return j
arr1 = [10, 16, 8, 12, 15, 6, 3, 9, 5]
```

```
print(f'Sorted array: {quicksort_first_pivot(arr1, 0, len(arr1) - 1)}')
```

2.

```
def quicksort_middle_pivot(arr, low, high):
    if low < high:
        pi = partition_middle(arr, low, high)
        print(f'Array after partitioning with pivot {arr[pi]}: {arr}')
        quicksort_middle_pivot(arr, low, pi - 1)
        quicksort_middle_pivot(arr, pi + 1, high)
    return arr

def partition_middle(arr, low, high):
    mid = low + (high - low) // 2
    pivot = arr[mid]
    arr[mid], arr[low] = arr[low], arr[mid]
    return partition(arr, low, high)
arr4 = [19, 72, 35, 46, 58, 91, 22, 31]
print(f'Sorted array: {quicksort_middle_pivot(arr4, 0, len(arr4) - 1)}')
```

3.

```
def binary_search(arr, low, high, x, count=0):
    if high >= low:
        mid = (high + low) // 2
        count += 1
        if arr[mid] == x:
            return mid, count
```

```
        elif arr[mid] > x:
            return binary_search(arr, low, mid - 1, x, count)
        else:
            return binary_search(arr, mid + 1, high, x, count)
    else:
        return -1, count
arr7 = [5, 10, 15, 20, 25, 30, 35, 40, 45]
key = 20
index, comparisons = binary_search(arr7, 0, len(arr7) - 1, key)
print(f'Element {key} is at index {index}, Comparisons made: {comparisons}')
arr8 = [10, 20, 30, 40, 50, 60]
key = 50
index, comparisons = binary_search(arr8, 0, len(arr8) - 1, key)
print(f'Element {key} is at index {index}, Comparisons made: {comparisons}')
```

4.

```
def binary_search_with_steps(arr, low, high, x):
    steps = []
    while low <= high:
        mid = (low + high) // 2
        steps.append(mid)
        if arr[mid] == x:
            return mid, steps
        elif arr[mid] < x:
            low = mid + 1
        else:
            high = mid - 1
```

```
    return -1, steps
arr10 = [3, 9, 14, 19, 25, 31, 42, 47, 53]
key = 31
index, steps = binary_search_with_steps(arr10, 0, len(arr10) - 1, key)
print(f'Element {key} is at index {index}, Steps: {steps}')
arr11 = [13, 19, 24, 29, 35, 41, 42]
key = 42
index, steps = binary_search_with_steps(arr11, 0, len(arr11) - 1, key)
print(f'Element {key} is at index {index}, Steps: {steps}')
```

5.

```
def binary_search_with_steps(arr, low, high, x):
    steps = []
    while low <= high:
        mid = (low + high) // 2
        steps.append(mid)
        if arr[mid] == x:
            return mid, steps
        elif arr[mid] < x:
            low = mid + 1
        else:
            high = mid - 1
    return -1, steps
arr10 = [3, 9, 14, 19, 25, 31, 42, 47, 53]
key = 31
index, steps = binary_search_with_steps(arr10, 0, len(arr10) - 1, key)
print(f'Element {key} is at index {index}, Steps: {steps}')
```

```
arr11 = [13, 19, 24, 29, 35, 41, 42]
key = 42
index, steps = binary_search_with_steps(arr11, 0, len(arr11) - 1, key)
print(f'Element {key} is at index {index}, Steps: {steps}')
```

6.

```
import sys
def optimal_bst(keys, freq, n):
    cost = [[0 for x in range(n)] for y in range(n)]
    root = [[0 for x in range(n)] for y in range(n)]

    for i in range(n):
        cost[i][i] = freq[i]
        root[i][i] = i

    for L in range(2, n + 1):
        for i in range(n - L + 1):
            j = i + L - 1
            cost[i][j] = sys.maxsize

            for r in range(i, j + 1):
                c = 0
                if r > i:
                    c += cost[i][r - 1]
                if r < j:
                    c += cost[r + 1][j]
                c += sum(freq[i:j + 1])
```

7.

```
keys = [10, 12, 16, 21]
freq = [4, 2, 6, 3]
n = len(keys)
print(f'Cost of Optimal BST: {optimal_bst(keys, freq, n)}')
keys = [10, 12]
freq = [34, 50]
n = len(keys)
print(f'Cost of Optimal BST: {optimal_bst(keys, freq, n)}')
keys = [10, 12, 20]
freq = [34, 8, 50]
n = len(keys)
print(f'Cost of Optimal BST: {optimal_bst(keys, freq, n)}')
```