Find k pairs with smallest sums in two arrays

Difficulty Level : Medium • Last Updated : 24 May, 2022

Given two integer arrays arr1[] and arr2[] sorted in ascending order and an integer k. Find k pairs with smallest sums such that one element of a pair belongs to arr1[] and other element belongs to arr2[]
Examples:

```
Input :  arr1[] = {1, 7, 11}
         arr2[] = {2, 4, 6}
         k = 3
Output : [1, 2],
         [1, 4],
         [1, 6]
Explanation: The first 3 pairs are returned
from the sequence [1, 2], [1, 4], [1, 6],
[7, 2], [7, 4], [11, 2], [7, 6], [11, 4],
[11, 6]
```

Recommended: Please try your approach on {IDE} first, before

1. Find all pairs and store their sums. Time complexity of this step is O(n1 * n2) where n1 and n2 are sizes of input arrays.
2. Then sort pairs according to sum. Time complexity of this step is O(n1 * n2 * log (n1 * n2))

Overall Time Complexity : O(n1 * n2 * log (n1 * n2))

Auxiliary Space : O(n1*n2)

Method 2 (Efficient):

We one by one find k smallest sum pairs, starting from least sum pair. The idea is to keep track of all elements of arr2[] which have been already considered for every element arr1[i1] so that in an iteration we only consider next element. For this purpose, we use an index array index2[] to track the indexes of next elements in the other array. It simply means that which element of second array to be added with the element of first array in each and every iteration. We increment value in index array for the element that forms next minimum value pair.

C++

```
// C++ program to prints first k pairs with least sum from two
// arrays.
#include<bits/stdc++.h>

using namespace std;

// Function to find k pairs with least sum such
// that one element of a pair is from arr1[] and
// other element is from arr2[]
void kSmallestPair(int arr1[], int n1, int arr2[],
                                    int n2, int k)
{
    if (k > n1*n2)
    {
```

```
// Stores current index in arr2[] for
// every element of arr1[]. Initially
// all values are considered 0.
// Here current index is the index before
// which all elements are considered as
// part of output.
int index2[n1];
memset(index2, 0, sizeof(index2));

while (k > 0)
{
    // Initialize current pair sum as infinite
    int min_sum = INT_MAX;
    int min_index = 0;

    // To pick next pair, traverse for all elements
    // of arr1[], for every element, find corresponding
    // current element in arr2[] and pick minimum of
    // all formed pairs.
    for (int i1 = 0; i1 < n1; i1++)
    {
        // Check if current element of arr1[] plus
        // element of array2 to be used gives minimum
        // sum
        if (index2[i1] < n2 &&
            arr1[i1] + arr2[index2[i1]] < min_sum)
        {
            // Update index that gives minimum
            min_index = i1;

            // update minimum sum
            min_sum = arr1[i1] + arr2[index2[i1]];
        }
    }

    cout << "(" << arr1[min_index] << ", "
         << arr2[index2[min_index]] << ") ";

    index2[min_index]++;

    k--;
```

```
int main()
{
    int arr1[] = {1, 3, 11};
    int n1 = sizeof(arr1) / sizeof(arr1[0]);

    int arr2[] = {2, 4, 8};
    int n2 = sizeof(arr2) / sizeof(arr2[0]);

    int k = 4;
    kSmallestPair( arr1, n1, arr2, n2, k);

    return 0;
}
```

Java

```
// Java code to print first k pairs with least
// sum from two arrays.
import java.io.*;

class KSmallestPair
{
    // Function to find k pairs with least sum such
    // that one element of a pair is from arr1[] and
    // other element is from arr2[]
    static void kSmallestPair(int arr1[], int n1, int arr2[],
                                          int n2, int k)
    {
        if (k > n1*n2)
        {
            System.out.print("k pairs don't exist");
            return ;
        }

        // Stores current index in arr2[] for
        // every element of arr1[]. Initially
        // all values are considered 0.
        // Here current index is the index before
        // which all elements are considered as
        // part of output
```

```
            // Initialize current pair sum as infinite
            int min_sum = Integer.MAX_VALUE;
            int min_index = 0;

            // To pick next pair, traverse for all
            // elements of arr1[], for every element, find
            // corresponding current element in arr2[] and
            // pick minimum of all formed pairs.
            for (int i1 = 0; i1 < n1; i1++)
            {
                // Check if current element of arr1[] plus
                // element of array2 to be used gives
                // minimum sum
                if (index2[i1] < n2 &&
                    arr1[i1] + arr2[index2[i1]] < min_sum)
                {
                    // Update index that gives minimum
                    min_index = i1;

                    // update minimum sum
                    min_sum = arr1[i1] + arr2[index2[i1]];
                }
            }

            System.out.print("(" + arr1[min_index] + ", " +
                           arr2[index2[min_index]]+ ") ");

            index2[min_index]++;
            k--;
        }
    }

    // Driver code
    public static void main (String[] args)
    {
        int arr1[] = {1, 3, 11};
        int n1 = arr1.length;

        int arr2[] = {2, 4, 8};
        int n2 = arr2.length;
```

```
/*This code is contributed by Prakriti Gupta*/
```

