practice-1

November 25, 2024

1 Practice-1

[6]:
```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;
```

Find the number of rotations in a circularly sorted array

[3]:
```cpp
int countRotations(int arr[], int n) {
    int low = 0, high = n - 1;

    while (low <= high) {
        if (arr[low] <= arr[high]) {
            return low;
        }

        int mid = low + (high - low) / 2;
        int next = (mid + 1) % n;
        int prev = (mid - 1 + n) % n;

        if (arr[mid] <= arr[next] && arr[mid] <= arr[prev]) {
            return mid;
        } else if (arr[mid] <= arr[high]) {
            low = mid + 1;
        } else if (arr[mid] >= arr[low]) {
            high = mid - 1;
        }
    }

    return -1;
}


int arr[] = {15, 18, 2, 3, 6, 12};
int n = sizeof(arr) / sizeof(arr[0]);
int rotations = countRotations(arr, n);
cout << "The array is rotated " << rotations << " times." << endl;
```

```
return 0;
```

```
The array is rotated 2 times.
```

Search an element in a circularly sorted array

[4]:
```
int searchInRotatedArray(vector<int>& arr, int key) {
    int low = 0, high = arr.size() - 1;

    while (low <= high) {
        int mid = low + (high - low) / 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;
}

vector<int> arr = {15, 18, 2, 3, 6, 12};
int key = 3;
int index = searchInRotatedArray(arr, key);
if (index != -1) {
    cout << "Element found at index " << index << endl;
} else {
    cout << "Element not found in the array" << endl;
}
```

```
Element found at index 3
```

Find the first or last occurrence of a given number in a sorted array

[ ]:
```
int findFirstOccurrence(vector<int>& arr, int key) {
    int low = 0, high = arr.size() - 1;
    int result = -1;

    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (arr[mid] == key) {
            result = mid;
            high = mid - 1;
        } else if (arr[mid] > key) {
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }

    return result;
}
```

[ ]:
```
int findLastOccurrence(vector<int>& arr, int key) {
    int low = 0, high = arr.size() - 1;
    int result = -1;

    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (arr[mid] == key) {
            result = mid;
            low = mid + 1;
        } else if (arr[mid] > key) {
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }

    return result;
}
```

[11]:
```
vector<int> arr = {2, 4, 10, 10, 10, 18, 20};
int key = 10;
int firstIndex = findFirstOccurrence(arr, key);
int lastIndex = findLastOccurrence(arr, key);

if (firstIndex != -1) {
    cout << "First occurrence of element " << key << " is at index " <<␣
  ↪firstIndex << endl;
} else {
    cout << "Element not found in the array" << endl;
}

if (lastIndex != -1) {
    cout << "Last occurrence of element " << key << " is at index " <<␣
  ↪lastIndex << endl;
} else {
    cout << "Element not found in the array" << endl;
}
```

```
First occurrence of element 10 is at index 2
Last occurrence of element 10 is at index 4
Last occurrence of element 10 is at index 4
```

Count occurrences of a number in a sorted array with duplicates

[12]:
```
int countOccurrences(vector<int>& arr, int key) {
    int firstIndex = findFirstOccurrence(arr, key);
    if (firstIndex == -1) {
        return 0;
    }
    int lastIndex = findLastOccurrence(arr, key);
    return lastIndex - firstIndex + 1;
}

vector<int> arr = {2, 4, 10, 10, 10, 18, 20};
int key = 10;
int count = countOccurrences(arr, key);
cout << "Element " << key << " occurs " << count << " times in the array." <<␣
  ↪endl;
```

```
Element 10 occurs 3 times in the array.
```

Find the smallest missing element from a sorted array

[13]:
```
int findSmallestMissingElement(vector<int>& arr) {
    int low = 0, high = arr.size() - 1;

    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (arr[mid] != mid) {
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }

    return low;
}

vector<int> arr = {0, 1, 2, 6, 9, 11, 15};
int missingElement = findSmallestMissingElement(arr);
cout << "The smallest missing element is " << missingElement << endl;
```

```
The smallest missing element is 3
```

Find floor and ceil of a number in a sorted integer array

[24]:
```
vector<int> arr = {1, 2, 8, 10, 10, 12, 19};
int key = 5;
int n = arr.size();
int floor = -1, ceil = -1;

int low = 0, high = n - 1;
while (low <= high) {
    int mid = low + (high - low) / 2;

    if (arr[mid] == key) {
        floor = arr[mid];
        ceil = arr[mid];
        break;
    } else if (arr[mid] < key) {
        floor = arr[mid];
        low = mid + 1;
    } else {
        ceil = arr[mid];
        high = mid - 1;
    }
}

cout << "Floor of " << key << " is " << floor << endl;
cout << "Ceil of " << key << " is " << ceil << endl;
```

```
Floor of 5 is 2
Ceil of 5 is 8
Ceil of 5 is 8
```

