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

Week 2 Assignment

PRN : 21510111
Batch : T1

Sorting Algorithm

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>
using namespace std;

int findduplicate(int a[], int n){

    int i = 0, j = 0;
    while (i < n) {
        if (a[i] == a[++j])
            return a[j];
        if (j == n - 1) {
            i++;
            j = i;
        }
    }
    return -1;
}

int main(){

    int arr[] = { 1, 2, 4, 3, 4, 5, 6, 3 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findduplicate(arr, n) << endl;
    return 0;
}
```

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.

```
#include <bits/stdc++.h>
using namespace std;

void findWinner(vector<int>& votes){
    unordered_map<int, int> mapObj;
    for (auto& str : votes) {
        mapObj[str]++;
```

```
    }

    int maxValueInMap = 0;
    int winner;
    for (auto& entry : mapObj) {
        int key = entry.first;
        int val = entry.second;
        if (val > maxValueInMap) {
            maxValueInMap = val;
            winner = key;
        }
        else if (val == maxValueInMap && winner > key)
            winner = key;
    }
    cout << winner << endl;
}


int main()
{
    vector<int> votes = { 23,44,34,23,23,44,23,34 };

    findWinner(votes);
    return 0;
}
```

Q) Given an array A of n elements, each of which is an integer in the range $[1, n^2]$. How do we sort the array in $O(n)$ time?

```
#include<iostream>
using namespace std;

int countSort(int arr[], int n, int exp){

    int output[n];
    int i, count[n] ;
    for (int i=0; i < n; i++)
    count[i] = 0;

    for (i = 0; i < n; i++)
        count[ (arr[i]/exp)%n ]++;

    for (i = 1; i < n; i++)
        count[i] += count[i - 1];

    for (i = n - 1; i >= 0; i--)
    {
        output[count[ (arr[i]/exp)%n] - 1] = arr[i];
        count[(arr[i]/exp)%n]--;
    }

    for (i = 0; i < n; i++)
        arr[i] = output[i];
}

void sort(int arr[], int n){
    countSort(arr, n, 1);
    countSort(arr, n, n);
}

void printArr(int arr[], int n){
```

```
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}

int main(){
    int arr[] = {40, 12, 45, 32, 33, 1, 22};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Given array is\n";
    printArr(arr, n);

    sort(arr, n);

    cout << "\nSorted array is\n";
    printArr(arr, n);
    return 0;
}
```

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 $a \epsilon A$ and $b \epsilon B$ such that $a+b = K$.

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

using namespace std;


void findPairs(int arr1[], int arr2[], int n,int m, int x){

    unordered_set<int> s;

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

        s.insert(arr1[i]);


    for (int j = 0; j < m; j++)

        if (s.find(x - arr2[j]) != s.end())

            cout << x - arr2[j] << " "

                 << arr2[j] << endl;

}


int main(){

    int arr1[] = { 1, 0, -4, 7, 6, 4 };

    int arr2[] = { 0, 2, 4, -3, 2, 1 };

    int x = 8;

    int n = sizeof(arr1) / sizeof(int);

    int m = sizeof(arr2) / sizeof(int);

    findPairs(arr1, arr2, n, m, x);

    return 0;

}
```