**Assignment Questions**

**(Inversion Count and Selection Algo)**

Q1. Given an integer array, find the kth largest element using the quickselect algorithm.

Input1:

arr[] = {1,3,2,4,5,6,7} , k = 3

Output 1:

output 1 : 5

Input 2:

arr[] = {4,3,3,2,1} , k = 4

Output 2:

2

public class QS {

private static int pt(int[] ar, int l, int h) {

int pv = ar[h];

int i = l;

for (int j = l; j < h; j++) {

if (ar[j] > pv) {

int t = ar[i];

ar[i] = ar[j];

ar[j] = t;

i++;

}

}

int t = ar[i];

ar[i] = ar[h];

ar[h] = t;

return i;

}

private static int qs(int[] ar, int l, int h, int k) {

if (l <= h) {

int pi = pt(ar, l, h);

if (pi == k) {

return ar[pi];

} else if (pi > k) {

return qs(ar, l, pi - 1, k);

} else {

return qs(ar, pi + 1, h, k);

}

}

return Integer.MAX\_VALUE;

}

public static int fl(int[] ar, int k) {

return qs(ar, 0, ar.length - 1, k - 1);

}

public static void main(String[] args) {

int[] a1 = {1, 3, 2, 4, 5, 6, 7};

int k1 = 3;

System.out.println("Output 1: " + fl(a1, k1));

int[] a2 = {4, 3, 3, 2, 1};

int k2 = 4;

System.out.println("Output 2: " + fl(a2, k2));

}

}

What is the time complexity of the Quick Select algorithm?

a) O(n)

b) O(nlogn)

c) O(n^2)

d) O(logn)

Which data structure is used in Quick Select algorithm?

a) Linked List

b) Array

c) Binary Tree

d) Stack

Which partitioning scheme is used in Quick Select algorithm?

a) Lomuto partition scheme

b) Hoare partition scheme

c) Merge sort partition scheme

d) Heap sort partition scheme

Note but over Wikipedia: Hoare's scheme is more efficient than Lomuto's partition scheme because it does three times fewer swaps on average. Also, as mentioned, the implementation given creates a balanced partition even when all values are equal., which Lomuto's scheme does not.

What is the worst-case time complexity of Quick Select algorithm?

a) O(n)

**b) O(n^2)**

c) O(nlogn)

d) O(logn)

In the Quick Select algorithm, which element is chosen as the pivot element?

a) The first element

b) The middle element

c) The last element

d) A random element

**Assignment Questions**

**(Divide and conquer)**

Q1. Given an array where all its elements are sorted in increasing order except two swapped

elements, sort it in linear time. Assume there are no duplicates in the array.

arr[] = [3, 8, 6, 7, 5, 9, 10]

arr[] = [3, 5, 6, 7, 8, 9, 10]

public class FSArray {

public static void fixArray(int[] ar) {

int n = ar.length;

int f = -1, s = -1;

for (int i = 0; i < n - 1; i++) {

if (ar[i] > ar[i + 1]) {

if (f == -1) {

f = i;

} else {

s = i + 1;

break;

}

}

}

if (s == -1) {

s = f + 1;

}

int t = ar[f];

ar[f] = ar[s];

ar[s] = t;

}

public static void main(String[] args) {

int[] ar = {3, 8, 6, 7, 5, 9, 10};

fixArray(ar);

for (int num : ar) {

System.out.print(num + " ");

}

}

}

Q2. Given an array of positive and negative integers, segregate them in linear time and constant

space. The output should print all negative numbers, followed by all positive numbers.

arr[] = {19, -20, 7, -4, -13, 11, -5, 3}

arr[] = {-20 ,-4, -13, -5, 19 ,11 ,3, 7}

public class SegregateArray {

public static void segregate(int[] ar) {

int l = 0, r = ar.length - 1;

while (l <= r) {

if (ar[l] < 0) {

l++;

} else if (ar[r] > 0) {

r--;

} else {

int t = ar[l];

ar[l] = ar[r];

ar[r] = t;

l++;

r--;

}

}

}

public static void main(String[] args) {

int[] ar = {19, -20, 7, -4, -13, 11, -5, 3};

segregate(ar);

for (int num : ar) {

System.out.print(num + " ");

}

}

}

Q3. Given an array of positive and negative integers, segregate them in linear time and constant

space. The output should print all negative numbers, followed by all positive numbers. The

relative order of elements must remain the same.

arr[] = {19, -20, 7, -4, -13, 11, -5, 3}

arr[] = {-20 ,-4, -13, -5, 19 ,7 ,11, 3}

public class StableSegregateArray {

public static void segregate(int[] ar) {

int n = ar.length;

int j = 0;

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

if (ar[i] < 0) {

int t = ar[i];

for (int k = i; k > j; k--) {

ar[k] = ar[k - 1];

}

ar[j] = t;

j++;

}

}

}

public static void main(String[] args) {

int[] ar = {19, -20, 7, -4, -13, 11, -5, 3};

segregate(ar);

for (int num : ar) {

System.out.print(num + " ");

}

}

}

Q4. Given two arrays of equal size n and an integer k. The task is to permute both arrays such

that the sum of their corresponding element is greater than or equal to k i.e a[i] + b[i] >= k. The

task is to print “Yes” if any such permutation exists, otherwise print “No”.

a[] = {2, 1, 3},

b[] = { 7, 8, 9 },

k = 10.

Yes

a[] = {1, 2, 2, 1},

b[] = { 3, 3, 3, 4 },

k = 5.

No

import java.util.Arrays;

public class PermuteArrays {

public static void checkPermutation(int[] a, int[] b, int k) {

int n = a.length;

Arrays.sort(a);

Arrays.sort(b);

for (int i = 0; i < n / 2; i++) {

int t = b[i];

b[i] = b[n - 1 - i];

b[n - 1 - i] = t;

}

boolean isValid = true;

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

if (a[i] + b[i] < k) {

isValid = false;

break;

}

}

System.out.println(isValid ? "Yes" : "No");

}

public static void main(String[] args) {

int[] a1 = {2, 1, 3};

int[] b1 = {7, 8, 9};

int k1 = 10;

checkPermutation(a1, b1, k1);

int[] a2 = {1, 2, 2, 1};

int[] b2 = {3, 3, 3, 4};

int k2 = 5;

checkPermutation(a2, b2, k2);

}

}

Q5. An interval is represented as a combination of start time and end time. Given a set of

intervals, check if any two intervals intersect.

arr[] = {{1, 3}, {5, 7}, {2, 4}, {6, 8}}

Yes

The intervals {1, 3} and {2, 4} overlap

arr[] = {{1, 3}, {7, 9}, {4, 6}, {10, 13}}

No

import java.util.Arrays;

import java.util.Comparator;

public class IntervalCheck {

public static boolean check(int[][] invs) {

Arrays.sort(invs, Comparator.comparingInt(a -> a[0]));

for (int i = 0; i < invs.length - 1; i++) {

if (invs[i][1] >= invs[i + 1][0]) {

return true;

}

}

return false;

}

public static void main(String[] args) {

int[][] a1 = {{1, 3}, {5, 7}, {2, 4}, {6, 8}};

int[][] a2 = {{1, 3}, {7, 9}, {4, 6}, {10, 13}};

System.out.println(check(a1) ? "Yes" : "No");

System.out.println(check(a2) ? "Yes" : "No");

}

}