Monday, December 27, 2021 2:07 AM

Sort by Set Bit Count

Easy Accuracy: 50.15% Submissions: 17336 Points: 2

Given an array of integers, sort the array (in descending order) according to count of set bits in binary representation of array elements.

Note: For integers having same number of set bits in their binary representation, sort according to their position in the original array i.e., a stable sort.

Example 1:
Input:
arr[] = {5, 2, 3, 9, 4, 6, 7, 15, 32};
Output:
15 7 5 3 9 6 2 4 32
Explanation:
The integers in their binary
representation are:
15 - 1111
7 - 0111
5 - 0101
3 - 0011
9 - 1001
6 - 0110
2 - 0010
4 - 0100
32 - 10000
hence the non-increasing sorted order is:
{15}, {7}, {5, 3, 9, 6}, {2, 4, 32}

Example 2:
Input:
arr[] = {1, 2, 3, 4, 5, 6};
Output:
3 5 6 1 2 4
Explanation:
3 - 0011
5 - 0101
6 - 0110
1 - 0001
2 - 0010
4 - 0100
hence the non-increasing sorted order is
{3, 5, 6}, {1, 2, 4}

Your Task:

You don't need to print anything, printing is done by the driver code itself. You just need to complete the function **sortBySetBitCount()** which takes the array **arr[]** and its size **N** as inputs and sort the array **arr[]** inplace. Use of extra space is prohibited.

Expected Time Complexity: O(N.log(N))

Expected Auxiliary Space: O(1)

From <https://practice.geeksforgeeks.org/problems/sort-by-set-bit-count1153/1>

Constraints:

$1 \le N \le 10^5$

$1 \le A[i] \le 10^6$

From <https://practice.geeksforgeeks.org/problems/sort-by-set-bit-count1153/1>

Solution

```
class Compute
{
    static void sortBySetBitCount(Integer arr[], int n)
    {
        // Your code goes here
            Arrays.sort(arr, (a, b) -> -Integer.compare
            (Integer.bitCount(a), Integer.bitCount(b)));



    }
}
```

Sort an array according to count of set bits

- Difficulty Level : Medium
- Last Updated : 28 Jun, 2021

Given an array of positive integers, sort the array in decreasing order of count of set bits in binary representations of array elements. For integers having the same number of set bits in their binary representation, sort according to their position in the original array i.e., a stable sort. For example, if the input array is {3, 5}, then the output array should also be {3, 5}. Note that both 3 and 5 have the same number set bits.

Examples:

```
Input: arr[] = {5, 2, 3, 9, 4, 6, 7, 15, 32};
Output: 15 7 5 3 9 6 2 4 32
Explanation:
The integers in their binary representation are:
    15 -1111
    7  -0111
    5  -0101
    3  -0011
    9  -1001
    6  -0110
    2  -0010
    4- -0100
    32 -10000
hence the non-increasing sorted order is:
{15}, {7}, {5, 3, 9, 6}, {2, 4, 32}
Input: arr[] = {1, 2, 3, 4, 5, 6};
Output: 3 5 6 1 2 4
Explanation:
    3  - 0011
    5  - 0101
    6  - 0110
    1  - 0001
    2  - 0010
    4  - 0100
```

hence the non-increasing sorted order is
{3, 5, 6}, {1, 2, 4}

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

Method 1: Simple

1. Create an auxiliary array and store the set-bit counts of all integers in the aux array
2. Simultaneously sort both arrays according to the non-increasing order of auxiliary array. (Note that we need to use a stable sort algorithm)

Before sort:
```
int arr[] = {1, 2, 3, 4, 5, 6};
int aux[] = {1, 1, 2, 1, 2, 2}
```
After sort:
```
arr = {3, 5, 6, 1, 2, 4}
aux = {2, 2, 2, 1, 1, 1}
```
Implementation:

