PROBLEMS IN BINARY SEARCH

1. Count the number of rotations in a circularly sorted array

```
#include <stdio.h>

int count_rotations(int arr[], int n) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        int next = (mid + 1) % n;
        int prev = (mid + n - 1) % n;
        if (arr[mid] <= arr[next] && arr[mid] <= arr[prev]) {
            return mid;
        }
        if (arr[mid] >= arr[low]) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return 0;
}

int main() {
    int arr[] = {15, 18, 2, 3, 6, 12};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Rotations: %d\n", count_rotations(arr, n));
    return 0;
}
```

2. Search an element in a circularly sorted array

```
#include <stdio.h>

int search(int arr[], int low, int high, int target) {
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == target) return mid;
        if (arr[low] <= arr[mid]) {
            if (target >= arr[low] && target < arr[mid]) high = mid - 1;
            else low = mid + 1;
        } else {
            if (target > arr[mid] && target <= arr[high]) low = mid + 1;
            else high = mid - 1;
        }
    }
    return -1;
}

int main() {
    int arr[] = {15, 18, 2, 3, 6, 12};
    int n = sizeof(arr) / sizeof(arr[0]);
    int target = 6;
    printf("Element found at index: %d\n", search(arr, 0, n - 1, target));
    return 0;
}
```

3. Find the first occurrence of a given number in a sorted array

```
#include <stdio.h>
```

```
int first_occurrence(int arr[], int low, int high, int target) {
    int result = -1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == target) {
            result = mid;
            high = mid - 1;
        } else if (arr[mid] < target) low = mid + 1;
        else high = mid - 1;
    }
    return result;
}

int main() {
    int arr[] = {1, 2, 2, 2, 3, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    int target = 2;
    printf("First occurrence at index: %d\n", first_occurrence(arr, 0, n - 1, target));
    return 0;
}
```

4. Find the last occurrence of a given number in a sorted array

```
#include <stdio.h>

int last_occurrence(int arr[], int low, int high, int target) {
    int result = -1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == target) {
            result = mid;
            low = mid + 1;
        } else if (arr[mid] < target) low = mid + 1;
        else high = mid - 1;
    }
    return result;
}

int main() {
    int arr[] = {1, 2, 2, 2, 3, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    int target = 2;
    printf("Last occurrence at index: %d\n", last_occurrence(arr, 0, n - 1, target));
    return 0;
}
```

5. Count occurrences of a number in a sorted array with duplicates

```
#include <stdio.h>

int count_occurrences(int arr[], int n, int target) {
    int first = first_occurrence(arr, 0, n - 1, target);
    if (first == -1) return 0;
    int last = last_occurrence(arr, 0, n - 1, target);
    return last - first + 1;
}

int main() {
    int arr[] = {1, 2, 2, 2, 3, 4, 5};
```

```
    int n = sizeof(arr) / sizeof(arr[0]);
    int target = 2;
    printf("Occurrences: %d\n", count_occurrences(arr, n, target));
    return 0;
}
```

6. Find the smallest element in a rotated sorted array

```
#include <stdio.h>

int find_min(int arr[], int n) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] <= arr[high]) high = mid - 1;
        else low = mid + 1;
    }
    return arr[low];
}

int main() {
    int arr[] = {15, 18, 2, 3, 6, 12};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Smallest element: %d\n", find_min(arr, n));
    return 0;
}
```

7. Find the floor and ceil of a number in a sorted array

```
#include <stdio.h>

void find_floor_ceil(int arr[], int n, int x) {
    int low = 0, high = n - 1;
    int floor = -1, ceil = -1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == x) {
            floor = arr[mid];
            ceil = arr[mid];
            break;
        }
        if (arr[mid] < x) {
            floor = arr[mid];
            low = mid + 1;
        } else {
            ceil = arr[mid];
            high = mid - 1;
        }
    }
    printf("Floor: %d, Ceil: %d\n", floor, ceil);
}

int main() {
    int arr[] = {1, 2, 8, 10, 10, 12, 19};
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 5;
    find_floor_ceil(arr, n, x);
    return 0;
```

```
}
```

8. Search in a nearly sorted array in logarithmic time

```
#include <stdio.h>

int search_nearly_sorted(int arr[], int n, int target) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == target) return mid;
        if (mid - 1 >= low && arr[mid - 1] == target) return mid - 1;
        if (mid + 1 <= high && arr[mid + 1] == target) return mid + 1;
        if (arr[mid] < target) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}

int main() {
    int arr[] = {5, 10, 30, 20, 40};
    int n = sizeof(arr) / sizeof(arr[0]);
    int target = 30;
    printf("Element found at index: %d\n", search_nearly_sorted(arr, n, target));
    return 0;
}
```

9. Find the number of 1's in a sorted binary array

```
#include <stdio.h>

int count_ones(int arr[], int n) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == 1) {
            if (mid == n - 1 || arr[mid + 1] == 0) return mid + 1;
            else low = mid + 1;
        } else high = mid - 1;
    }
    return 0;
}

int main() {
    int arr[] = {0, 0, 1, 1, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Number of 1's: %d\n", count_ones(arr, n));
    return 0;
}
```