Python3

```
# Python3 program to prints first k pairs with least sum from two
# arrays.

import sys
# Function to find k pairs with least sum such
# that one element of a pair is from arr1[] and
# other element is from arr2[]
def kSmallestPair(arr1, n1, arr2, n2, k):
    if (k > n1*n2):
        print("k pairs don't exist")
        return

    # Stores current index in arr2[] for
    # every element of arr1[]. Initially
    # all values are considered 0.
    # Here current index is the index before
    # which all elements are considered as
    # part of output.
    index2 = [0 for i in range(n1)]

    while (k > 0):
        # Initialize current pair sum as infinite
        min_sum = sys.maxsize
        min_index = 0

        # To pick next pair, traverse for all elements
        # of arr1[], for every element, find corresponding
        # current element in arr2[] and pick minimum of
        # all formed pairs.
        for i1 in range(0,n1,1):
            # Check if current element of arr1[] plus
            # element of array2 to be used gives minimum
            # sum
            if (index2[i1] < n2 and arr1[i1] + arr2[index2[i1]] < min_sur
                # Update index that gives minimum
                min_index = i1
```

```
            print("(",arr1[min_index],",",arr2[index2[min_index]],")",end = "
            index2[min_index] += 1
            k -= 1
# Driver code
if __name__ == '__main__':
    arr1 = [1, 3, 11]
    n1 = len(arr1)
    arr2 = [2, 4, 8]
    n2 = len(arr2)
    k = 4
    kSmallestPair( arr1, n1, arr2, n2, k)
# This code is contributed by
# Shashank_Sharma
```

C#

```
// C# code to print first k pairs with
// least with least sum from two arrays.
using System;
class KSmallestPair
{
    // Function to find k pairs with least
    // sum such that one element of a pair
    // is from arr1[] and other element is
    // from arr2[]
    static void kSmallestPair(int []arr1, int n1,
                        int []arr2, int n2, int k)
    {
        if (k > n1 * n2)
        {
            Console.Write("k pairs don't exist");
            return;
```

```
        // current index is the index before
        // which all elements are considered
        // as part of output.
        int []index2 = new int[n1];

        while (k > 0)
        {
            // Initialize current pair sum as infinite
            int min_sum = int.MaxValue;
            int min_index = 0;

            // To pick next pair, traverse for all
            // elements of arr1[], for every element,
            // find corresponding current element in
            // arr2[] and pick minimum of all formed pairs.
            for (int i1 = 0; i1 < n1; i1++)
            {
                // Check if current element of arr1[]
                // plus element of array2 to be used
                // gives minimum sum
                if (index2[i1] < n2 && arr1[i1] +
                    arr2[index2[i1]] < min_sum)
                {
                    // Update index that gives minimum
                    min_index = i1;

                    // update minimum sum
                    min_sum = arr1[i1] + arr2[index2[i1]];
                }
            }

        Console.Write("(" + arr1[min_index] + ", " +
                        arr2[index2[min_index]] + ") ");

            index2[min_index]++;
            k--;
        }
    }

    // Driver code
    public static void Main (String[] args)
```

```
        int []arr2 = {2, 4, 8};
        int n2 = arr2.Length;

        int k = 4;
        kSmallestPair( arr1, n1, arr2, n2, k);
    }
}

// This code is contributed by Parashar.
```

Javascript

```
<script>
// javascript program to prints first k pairs with least sum from two
// arrays.
// Function to find k pairs with least sum such
// that one element of a pair is from arr1[] and
// other element is from arr2[]
function kSmallestPair(arr1,n1,arr2,n2,k)
{
    if (k > n1*n2)
    {
        document.write("k pairs don't exist");
        return ;
    }

    // Stores current index in arr2[] for
    // every element of arr1[]. Initially
    // all values are considered 0.
    // Here current index is the index before
    // which all elements are considered as
    // part of output.
    let index2 = new Array(n1);
    index2.fill(0);

    while (k > 0)
    {
        // Initialize current pair sum as infinite
        let min_sum = Number.MAX_VALUE;
        let min_index = 0;
```

```
        // all formed pairs.
        for (let i1 = 0; i1 < n1; i1++)
        {
            // Check if current element of arr1[] plus
            // element of array2 to be used gives minimum
            // sum
            if (index2[i1] < n2 &&
                arr1[i1] + arr2[index2[i1]] < min_sum)
            {
                // Update index that gives minimum
                min_index = i1;

                // update minimum sum
                min_sum = arr1[i1] + arr2[index2[i1]];
            }
        }

        document.write("(" + arr1[min_index] + ", "
            + arr2[index2[min_index]] + ") ");

        index2[min_index]++;

        k--;
    }
}

    let arr1 = [1, 3, 11];
    let n1 = arr1.length;

    let arr2 = [2, 4, 8];
    let n2 = arr2.length;

    let k = 4;
    kSmallestPair( arr1, n1, arr2, n2, k);



</script>
```

Time Complexity : O(k*n1)

Auxiliary Space : O(n1)

Method 3 : Using Sorting, Min heap, Map
Instead of brute forcing through all the possible sum combinations we should find a way to limit our search space to possible candidate sum combinations.