- C++
- Java
- Python3
- C#
- Javascript

```
// Java program to implement
// simple approach to sort
// an array according to
// count of set bits.
import java.io.*;

class GFG
{

// a utility function that
// returns total set bits
// count in an integer
static int countBits(int a)
{
    int count = 0;
    while (a > 0)
    {
        if ((a & 1) > 0)
            count+= 1;
        a = a >> 1;
    }
    return count;
}

// Function to simultaneously
// sort both arrays using
// insertion sort
// ( https://www.geeksforgeeks.org/insertion-sort/ )
static void insertionSort(int arr[],
                          int aux[], int n)
{
    for (int i = 1; i < n; i++)
    {
        // use 2 keys because we
        // need to sort both
        // arrays simultaneously
        int key1 = aux[i];
        int key2 = arr[i];
        int j = i - 1;

        /* Move elements of arr[0..i-1]
        and aux[0..i-1], such that
        elements of aux[0..i-1] are
        greater than key1, to one
```

```
        position ahead of their current
        position */
        while (j >= 0 && aux[j] < key1)
        {
            aux[j + 1] = aux[j];
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        aux[j + 1] = key1;
        arr[j + 1] = key2;
    }
}

// Function to sort according
// to bit count using an
// auxiliary array
static void sortBySetBitCount(int arr[],
                              int n)
{
    // Create an array and
    // store count of
    // set bits in it.
    int aux[] = new int[n];
    for (int i = 0; i < n; i++)
        aux[i] = countBits(arr[i]);

    // Sort arr[] according
    // to values in aux[]
    insertionSort(arr, aux, n);
}

// Utility function
// to print an array
static void printArr(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        System.out.print(arr[i] + " ");
}

// Driver Code
public static void main (String[] args)
{
    int arr[] = {1, 2, 3, 4, 5, 6};
    int n = arr.length;
    sortBySetBitCount(arr, n);
    printArr(arr, n);
}
}

// This code is contributed by anuj_67.
```

Output:

```
3 5 6 1 2 4
```

Auxiliary Space: O(n)

Time complexity: O(n^2)

Note: Time complexity can be improved to O(nLogn) by using a stable O(nlogn) sorting algorithm.

Method 2: Using std::sort()

Using custom comparator of std::sort to sort the array according to set-bit count

- C++
- Java
- Python3
- Javascript

```
// Java program to sort an array according to
// count of set bits using std::sort()
import java.util.Arrays;
import java.util.Comparator;
```

```
public class Test {

    public static void main(String[] args)
    {
        // TODO Auto-generated method stub
        Integer arr[] = { 1, 2, 3, 4, 5, 6 };
        int n = 6;
        sortBySetBitCount(arr, n);
        printArr(arr, n);
        System.out.println();
    }

    private static void printArr(Integer[] arr, int n)
    {
        // TODO Auto-generated method stub
        for (int i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
        // cout << arr[i] << " ";
    }

    private static Integer[] sortBySetBitCount(
        Integer[] arr, int n)
    {
        // TODO Auto-generated method stub
        Arrays.sort(arr, new Comparator<Integer>() {
            @Override
            public int compare(Integer arg0, Integer arg1)
            {
                // TODO Auto-generated method stub
                int c1 = Integer.bitCount(arg0);
                int c2 = Integer.bitCount(arg1);
                if (c1 <= c2)
                    return 1;
                else
                    return -1;
            }
        });
        return arr;
    }
}

// This code is contributed by 4-Bit-R
```

Output:

```
3 5 6 1 2 4
```

Auxiliary Space: O(1)

Time complexity: O(n log n)

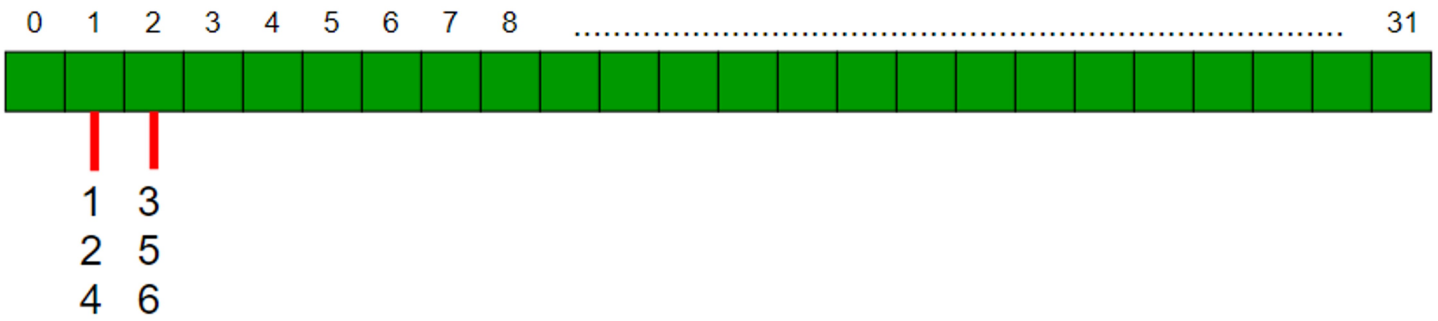
Method 3: Counting Sort based

This problem can be solved in O(n) time. The idea is similar to counting sort.