10. Find the peak element in an array

```
#include <stdio.h>

int find_peak(int arr[], int n) {
    int low = 0, high = n - 1;
    while (low <= high) {
```

```
        int mid = (low + high) / 2;
        if ((mid == 0 || arr[mid - 1] <= arr[mid]) && (mid == n - 1 || arr[mid + 1] <= arr[mid])) {
            return mid;
        } else if (mid > 0 && arr[mid - 1] > arr[mid]) {
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }
    return -1;
}

int main() {
    int arr[] = {1, 3, 20, 4, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Peak element at index: %d\n", find_peak(arr, n));
    return 0;
}
```

11. Find the missing element in a sequence in logarithmic time

```
#include <stdio.h>

int find_missing(int arr[], int n) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] != mid + 1) high = mid - 1;
        else low = mid + 1;
    }
    return low + 1;
}

int main() {
    int arr[] = {1, 2, 3, 4, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Missing element: %d\n", find_missing(arr, n));
    return 0;
}
```

12. Find floor and ceil of a number in a sorted array (recursive solution)

```
#include <stdio.h>

void find_floor_ceil_recursive(int arr[], int low, int high, int x, int *floor, int *ceil) {
    if (low > high) return;
    int mid = (low + high) / 2;
    if (arr[mid] == x) {
        *floor = *ceil = arr[mid];
        return;
    }
    if (arr[mid] < x) {
        *floor = arr[mid];
        find_floor_ceil_recursive(arr, mid + 1, high, x, floor, ceil);
    } else {
        *ceil = arr[mid];
        find_floor_ceil_recursive(arr, low, mid - 1, x, floor, ceil);
    }
```

```
}

int main() {
    int arr[] = {1, 2, 8, 10, 10, 12, 19};
    int n = sizeof(arr) / sizeof(arr[0]);
    int floor = -1, ceil = -1;
    find_floor_ceil_recursive(arr, 0, n - 1, 5, &floor, &ceil);
    printf("Floor: %d, Ceil: %d\n", floor, ceil);
    return 0;
}
```

13. Find the frequency of each element in a sorted array containing duplicates

```
#include <stdio.h>

void find_frequency(int arr[], int n) {
    int low = 0;
    while (low < n) {
        int target = arr[low];
        int first = first_occurrence(arr, 0, n - 1, target);
        int last = last_occurrence(arr, 0, n - 1, target);
        printf("%d occurs %d times\n", target, last - first + 1);
        low = last + 1;
    }
}

int main() {
    int arr[] = {1, 2, 2, 2, 3, 3, 4, 4, 4};
    int n = sizeof(arr) / sizeof(arr[0]);
    find_frequency(arr, n);
    return 0;
}
```

14. Find the square root of a number using binary search

```
#include <stdio.h>

double square_root(int n) {
    double low = 0, high = n, mid;
    while (high - low > 0.0001) {
        mid = (low + high) / 2;
        if (mid * mid > n) high = mid;
        else low = mid;
    }
    return (low + high) / 2;
}

int main() {
    int n = 16;
    printf("Square root: %lf\n", square_root(n));
    return 0;
}
```

15. Division of two numbers using binary search algorithm

```
#include <stdio.h>
```

```
int division(int dividend, int divisor) {
    int low = 0, high = dividend;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (mid * divisor == dividend) return mid;
        if (mid * divisor < dividend) low = mid + 1;
        else high = mid - 1;
    }
    return high;
}

int main() {
    int dividend = 10, divisor = 2;
    printf("Result: %d\n", division(dividend, divisor));
    return 0;
}
```

16. Find the odd occurring element in an array in logarithmic time

```
#include <stdio.h>

int find_odd_occurrence(int arr[], int n) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (mid % 2 == 0) {
            if (arr[mid] == arr[mid + 1]) low = mid + 2;
            else high = mid - 1;
        } else {
            if (arr[mid] == arr[mid - 1]) low = mid + 1;
            else high = mid - 1;
        }
    }
    return arr[low];
}

int main() {
    int arr[] = {1, 1, 2, 2, 3};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Odd occurring element: %d\n", find_odd_occurrence(arr, n));
    return 0;
}
```

17. Find pairs with a difference in an array (constant space solution)

```
#include <stdio.h>

void find_pairs_with_difference(int arr[], int n, int diff) {
    int low = 0, high = 1;
    while (high < n) {
        int d = arr[high] - arr[low];
        if (d == diff) {
            printf("Pair: (%d, %d)\n", arr[low], arr[high]);
            low++;
            high++;
        } else if (d < diff) high++;
        else low++;
    }
```

```
}

int main() {
    int arr[] = {1, 5, 9, 12};
    int n = sizeof(arr) / sizeof(arr[0]);
    int diff = 4;
    find_pairs_with_difference(arr, n, diff);
    return 0;
}
```

18. Find k closest elements to a given value in an array

```
#include <stdio.h>

void find_k_closest_elements(int arr[], int n, int k, int x) {
    int low = 0, high = n - 1;
    while (high - low >= k) {
        if (x - arr[low] <= arr[high] - x) high--;
        else low++;
    }
    for (int i = low; i <= high; i++) printf("%d ", arr[i]);
}

int main() {
    int arr[] = {1, 3, 5, 7, 9, 11};
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3, x = 6;
    find_k_closest_elements(arr, n, k, x);
    return 0;
}
```