Problems on Binary Search

1. Find the number of rotations in a circularly sorted array
2. Search an element in a circularly sorted array
3. Find the first or last occurrence of a given number in a sorted array
4. Count occurrences of a number in a sorted array with duplicates
5. Find the smallest missing element from a sorted array
6. Find floor and ceil of a number in a sorted integer array
7. Search in a nearly sorted array in logarithmic time
8. Find the number of 1's in a sorted binary array
9. Find the peak element in an array
10. Find the missing term in a sequence in logarithmic time
11. Find floor and ceil of a number in a sorted array (Recursive solution
12. Find the frequency of each element in a sorted array containing duplicates
13. Find the square root of a number using a binary search
14. Division of two numbers using binary search algorithm,
15. Find the odd occurring element in an array in logarithmic time
16. Find pairs with difference k in an array| Constant Space Solution
17. Find 'k' closest element to a given value in an array.

1 to 8

```
#include <stdio.h>


// 1. Find the number of rotations in a circularly sorted array
int findRotations(int arr[], int n) {
    int low = 0, high = n - 1;
    while (low <= high) {
        if (arr[low] <= arr[high]) return low;
        int mid = (low + high) / 2;
        int next = (mid + 1) % n;
        int prev = (mid - 1 + n) % n;
        if (arr[mid] <= arr[next] && arr[mid] <= arr[prev]) return mid;
```

```
        if (arr[mid] <= arr[high]) high = mid - 1;
        else low = mid + 1;
    }
    return 0;
}

// 2. Search an element in a circularly sorted array
int searchCircular(int arr[], int n, int key) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == key) return mid;
        if (arr[low] <= arr[mid]) {
            if (key >= arr[low] && key < arr[mid]) high = mid - 1;
            else low = mid + 1;
        } else {
            if (key > arr[mid] && key <= arr[high]) low = mid + 1;
            else high = mid - 1;
        }
    }
    return -1;
}

// 3. Find the first or last occurrence of a given number in a sorted array
int findOccurrence(int arr[], int n, int key, int findFirst) {
    int low = 0, high = n - 1, result = -1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == key) {
            result = mid;
            if (findFirst) high = mid - 1;
```

```
            else low = mid + 1;
        } else if (arr[mid] < key) low = mid + 1;
        else high = mid - 1;
    }
    return result;
}

// 4. Count occurrences of a number in a sorted array with duplicates
int countOccurrences(int arr[], int n, int key) {
    int first = findOccurrence(arr, n, key, 1);
    if (first == -1) return 0;
    int last = findOccurrence(arr, n, key, 0);
    return last - first + 1;
}

// 5. Find the smallest missing element from a sorted array
int smallestMissing(int arr[], int n) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] != mid) high = mid - 1;
        else low = mid + 1;
    }
    return low;
}

// 6. Find floor and ceil of a number in a sorted integer array
void findFloorCeil(int arr[], int n, int key, int *floor, int *ceil) {
    *floor = -1, *ceil = -1;
    int low = 0, high = n - 1;
    while (low <= high) {
```

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

// 7. Search in a nearly sorted array in logarithmic time
int searchNearlySorted(int arr[], int n, int key) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == key) return mid;
        if (mid > low && arr[mid - 1] == key) return mid - 1;
        if (mid < high && arr[mid + 1] == key) return mid + 1;
        if (arr[mid] < key) low = mid + 2;
        else high = mid - 2;
    }
    return -1;
}

// 8. Find the number of 1's in a sorted binary array
int countOnes(int arr[], int n) {
    int low = 0, high = n - 1;
```

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

// Main function
int main() {
    int arr1[] = {15, 18, 2, 3, 6, 12};
    int n1 = sizeof(arr1) / sizeof(arr1[0]);

    printf("Number of rotations: %d\n", findRotations(arr1, n1));

    int key = 6;
    printf("Element %d found at index: %d\n", key, searchCircular(arr1, n1, key));

    int arr2[] = {1, 2, 2, 2, 3, 4, 5};
    int n2 = sizeof(arr2) / sizeof(arr2[0]);
    key = 2;
    printf("First occurrence of %d: %d\n", key, findOccurrence(arr2, n2, key, 1));
    printf("Last occurrence of %d: %d\n", key, findOccurrence(arr2, n2, key, 0));
    printf("Count of %d: %d\n", key, countOccurrences(arr2, n2, key));

    int arr3[] = {0, 1, 2, 6, 9};
    int n3 = sizeof(arr3) / sizeof(arr3[0]);
    printf("Smallest missing element: %d\n", smallestMissing(arr3, n3));

    int floor, ceil;
```

```
    int arr4[] = {1, 2, 8, 10, 10, 12, 19};
    int n4 = sizeof(arr4) / sizeof(arr4[0]);
    key = 5;
    findFloorCeil(arr4, n4, key, &floor, &ceil);
    printf("Floor of %d: %d, Ceil of %d: %d\n", key, floor, key, ceil);

    int arr5[] = {10, 3, 40, 20, 50, 80, 70};
    int n5 = sizeof(arr5) / sizeof(arr5[0]);
    key = 40;
    printf("Element %d found at index (nearly sorted array): %d\n", key, searchNearlySorted(arr5, n5,
key));

    int arr6[] = {0, 0, 0, 1, 1, 1, 1};
    int n6 = sizeof(arr6) / sizeof(arr6[0]);
    printf("Number of 1's: %d\n", countOnes(arr6, n6));

    return 0;
}
```

