```python
import bisect

data = [2, 4, 5, 7, 8, 9, 12, 14, 17, 19, 22, 25, 27, 28, 33, 37]
target = 28


# Linear Search
def Linear_Search(data, target):
    for i in range(len(data)):
        if data[i] == target:
            return True
    return False


# Iterative Binary Search
def binary_search_iterative(data, target):
    low = 0
    high = len(data) - 1

    while low <= high:
        mid = (low + high) // 2
        if target == data[mid]:
            return True
        elif target < data[mid]:
            high = mid - 1
        else:
            low = mid + 1
    return False


# Binary Search Recursive
def binary_search_recursive(data, target, low, high):
    if low > high:
        return False
    else:
        mid = (low + high) // 2
        if target == data[mid]:
            return True
        elif target < data[mid]:
            return binary_search_recursive(data, target, low, mid - 1)
        else:
            return binary_search_recursive(data, target, mid + 1, high)


# A = [1, 2, 4, 5, 6, 6, 8, 9]
# A = [2, 5, 6, 7, 8, 8, 9]
target = 11


def find_closest_num(A, target):
    min_diff = float("inf")
    low = 0
    high = len(A) - 1
    closest_num = None

    # Edge cases for empty list or
    # when the list is only one element
    if len(A) == 0:
        return None
    if len(A) == 1:
        return A[0]

    while low <= high:
        mid = (low + high) // 2

        # Ensure we don't read beyond the bounds of the list
        # And obtain the left and right differences values
        if mid + 1 < len(A):
```

```
            min_diff_right = abs(A[mid + 1] - target)
        if mid > 0:
            min_diff_left = abs(A[mid - 1] - target)

        # check if the absolute value between left and right
        # elements are smaller than any seen prior
        if min_diff_left < min_diff:
            min_diff = min_diff_left
            closest_num = A[mid - 1]

        if min_diff_right < min_diff:
            min_diff = min_diff_right
            closest_num = A[mid + 1]

        # Move the mid point accordingly as is done
        # via binary search
        if A[mid] < target:
            low = mid + 1
        elif A[mid] > target:
            high = mid - 1
        # If the element is the target itself, the closest
        # number to it is itself
        else:
            return A[mid]

    return closest_num


# Fixed Point Problem
A1 = [-10, -5, 0, 3, 7]
A = [0, 2, 5, 8, 17]
A = [-10, -5, 3, 4, 7, 9]


# Time Complexity: O(n) Space Complexity: O(1)
def find_fixed_point_linear(A):
    for i in range(len(A)):
        if A[i] == i:
            return True
    return None


# Time Complexity: O(log n) Space Complexity: O(1)
def find_find_point_binary(A):
    low = 0
    high = len(A) - 1

    while low <= high:
        mid = (low + high) // 2

        if A[mid] < mid:
            low = mid + 1
        elif A[mid] > mid:
            high = mid - 1
        else:
            return A[mid]
    return None


# Function takes an array of sorted integers and a key and
# returns the index of the first occurence of that key from the array
AA = [-14, -10, 2, 108, 108, 243, 285, 285, 285, 401]
target = 108


def find_(A, target):
    for i in range(len(A)):
        if A[i] == target:
```

```
            return i
    return None


def find_binary_search(A, target):
    low = 0
    high = len(A) - 1
    while low <= high:
        mid = (low + high) // 2

        if A[mid] < target:
            low = mid + 1
        elif A[mid] > target:
            high = mid - 1
        else:
            if mid - 1 < 0:
                return mid
            if A[mid - 1] != target:
                return mid
            high = mid - 1
    return None


# print(find_binary_search(A5, target))


# Bitonic Sequence Peak
def find_highest_number_bitonic(A):
    low = 0
    high = len(A) - 1
    # Require atleast three elements for a valid bitonic sequence
    if len(A) < 3:
        return None

    while low <= high:
        mid = (low + high) // 2

        mid_left = A[mid - 1]
        mid_right = A[mid + 1]

        if mid_left < A[mid] < mid_right:
            low = mid + 1
        elif mid_left > A[mid] > mid_right:
            high = mid - 1
        else:
            return A[mid]


# Peak Element is "5"
A11 = [1, 2, 3, 4, 5, 4, 3, 2, 1]

# Peak Element is 4
A22 = [1, 2, 3, 4, 1]

# Peak Element is 6
A33 = [1, 6, 5, 4, 3, 2, 1]

# print(find_highest_number_bitonic(A33))

# Bisect module functions
# print(bisect.bisect_left(AA, 285))

# Integer square problem
k = 12  # Nearest Square Root


def integer_square_root(k):
    low = 0
```

```python
    high = k
    while low <= high:
        mid = (low + high) // 2
        mid_squared = mid * mid

        if mid_squared <= k:
            low = mid + 1
        else:
            high = mid - 1
    return low - 1


# print(integer_square_root(k))

# Cyclic Shift Array
# To find the smallest element in cylic Array
A_cyclic = [4, 5, 6, 7, 1, 2, 3]


def find(A):
    low = 0
    high = len(A) - 1

    while low < high:
        mid = (low + high) // 2

        if A[mid] > A[high]:
            low = mid + 1
        elif A[mid] <= A[high]:
            high = mid
    return low


idx = find(A_cyclic)
print(idx)
print(A_cyclic[idx])
```