Search in a nearly sorted array in logarithmic time

[25]:
```
int searchInNearlySortedArray(vector<int>& arr, int key) {
    int low = 0, high = arr.size() - 1;

    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (arr[mid] == key) {
            return mid;
        } else if (mid > low && arr[mid - 1] == key) {
            return mid - 1;
        } else if (mid < high && arr[mid + 1] == key) {
            return mid + 1;
        }

        if (arr[mid] > key) {
            high = mid - 2;
        } else {
            low = mid + 2;
        }
    }

    return -1;
}

vector<int> arr = {10, 3, 40, 20, 50, 80, 70};
int key = 40;
int index = searchInNearlySortedArray(arr, key);
if (index != -1) {
    cout << "Element found at index " << index << endl;
} else {
    cout << "Element not found in the array" << endl;
}
```

```
Element found at index 2
```

Find the number of 1s in a sorted binary array

[27]:
```
int countOnes(vector<int>& arr) {
    int low = 0, high = arr.size() - 1;
    int firstOneIndex = -1;

    while (low <= high) {
        int mid = low + (high - low) / 2;

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

    return arr.size() - firstOneIndex;
}

vector<int> arr = {0, 0, 0, 1, 1, 1, 1};
int count = countOnes(arr);
cout << "Number of 1s in the array is " << count << endl;
```

```
Number of 1s in the array is 4
```

Find the peak element in an array

[29]:
```
int findPeakElement(vector<int>& arr) {
    int low = 0, high = arr.size() - 1;

    while (low < high) {
        int mid = low + (high - low) / 2;

        if (arr[mid] > arr[mid + 1]) {
            high = mid;
        } else {
            low = mid + 1;
        }
    }

    return low;
}

vector<int> arr = {1, 3, 20, 4, 1, 0};
int peakIndex = findPeakElement(arr);
cout << "Peak element is at index " << peakIndex << " with value " <<␣
 ↪arr[peakIndex] << endl;
```

```
Peak element is at index 2 with value 20
```

Find the missing term in a sequence in logarithmic time

[31]:
```
int findMissingTerm(vector<int>& arr) {
    int low = 0, high = arr.size() - 1;
    int diff = (arr[high] - arr[low]) / arr.size();

    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (arr[mid] == arr[0] + mid * diff) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    return arr[0] + low * diff;
}

vector<int> arr = {2, 4, 6, 8, 12, 14};
int missingTerm = findMissingTerm(arr);
cout << "The missing term in the sequence is " << missingTerm << endl;
```

```
The missing term in the sequence is 10
```

Find the floor and ceil of a number in a sorted array (Recursive solution)

[32]:
```
int findFloorRecursive(vector<int>& arr, int low, int high, int key) {
    if (low > high) {
        return -1;
    }

    if (key >= arr[high]) {
        return arr[high];
    }

    int mid = low + (high - low) / 2;

    if (arr[mid] == key) {
        return arr[mid];
    }

    if (mid > 0 && arr[mid - 1] <= key && key < arr[mid]) {
        return arr[mid - 1];
    }

    if (key < arr[mid]) {
        return findFloorRecursive(arr, low, mid - 1, key);
    }

    return findFloorRecursive(arr, mid + 1, high, key);
}
```

[33]:
```
int findCeilRecursive(vector<int>& arr, int low, int high, int key) {
    if (low > high) {
        return -1;
    }

    if (key <= arr[low]) {
        return arr[low];
    }

    int mid = low + (high - low) / 2;

    if (arr[mid] == key) {
        return arr[mid];
    }

    if (mid < high && arr[mid] < key && key <= arr[mid + 1]) {
        return arr[mid + 1];
    }

    if (key < arr[mid]) {
        return findCeilRecursive(arr, low, mid - 1, key);
    }

    return findCeilRecursive(arr, mid + 1, high, key);
}
```

[34]:
```
vector<int> arr = {1, 2, 8, 10, 10, 12, 19};
int key = 5;
int n = arr.size();

int floor = findFloorRecursive(arr, 0, n - 1, key);
int ceil = findCeilRecursive(arr, 0, n - 1, key);

cout << "Floor of " << key << " is " << floor << endl;
cout << "Ceil of " << key << " is " << ceil << endl;
```

```
Floor of 5 is 2
Ceil of 5 is 8
Ceil of 5 is 8
```

Find the square root of a number using binary search

[ ]:
```
#include <iostream>
using namespace std;

int x = 25;
if (x == 0 || x == 1) {
    cout << "Square root of " << x << " is " << x << endl;
} else {
    int low = 1, high = x, ans = 0;
    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (mid * mid == x) {
```

```cpp
            ans = mid;
            break;
        }

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

    cout << "Square root of " << x << " is " << ans << endl;
}
```

```
Square root of 25 is 5
```