9 to 14

```
#include <stdio.h>

// 9. Find the peak element in an array
int findPeakElement(int arr[], int n) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if ((mid == 0 || arr[mid] >= arr[mid - 1]) &&
            (mid == n - 1 || arr[mid] >= arr[mid + 1]))
```

```
            return mid;
        if (mid > 0 && arr[mid - 1] > arr[mid])
            high = mid - 1;
        else
            low = mid + 1;
    }
    return -1;
}

// 10. Find the missing term in a sequence in logarithmic time
int findMissingTerm(int arr[], int n) {
    int low = 0, high = n - 1;
    int diff = (arr[n - 1] - arr[0]) / n;
    while (low <= high) {
        int mid = (low + high) / 2;
        if ((arr[mid] != arr[0] + mid * diff) &&
            (arr[mid - 1] == arr[0] + (mid - 1) * diff))
            return arr[0] + mid * diff;
        if (arr[mid] == arr[0] + mid * diff)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}

// 11. Find floor and ceil of a number in a sorted array (Recursive solution)
void findFloorCeilRecursive(int arr[], int low, int high, int key, int *floor, int *ceil) {
    if (low > high) return;
    int mid = (low + high) / 2;
    if (arr[mid] == key) {
```

```
        *floor = *ceil = arr[mid];
        return;
    }
    if (arr[mid] < key) {
        *floor = arr[mid];
        findFloorCeilRecursive(arr, mid + 1, high, key, floor, ceil);
    } else {
        *ceil = arr[mid];
        findFloorCeilRecursive(arr, low, mid - 1, key, floor, ceil);
    }
}

// 12. Find the frequency of each element in a sorted array containing duplicates
void findFrequency(int arr[], int n) {
    int i = 0;
    while (i < n) {
        int count = 1;
        while (i + 1 < n && arr[i] == arr[i + 1]) {
            count++;
            i++;
        }
        printf("Element %d appears %d times\n", arr[i], count);
        i++;
    }
}

// 13. Find the square root of a number using binary search
int findSquareRoot(int num) {
    int low = 0, high = num, ans = -1;
    while (low <= high) {
        int mid = (low + high) / 2;
```

```
        if (mid * mid == num) return mid;
        if (mid * mid < num) {
            ans = mid;
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return ans;
}

// 14. Division of two numbers using binary search algorithm
int divide(int dividend, int divisor) {
    if (divisor == 0) return -1; // Division by zero
    int low = 0, high = dividend, ans = 0;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (mid * divisor <= dividend) {
            ans = mid;
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return ans;
}

// Main function
int main() {
    int arr1[] = {1, 3, 20, 4, 1, 0};
    int n1 = sizeof(arr1) / sizeof(arr1[0]);
```

```
    printf("Peak element index: %d\n", findPeakElement(arr1, n1));

    int arr2[] = {2, 4, 6, 8, 10, 14};
    int n2 = sizeof(arr2) / sizeof(arr2[0]);
    printf("Missing term: %d\n", findMissingTerm(arr2, n2));

    int arr3[] = {1, 2, 8, 10, 10, 12, 19};
    int n3 = sizeof(arr3) / sizeof(arr3[0]);
    int floor = -1, ceil = -1;
    int key = 5;
    findFloorCeilRecursive(arr3, 0, n3 - 1, key, &floor, &ceil);
    printf("Floor of %d: %d, Ceil of %d: %d\n", key, floor, key, ceil);

    int arr4[] = {2, 2, 2, 3, 3, 4, 5, 5, 5};
    int n4 = sizeof(arr4) / sizeof(arr4[0]);
    printf("Frequencies:\n");
    findFrequency(arr4, n4);

    int num = 16;
    printf("Square root of %d: %d\n", num, findSquareRoot(num));

    int dividend = 22, divisor = 7;
    printf("%d divided by %d = %d\n", dividend, divisor, divide(dividend, divisor));

    return 0;
}
```

15 to 17

```
#include <stdio.h>

// 15. Find the odd occurring element in an array in logarithmic time
```

```
int findOddOccurring(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;
        } else {
            if (arr[mid] == arr[mid - 1])
                low = mid + 1;
            else
                high = mid - 1;
        }
    }
    return arr[low];
}

// 16. Find pairs with difference k in an array (Constant Space Solution)
void findPairsWithDifferenceK(int arr[], int n, int k) {
    int i = 0, j = 1;
    while (i < n && j < n) {
        if (i != j && arr[j] - arr[i] == k) {
            printf("Pair: (%d, %d)\n", arr[i], arr[j]);
            i++;
            j++;
        } else if (arr[j] - arr[i] < k) {
            j++;
        } else {
            i++;
```

```
        }
    }
}

// 17. Find 'k' closest elements to a given value in an array
void findKClosestElements(int arr[], int n, int key, int k) {
    int low = 0, high = n - 1;
    while (high - low >= k) {
        if (abs(arr[low] - key) > abs(arr[high] - key))
            low++;
        else
            high--;
    }
    printf("The %d closest elements to %d are: ", k, key);
    for (int i = low; i <= high; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

// Main function
int main() {
    // 15. Find the odd occurring element in an array in logarithmic time
    int arr1[] = {1, 1, 2, 2, 3, 4, 4, 5, 5};
    int n1 = sizeof(arr1) / sizeof(arr1[0]);
    printf("Odd occurring element: %d\n", findOddOccurring(arr1, n1));

    // 16. Find pairs with difference k in an array (Constant Space Solution)
    int arr2[] = {1, 3, 5, 7, 9};
    int n2 = sizeof(arr2) / sizeof(arr2[0]);
    int k = 2;
```

```
    printf("Pairs with difference %d:\n", k);
    findPairsWithDifferenceK(arr2, n2, k);

    // 17. Find 'k' closest elements to a given value in an array
    int arr3[] = {1, 2, 3, 4, 5, 6, 7};
    int n3 = sizeof(arr3) / sizeof(arr3[0]);
    int key = 5;
    k = 3;
    findKClosestElements(arr3, n3, key, k);

    return 0;
}
```