Walchand College of Engineering,Sangli

Computer Science & Engineering Third Year

Course: Design and analysis of algorithm Lab

Lab course coordinator:

Mrs A M Chimanna- Batch: - T1, T2, T3,T4

# Assignment 2

# Sorting Algorithm

# Sakshi Dhamapurkar

# 21510073

# Batch:T3

Q) Given an array *A[0…n-1]* of *n* numbers containing repetition of some number. Given an algorithm for checking whether there are repeated element or not. Assume that we are not allowed to use additional space (i.e., we can use a few temporary variable, *O(1)* storage).

*#include* <iostream>

bool hasDuplicate(int *A*[], int *n*) {

    int slow = *A*[0];

    int fast = *A*[0];

*do* {

        slow = *A*[slow];

        fast = *A*[*A*[fast]];

    } *while* (slow != fast);

*// Reset one pointers to the beginning*

    slow = *A*[0];

*while* (slow != fast) {

        slow = *A*[slow];

        fast = *A*[fast];

    }

*return* slow == fast;

}

int main() {

    int A[] = {2, 3, 1, 4, 2};

    int n = sizeof(A) / sizeof(A[0]);

*if* (hasDuplicate(A, n)) {

        std::cout << "Duplicates exist in the array." << std::endl;

    } *else* {

        std::cout << "No duplicates found in the array." << std::endl;

    }

*return* 0;

}

TC:O(N)

SC:O(1)

Q) Given an array *A[0…n-1]* , where each element of the array represents a vote in the election. Assume that each vote is given as an integer representing the ID of the chosen candidate. Given an algorithm for determining who wins the election.

*// Given an array A[0…n-1] , where each element of the array represents a vote in the election. Assume that each vote is given as an integer representing the ID of the chosen candidate. Given an algorithm for determining who wins the election*

*#include* <bits/stdc++.h>

using namespace std;

int findElectionWinner(vector<int> *votes*)

{

    unordered\_map<int, int> voteCounts;

*for* (int i = 0; i < *votes*.size(); i++)

    {

        int candidate = *votes*[i];

*if* (voteCounts.find(candidate) != voteCounts.end())

        {

            voteCounts[candidate]++;

        }

*else*

        {

            voteCounts[candidate] = 1;

        }

    }

    int winner = -1;

    int maxi = -1;

*// Iterate through the voteCounts map to find the candidate with the most votes.*

*for* (auto it = voteCounts.begin(); it != voteCounts.end(); ++it)

    {

*if* (it->second > maxi)

        {

            maxi = it->second;

            winner = it->first;

        }

    }

*return* winner;

}

int main()

{

    vector<int> votes = {1, 2, 3, 2, 1, 3, 1, 3, 3};

    int winner = findElectionWinner(votes);

*if* (winner != -1)

    {

        cout << "The winner is Candidate #" << winner << endl;

    }

*else*

    {

        cout << "No winner found." << endl;

    }

*return* 0;

}

TC: O(n + k), where 'n' is the number of votes, and 'k' is the number of distinct candidates. In most cases, 'k' is much smaller than 'n', so you can consider this algorithm to have a time complexity of approximately O(n).

SC:O(k)

Q) Given an array *A* of *n* elements, each of which is an integer in the range *[1, n2]*. How do we sort the array in *O(n)* time?

*// Given an array A of n elements, each of which is an integer in the range [1, n2]. How do we sort the array in O(n) time?*

*#include*<bits/stdc++.h>

using namespace std;

void countingSort(vector<int>& *A*, int *n*) {

    int maxElement = *n* \* *n*;

    vector<int> count(maxElement + 1, 0);

*for* (int i = 0; i < *n*; ++i) {

        count[*A*[i]]++;

    }

*// Reconstruct the sorted array*

    int index = 0;

*for* (int i = 1; i <= maxElement; ++i) {

*while* (count[i] > 0) {

*A*[index++] = i;

            count[i]--;

        }

    }

}

int main() {

    vector<int> A = {4, 2, 3, 1, 8, 6, 5, 7};

    int n = A.size();

    countingSort(A, n);

    cout << "Sorted Array: ";

*for* (int i = 0; i < n; ++i) {

        cout << A[i] << " ";

    }

    cout << endl;

*return* 0;

}

TC:O(n)

SC: Array of size of range [1,n^2]

Q) Let *A* and *B* tow arrays of n elements, each. Given a number *K*, give an *O (nlogn*) time algorithm for determining whether there exists *a ϵ A* and *b ϵ B* such that *a+b =K*.

Ans:

*//Let A and B tow arrays of n elements, each. Given a number K, give an O (nlogn) time algorithm for determining whether there exists a ϵ A and b ϵ B such that a+b =K.*

*#include*<bits/stdc++.h>

*//sakshi codes here*

using namespace std;

bool hasPairWithSum(vector<int>& *A*, vector<int>& *B*, int *K*) {

    sort(*B*.begin(), *B*.end());*//ascending order*

*for* (int a : *A*) {

        int target = *K* - a;

        auto it = lower\_bound(*B*.begin(), *B*.end(), target);

*if* (it != *B*.end() && \*it == target) {

*return* true;

        }

    }

*return* false;*//no pair found*

}

int main() {

    vector<int> A = {1, 2, 3, 4, 5};

    vector<int> B = {6, 7, 8, 9, 10};

    int K = 12;

*if* (hasPairWithSum(A, B, K)) {

        cout << "There exists a pair (a, b) such that a + b = " << K << endl;

    } *else* {

        cout << "No such pair found." << endl;

    }

*return* 0;

}

TC: O(n\*logn)

SC: O(1) and O(1) if sorting is done in-place