Binary Search Cheat Sheet (C++)

1. Basics

Binary search is an O(log n) algorithm for searching sorted data. It halves the search space each step by comparing the middle element with the target.

```
// Iterative binary search (C++)
int binary_search_iterative(const vector<int>& arr, int target) {
    int l = 0, r = (int)arr.size() - 1;
    while (l <= r) {
        int mid = l + (r - l) / 2;
        if (arr[mid] == target) return mid;
        else if (arr[mid] < target) l = mid + 1;
        else r = mid - 1;
    }
    return -1;
}
```

2. Variants

Common variations used in practice:

• Lower Bound

```
int lower_bound_idx(const vector<int>& arr, int x) {
    int l = 0, r = (int)arr.size();
    while (l < r) {
        int mid = l + (r - l) / 2;
        if (arr[mid] < x) l = mid + 1;
        else r = mid;
    }
    return l;
}
```

• Upper Bound

```
int upper_bound_idx(const vector<int>& arr, int x) {
    int l = 0, r = (int)arr.size();
    while (l < r) {
        int mid = l + (r - l) / 2;
        if (arr[mid] <= x) l = mid + 1;
        else r = mid;
    }
    return l;
}
```

• Rotated Sorted Array Search

```
int search_in_rotated(const vector<int>& nums, int target) {
    int l = 0, r = nums.size() - 1;
    while (l <= r) {
        int mid = l + (r - l) / 2;
        if (nums[mid] == target) return mid;
        if (nums[l] <= nums[mid]) {
            if (nums[l] <= target && target < nums[mid]) r = mid - 1;
            else l = mid + 1;
        } else {
            if (nums[mid] < target && target <= nums[r]) l = mid + 1;
            else r = mid - 1;
        }
    }
    return -1;
}
```

3. Generic Predicate Template

```
// Find first x in [lo, hi] such that predicate P(x) is true
long long first_true(long long lo, long long hi, function<bool(long long)> P) {
```

```
    while (lo < hi) {
        long long mid = lo + (hi - lo) / 2;
        if (P(mid)) hi = mid;
        else lo = mid + 1;
    }
    return lo;
}
```