1. Sort both arrays array A and array B.
2. Create a min heap i.e priority_queue in C++ to store the sum combinations along with the indices of elements from both arrays A and B which make up the sum. Heap is ordered by the sum.
3. Initialize the heap with the minimum possible sum combination i.e (A[0] + B[0]) and with the indices of elements from both arrays (0, 0). The tuple inside min heap will be (A[0] + B[0], 0, 0). Heap is ordered by first value i.e sum of both elements.
4. Pop the heap to get the current smallest sum and along with the indices of the element that make up the sum. Let the tuple be (sum, i, j).
   - Next insert (A[i + 1] + B[j], i + 1, j) and (A[i] + B[j + 1], i, j + 1) into the min heap but make sure that the pair of indices i.e (i + 1, j) and (i, j + 1) are not already present in the min heap.To check this we can use set in C++.
   - Go back to 4 until K times.

C++

```
// C++ program to Prints
// first k pairs with
// least sum from two arrays.
```

```
// with least sum such
// that one element of a pair
// is from arr1[] and
// other element is from arr2[]
void kSmallestPair(vector<int> A, vector<int> B, int K)
{
    sort(A.begin(), A.end());
    sort(B.begin(), B.end());

    int n = A.size();

    // Min heap which contains tuple of the format
    // (sum, (i, j)) i and j are the indices
    // of the elements from array A
    // and array B which make up the sum.

    priority_queue<pair<int, pair<int, int> >,
                   vector<pair<int, pair<int, int> > >,
                   greater<pair<int, pair<int, int> > > >
        pq;

    // my_set is used to store the indices of
    // the  pair(i, j) we use my_set to make sure
    // the indices does not repeat inside min heap.

    set<pair<int, int> > my_set;

    // initialize the heap with the minimum sum
    // combination i.e. (A[0] + B[0])
    // and also push indices (0,0) along
    // with sum.

    pq.push(make_pair(A[0] + B[0], make_pair(0, 0)));

    my_set.insert(make_pair(0, 0));

    // iterate upto K
    int flag=1;
    for (int count = 0; count < K && flag; count++) {

        // tuple format (sum, i, j).
```

```
int j = temp.second.second;

cout << "(" << A[i] << ", " << B[j] << ")"
     << endl; // Extracting pair with least sum such
              // that one element is from arr1 and
              // another is from arr2

// check if i+1 is in the range of our first array A
flag=0;
if (i + 1 < A.size()) {
    int sum = A[i + 1] + B[j];
    // insert (A[i + 1] + B[j], (i + 1, j))
    // into min heap.
    pair<int, int> temp1 = make_pair(i + 1, j);

    // insert only if the pair (i + 1, j) is
    // not already present inside the map i.e.
    // no repeating pair should be present inside
    // the heap.

    if (my_set.find(temp1) == my_set.end()) {
        pq.push(make_pair(sum, temp1));
        my_set.insert(temp1);
    }
  flag=1;
}
// check if j+1 is in the range of our second array
// B
if (j + 1 < B.size()) {
    // insert (A[i] + B[j + 1], (i, j + 1))
    // into min heap.

    int sum = A[i] + B[j + 1];
    pair<int, int> temp1 = make_pair(i, j + 1);

    // insert only if the pair (i, j + 1)
    // is not present inside the heap.

    if (my_set.find(temp1) == my_set.end()) {
        pq.push(make_pair(sum, temp1));
        my_set.insert(temp1);
```

```
}

// Driver Code.
int main()
{
    vector<int> A = { 1 };
    vector<int> B = { 2, 4, 5, 9 };
    int K = 3;
    kSmallestPair(A, B, K);
    return 0;
}

// This code is contributed by Dhairva.
```

Output

```
(1, 2)
(1, 4)
(1, 5)
```

Time Complexity : O(n*logn) assuming k<=n

Auxiliary Space: O(n) as we are using extra space

Reference :

http://sudhansu-codezone.blogspot.in/2012/02/triplets-in-array-with-sum-0.html

This article is contributed by **Sahil Chhabra** . If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Like 33

Previous

k smallest elements in same order using O(1) extra space

Next

k-th smallest absolute difference of two elements in an array

RECOMMENDED ARTICLES

01 Javascript Program for Find k pairs with smallest sums in two arrays
21, Dec 21

02 Java Program to Find k pairs with smallest sums in two arrays
21, Dec 21

03 Python3 Program for Find k pairs with smallest sums in two arrays

05 Count of pairs between two arrays such that the sums are distinct
08, Jul 19

06 Find the sums for which an array can be divided into sub-arrays of equal sum
31, Dec 18

07 Find N - 1 pairs from given array such that GCD of all pair-

arrays
21, Oct 21

already exist in array
26, Jan 17

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : parashar, Dhairya Shah, Shashank_Sharma, shethnilay12, impwork8053, vaibhavrabadiya117, ss100429, sagartomar9927, anandkumarshivam2266, simmytarika5

Article Tags : Order-Statistics, Arrays

Practice Tags : Arrays

Improve Article | Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

NodeJS