# C++ Implementations

## 1. Lower Bound and Upper Bound

#include <iostream>  
#include <vector>  
#include <algorithm>  
using namespace std;  
  
int main() {  
 vector<int> arr = {1, 2, 4, 4, 5, 6, 7, 8, 10};  
   
 int key = 4;  
  
 // Lower Bound  
 auto lb = lower\_bound(arr.begin(), arr.end(), key);  
 cout << "Lower Bound of " << key << ": " << (lb - arr.begin()) << endl;  
  
 // Upper Bound  
 auto ub = upper\_bound(arr.begin(), arr.end(), key);  
 cout << "Upper Bound of " << key << ": " << (ub - arr.begin()) << endl;  
  
 return 0;  
}

## 2. LeetCode Problems

### LeetCode 1011 - Capacity to Ship Packages Within D Days

#include <iostream>  
#include <vector>  
using namespace std;  
  
bool canShipInDays(vector<int>& weights, int capacity, int days) {  
 int currentLoad = 0, requiredDays = 1;  
 for (int weight : weights) {  
 if (currentLoad + weight > capacity) {  
 requiredDays++;  
 currentLoad = 0;  
 }  
 currentLoad += weight;  
 }  
 return requiredDays <= days;  
}  
  
int shipWithinDays(vector<int>& weights, int days) {  
 int left = \*max\_element(weights.begin(), weights.end());  
 int right = accumulate(weights.begin(), weights.end(), 0);  
   
 while (left < right) {  
 int mid = left + (right - left) / 2;  
 if (canShipInDays(weights, mid, days)) {  
 right = mid;  
 } else {  
 left = mid + 1;  
 }  
 }  
 return left;  
}

### Aggressive Cow Problem

#include <iostream>  
#include <vector>  
#include <algorithm>  
using namespace std;  
  
bool canPlaceCows(vector<int>& stalls, int dist, int cows) {  
 int lastCow = stalls[0], count = 1;  
 for (int i = 1; i < stalls.size(); i++) {  
 if (stalls[i] - lastCow >= dist) {  
 count++;  
 lastCow = stalls[i];  
 }  
 if (count == cows) return true;  
 }  
 return false;  
}  
  
int aggressiveCows(vector<int>& stalls, int cows) {  
 sort(stalls.begin(), stalls.end());  
 int low = 1, high = stalls.back() - stalls.front();  
   
 while (low <= high) {  
 int mid = low + (high - low) / 2;  
 if (canPlaceCows(stalls, mid, cows)) {  
 low = mid + 1;  
 } else {  
 high = mid - 1;  
 }  
 }  
 return high;  
}

### Painters Partition Problem

#include <iostream>  
#include <vector>  
#include <numeric>  
using namespace std;  
  
bool canPaintInTime(vector<int>& boards, int painters, int maxTime) {  
 int painterCount = 1, currentTime = 0;  
 for (int board : boards) {  
 if (currentTime + board > maxTime) {  
 painterCount++;  
 currentTime = 0;  
 }  
 currentTime += board;  
 }  
 return painterCount <= painters;  
}  
  
int paintersPartition(vector<int>& boards, int painters) {  
 int left = \*max\_element(boards.begin(), boards.end());  
 int right = accumulate(boards.begin(), boards.end(), 0);  
   
 while (left < right) {  
 int mid = left + (right - left) / 2;  
 if (canPaintInTime(boards, painters, mid)) {  
 right = mid;  
 } else {  
 left = mid + 1;  
 }  
 }  
 return left;  
}

## 3. Quick Select - Kth Smallest Element

#include <iostream>  
#include <vector>  
using namespace std;  
  
int partition(vector<int>& arr, int left, int right) {  
 int pivot = arr[right];  
 int i = left;  
 for (int j = left; j < right; j++) {  
 if (arr[j] <= pivot) {  
 swap(arr[i], arr[j]);  
 i++;  
 }  
 }  
 swap(arr[i], arr[right]);  
 return i;  
}  
  
int quickSelect(vector<int>& arr, int left, int right, int k) {  
 if (left == right) return arr[left];  
   
 int pivotIndex = partition(arr, left, right);  
 if (k == pivotIndex) return arr[k];  
 else if (k < pivotIndex) return quickSelect(arr, left, pivotIndex - 1, k);  
 else return quickSelect(arr, pivotIndex + 1, right, k);  
}

## 4. Minimum Time to Get the Job Done (Garden Worker Problem)

#include <iostream>  
#include <vector>  
#include <algorithm>  
using namespace std;  
  
bool canComplete(vector<int>& gardens, int workers, int maxTime) {  
 int totalTime = 0;  
 for (int garden : gardens) {  
 totalTime += garden / workers;  
 if (garden % workers != 0) totalTime++;  
 if (totalTime > maxTime) return false;  
 }  
 return true;  
}  
  
int minTime(vector<int>& gardens, int workers) {  
 int left = 1, right = \*max\_element(gardens.begin(), gardens.end());  
   
 while (left < right) {  
 int mid = left + (right - left) / 2;  
 if (canComplete(gardens, workers, mid)) {  
 right = mid;  
 } else {  
 left = mid + 1;  
 }  
 }  
 return left;  
}

## 5. Best Meeting Point Problem

#include <iostream>  
#include <vector>  
#include <algorithm>  
using namespace std;  
  
int minTotalDistance(vector<vector<int>>& grid) {  
 vector<int> rows, cols;  
   
 // Collect the coordinates of people in the grid  
 for (int i = 0; i < grid.size(); i++) {  
 for (int j = 0; j < grid[0].size(); j++) {  
 if (grid[i][j] == 1) rows.push\_back(i);  
 }  
 }  
   
 for (int j = 0; j < grid[0].size(); j++) {  
 for (int i = 0; i < grid.size(); i++) {  
 if (grid[i][j] == 1) cols.push\_back(j);  
 }  
 }  
   
 // Find the median position to minimize total distance  
 int rowMedian = rows[rows.size() / 2];  
 int colMedian = cols[cols.size() / 2];  
   
 int totalDistance = 0;  
 for (int r : rows) totalDistance += abs(r - rowMedian);  
 for (int c : cols) totalDistance += abs(c - colMedian);  
   
 return totalDistance;  
}