Find Pair with given Sum in the Array

Given an unsorted array of integers, find a pair with given sum in it.

For example,

Input:

```
arr = [8, 7, 2, 5, 3, 1]
sum = 10
```

Output:

```
Pair found at index 0 and 2 (8 + 2)
OR
Pair found at index 1 and 4 (7 + 3)
```

1. Naive Approach –

Naive solution would be to consider every pair in given array and return if desired sum is found.

C Java

```c
#include <stdio.h>

// Naive method to find a pair in an array with given sum
void findPair(int arr[], int n, int sum)
{
    // consider each element except last element
    for (int i = 0; i < n - 1; i++)
    {
        // start from i'th element till last element
        for (int j = i + 1; j < n; j++)
        {
            // if desired sum is found, print it and return
            if (arr[i] + arr[j] == sum)
            {
                printf("Pair found at index %d and %d", i, j);
                return;
            }
        }
    }

    // No pair with given sum exists in the array
    printf("Pair not found");
}

// Find pair with given sum in the array
int main()
{
    int arr[] = { 8, 7, 2, 5, 3, 1 };
    int sum = 10;

    int n = sizeof(arr)/sizeof(arr[0]);

    findPair(arr, n, sum);

    return 0;
}
```

Download Run Code

Output:

```
Pair found at index 0 and 2
```

The time complexity of above solution is $O(n^2)$ and auxiliary space used by the program is $O(1)$.

2. O(nlogn) solution using Sorting –

The idea is to sort the given array in ascending order and maintain search space by maintaining two indices (low and high) that initially points to two end-points of the array. Then we loop till low is less than high index and reduce search space `arr[low..high]` at each iteration of the loop. We compare sum of elements present at index low and high with desired sum and increment low if sum is less than the desired sum else we decrement high is sum is more than the desired sum. Finally, we return if pair found in the array.

C++ Java

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

// Function to find a pair in an array with given sum using Sorting
void findPair(int arr[], int n, int sum)
{
    // sort the array in ascending order
    std::sort(arr, arr + n);

    // maintain two indices pointing to end-points of the array
    int low = 0;
    int high = n - 1;

    // reduce search space arr[low..high] at each iteration of the loop

    // loop till low is less than high
    while (low < high)
    {
        // sum found
        if (arr[low] + arr[high] == sum)
        {
            std::cout << "Pair found";
            return;
        }

        // increment low index if total is less than the desired sum
        // decrement high index is total is more than the sum
        (arr[low] + arr[high] < sum)? low++: high--;
    }

    // No pair with given sum exists in the array
    std::cout << "Pair not found";
}

// Find pair with given sum in the array
int main()
{
    int arr[] = { 8, 7, 2, 5, 3, 1};
    int sum = 10;

    int n = sizeof(arr)/sizeof(arr[0]);

    findPair(arr, n, sum);

    return 0;
}
```

Download Run Code

Output:

```
Pair found
```

The time complexity of above solution is $O(nlogn)$ and auxiliary space used by the program is $O(1)$.

3. O(n) solution using Hashing –

We can use map to easily solve this problem in linear time. The idea is to insert each element of the array `arr[i]` in a map. We also checks if difference `(arr[i], sum-arr[i])` already exists in the map or not. If the difference is seen before, we print the pair and return.

C++ Java

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

// Function to find a pair in an array with given sum using Hashing
void findPair(int arr[], int n, int sum)
{
    // create an empty map
    unordered_map<int, int> map;

    // do for each element
    for (int i = 0; i < n; i++)
    {
        // check if pair (arr[i], sum-arr[i]) exists

        // if difference is seen before, print the pair
        if (map.find(sum - arr[i]) != map.end())
        {
            cout << "Pair found at index " << map[sum - arr[i]] <<
                    " and " << i;
            return;
        }

        // store index of current element in the map
        map[arr[i]] = i;
    }

    // we reach here if pair is not found
    cout << "Pair not found";
}

// Find pair with given sum in the array
int main()
{
    int arr[] = { 8, 7, 2, 5, 3, 1};
    int sum = 10;

    int n = sizeof(arr)/sizeof(arr[0]);

    findPair(arr, n, sum);

    return 0;
}
```

Download Run Code

Output:

```
Pair found at index 0 and 2
```

The time complexity of above solution is $O(n)$ and auxiliary space used by the program is $O(n)$.