Note: There can be a minimum 1 set-bit and only a maximum of 31set-bits in an integer.

Steps (assuming that an integer takes 32 bits):

3. Create a vector “count” of size 32. Each cell of count i.e., count[i] is another vector that stores all the elements whose set-bit-count is i
4. Traverse the array and do the following for each element:
5. Count the number set-bits of this element. Let it be ‘setbitcount’
6. count[setbitcount].push_back(element)
7. Traverse ‘count’ in reverse fashion(as we need to sort in non-increasing order) and modify the array.



- C++
- Java
- Python3
- C#
- Javascript

```
// Java program to sort an
// array according to count
// of set bits using std::sort()
import java.util.*;
class GFG{

// a utility function that
// returns total set bits
// count in an integer
static int countBits(int a)
{
  int count = 0;
  while (a > 0)
  {
    if ((a & 1) > 0 )
      count += 1;
    a = a >> 1;
  }
  return count;
}

// Function to sort according to
// bit count. This function assumes
// that there are 32 bits in an integer.
static void sortBySetBitCount(int arr[],
                              int n)
{
  Vector<Integer> []count =
         new Vector[32];

  for (int i = 0;
           i < count.length; i++)
    count[i] = new Vector<Integer>();

  int setbitcount = 0;

  for (int i = 0; i < n; i++)
  {
    setbitcount = countBits(arr[i]);
    count[setbitcount].add(arr[i]);
  }

  // Used as an index in
  // final sorted array
  int j = 0;

  // Traverse through all bit
  // counts (Note that we sort
  // array in decreasing order)
  for (int i = 31; i >= 0; i--)
  {
    Vector<Integer> v1 = count[i];
```

```
    for (int p = 0; p < v1.size(); p++)
      arr[j++] = v1.get(p);
  }
}

// Utility function to print
// an array
static void printArr(int arr[],
                     int n)
{
  for (int i = 0; i < n; i++)
    System.out.print(arr[i] + " ");
}

// Driver Code
public static void main(String[] args)
{
  int arr[] = {1, 2, 3, 4, 5, 6};
  int n = arr.length;
  sortBySetBitCount(arr, n);
  printArr(arr, n);
}
}

// This code is contributed by 29AjayKumar
```

Output:

```
3 5 6 1 2 4
```

Method 4: Using MultiMap

Steps:

- Create a MultiMap whose key values will be the negative of the number of set-bits of the element.
- Traverse the array and do following for each element:
- Count the number set-bits of this element. Let it be ‘setBitCount’
- count.insert({(-1) * setBitCount, element})
- Traverse ‘count’ and print the second elements.
  Below is the implementation of the above approach:

- C++
- Java
- Python3
- C#

```
// Java program to implement
// simple approach to sort
// an array according to
// count of set bits.
import java.io.*;
import java.util.*;
class GFG
{

  // Function to count setbits
  static int setBitCount(int num)
  {
    int count = 0;
    while ( num != 0 )
    {
      if ( (num & 1) != 0)
        count++;
      num >>= 1;
    }
    return count;
  }

  // Function to sort By SetBitCount
  static void sortBySetBitCount(int[] arr, int n)
  {
    ArrayList<ArrayList<Integer>> count = new ArrayList<ArrayList<Integer>>
();
```

```
    // Iterate over all values and
    // insert into multimap
    for( int i = 0 ; i < n ; ++i )
    {
      count.add(new ArrayList<Integer>(Arrays.asList((-1) *
setBitCount(arr[i]), arr[i])));
    }

    Collections.sort(count, new Comparator<ArrayList<Integer>>() {
      @Override
      public int compare(ArrayList<Integer> o1, ArrayList<Integer> o2) {
        return o1.get(0).compareTo(o2.get(0));
      }
    });

    for(int i = 0; i < count.size(); i++)
    {
      System.out.print(count.get(i).get(1) + " ");
    }

  }

  // Driver code
  public static void main (String[] args)
  {

    int arr[] = {1, 2, 3, 4, 5, 6};
    int n = arr.length;
    sortBySetBitCount(arr, n);
  }
}

// This code is contributed by avanitrachhadiya2155
```

Output:

```
3 5 6 1 2 4
```

Time complexity: O(n log n)

Auxiliary Space: O(n)

From <https://www.geeksforgeeks.org/sort-array-according-count-set-bits/>