**Data Structure Interview Question- Array**

1. **Program for array rotation?**

Ans-

**METHOD 1 (Using temp array)**

Input arr[] = [1, 2, 3, 4, 5, 6, 7], d = 2, n =7

1) Store d elements in a temp array

temp[] = [1, 2]

2) Shift rest of the arr[]

arr[] = [3, 4, 5, 6, 7, 6, 7]

3) Store back the d elements

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

**Time complexity :** O(n)  
**Auxiliary Space :**O(d)

**METHOD 2 (Rotate one by one)**

leftRotate(arr[], d, n)

start

For i = 0 to i < d

Left rotate all elements of arr[] by one

end

**Time complexity :** O(n \* d)  
**Auxiliary Space :** O(1)

  /\*Function to left rotate arr[] of size n by d\*/

    void leftRotate(int arr[], int d, int n)

    {

        for (int i = 0; i < d; i++)

            leftRotatebyOne(arr, n);

    }

    void leftRotatebyOne(int arr[], int n)

    {

        int i, temp;

        temp = arr[0];

        for (i = 0; i < n - 1; i++)

            arr[i] = arr[i + 1];

        arr[i] = temp;

    }

**Method 3 (The Reversal Algorithm) :**

**Algorithm :**

rotate(arr[], d, n)

reverse(arr[], 1, d) ;

reverse(arr[], d + 1, n);

reverse(arr[], 1, n)

**Example :**  
Let the array be arr[] = [1, 2, 3, 4, 5, 6, 7], d =2 and n = 7  
A = [1, 2] and B = [3, 4, 5, 6, 7]

* Reverse A, we get ArB = [2, 1, 3, 4, 5, 6, 7]
* Reverse B, we get ArBr = [2, 1, 7, 6, 5, 4, 3]
* Reverse all, we get (ArBr)r = [3, 4, 5, 6, 7, 1, 2]

 /\* Function to left rotate arr[] of size n by d \*/

    static void leftRotate(int arr[], int d)

    {

        if (d == 0)

            return;

        int n = arr.length;

        rvereseArray(arr, 0, d - 1);

        rvereseArray(arr, d, n - 1);

        rvereseArray(arr, 0, n - 1);

    }

    /\*Function to reverse arr[] from index start to end\*/

    static void rvereseArray(int arr[], int start, int end)

    {

        int temp;

        while (start < end) {

            temp = arr[start];

            arr[start] = arr[end];

            arr[end] = temp;

            start++;

            end--;

        }

    }

1. **Search for an element in sorted and rotated array?**

Ans- The idea is to find the pivot point, divide the array in two sub-arrays and call binary searc

Input arr[] = {3, 4, 5, 1, 2}

Element to Search = 1

1) Find out pivot point and divide the array in two

sub-arrays. (pivot = 2) /\*Index of 5\*/

2) Now call binary search for one of the two sub-arrays.

(a) **If** element is greater than 0th element then

search in left array

(b) **Else** Search in right array

(1 will go in else as 1 < 0th element(3))

3) **If** element is found in selected sub-array then return index

**Else** return -1.

/\* Standard Binary Search function\*/

int binarySearch(int arr[], int low,

                  int high, int key)

{

  if (high < low)

    return -1;

  int mid = (low + high)/2; /\*low + (high - low)/2;\*/

  if (key == arr[mid])

    return mid;

  if (key > arr[mid])

    return binarySearch(arr, (mid + 1), high, key);

  // else

    return binarySearch(arr, low, (mid -1), key);

}

/\* Function to get pivot. For array 3, 4, 5, 6, 1, 2

   it returns 3 (index of 6) \*/

int findPivot(int arr[], int low, int high)

{

  // base cases

  if (high < low) return -1;

  if (high == low) return low;

   int mid = (low + high)/2; /\*low + (high - low)/2;\*/

   if (mid < high && arr[mid] > arr[mid + 1])

    return mid;

   if (mid > low && arr[mid] < arr[mid - 1])

    return (mid-1);

   if (arr[low] >= arr[mid])

    return findPivot(arr, low, mid-1);

   return findPivot(arr, mid + 1, high);

}

/\* Searches an element key in a pivoted

   sorted array arr[] of size n \*/

int pivotedBinarySearch(int arr[], int n, int key)

{

  int pivot = findPivot(arr, 0, n-1);

  // If we didn't find a pivot,

  // then array is not rotated at all

  if (pivot == -1)

    return binarySearch(arr, 0, n-1, key);

  // If we found a pivot, then first compare with pivot

  // and then search in two subarrays around pivot

  if (arr[pivot] == key)

    return pivot;

  if (arr[0] <= key)

    return binarySearch(arr, 0, pivot-1, key);

    return binarySearch(arr, pivot+1, n-1, key);

}

1. **Given an array A[] of n numbers and another number x, determines whether or not there exist two elements in S whose sum is exactly x.**

Ans-

**METHOD 1 (Use Sorting)**

**Algorithm :**

hasArrayTwoCandidates (A[], ar\_size, sum)

1) Sort the array in non-decreasing order.

2) Initialize two index variables to find the candidate

elements in the sorted array.

(a) Initialize first to the leftmost index: l = 0

(b) Initialize second the rightmost index: r = ar\_size-1

3) Loop while l < r.

(a) If (A[l] + A[r] == sum) then return 1

(b) Else if( A[l] + A[r] < sum ) then l++

(c) Else r--

4) No candidates in whole array - return 0

**Time Complexity:** Depends on what sorting algorithm we use. If we use Merge Sort or Heap Sort then (-)(nlogn) in worst case. If we use Quick Sort then O(n^2) in worst case.  
**Auxiliary Space :** Again, depends on sorting algorithm. For example auxiliary space is O(n) for merge sort and O(1) for Heap Sort.

**METHOD 2 (Use Hashing)**  
This method works in O(n) time.

1) Initialize an empty hash table s.

2) Do following for each element A[i] in A[]

(a) If s[x - A[i]] is set then print the pair (A[i], x - A[i])

(b) Insert A[i] into s.

static void printpairs(int arr[], int sum)

    {

        HashSet<Integer> s = new HashSet<Integer>();

        for (int i = 0; i < arr.length; ++i) {

            int temp = sum - arr[i];

            // checking for condition

            if (s.contains(temp)) {

                System.out.println("Pair with given sum " + sum + " is (" + arr[i] + ", " + temp + ")");

            }

            s.add(arr[i]);

        }

    }

**Time Complexity:**O(n)  
**Auxiliary Space:** O(n) where n is size of array.