Sorting and Searching

**[PROBLEM]** Given a binary array, 0s and 1s. Of size N (N <= 106) Sort the array.

*vector*<int> A = { 1,0,0,1,1,1,1,0,0,0,0,1 };

N = 12

**Approach 1:**

Using built-in sort.

TC: NlogN

SC: 1

**Approach 2:**

TC: 2n

SC: n

Take two lists zeroes[] and ones[]

In the end, fill A[] by looping over zeroes[] first and then looping over ones[] second.

**Approach 3:**

TC: 2N

SC: 2

Count Zeroes, And fill the array left to right with 0s and remaining with 1s.

Sum Array, Fill from right to left with 1s (=sumOfArray), remaining with 0s.

|  |
| --- |
| void sortArray(int a[], int n) {  int cz = 0;  for (int i = 0; i < n; i++)  if (a[i] == 0)  cz++;  for (int j = 0; j < n; j++)  a[j] = (j < cz) ? 0 : 1;  } |

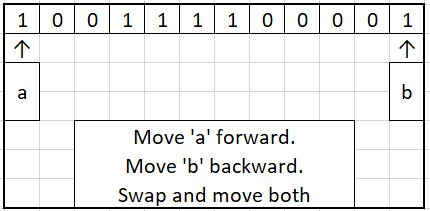
**Approach 4:**

Two Pointer Technique

TC: N

SC: O(1)

We can stop when a >=b



|  |
| --- |
| void sortBinArray(int a[], int n) {  int i = 0, j = n - 1;  while (i < j) {  if (a[i] == 1 && a[j] == 0) {  a[i++] = 0;  a[j--] = 1;  }  else if (a[i] == 0) i++;  else j--;  }  } |

**[PROBLEM]** Given the age of N (1 <= N <=106) different people working in a company in an array, we are required to sort the given array.

* Max age till which a person can work (with extension) is 70.
* Min working age for a person is 18.

*vector*<int> A = { 23,64,22,32,32,43,43,56,56,43,43,21,21,22,23,34,43,34 };

int N = 18;

**Approach 1:**

Built-In Sort

TC: NlogN

SC: 1 (logN max)

**Approach 2:**

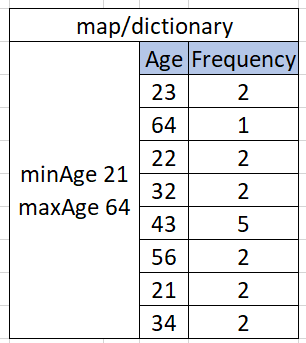
Implement our own sorting algo (Bubble/Selection/Insertion)

TC: N2

SC: 1

**Approach 3:**

Implement our own sorting algo (Quick/Merge)

TC: NlogN

SC: logN (Quick), N + logN (Merge)

**Approach 4:** Can we extend the idea of counting to solve this problem?

Using a map and counting freq.

TC: **NLogN (build map) + (maxAge — MinAge + 1) logN = O[ max(N, maxAge-MinAge+1)\*logN)]**

SC: N

**If Map is a HASHTABLE (then its operations will have time complexity as O(1)]**

**APPROACH 5:** (Implementing our own map using array of maxAge + 1 size)

Take hashTable[] of size MAX\_AGE + 1 and fill with 0s

Loop over ages[] to update freq of each age.

i = [minAge, maxAge] and refill array ages[]

**APPROACH 6:** (Implementing our own map using array of only necessary size)

This is called COUNT SORT.

TC: O(n) : 2n, 3n is fine (we need linear time complexity)

SC: (maxAge - minAge + 1) :

maxAge and minAge should be calculated from the array itself.

A = {45, 23,45, 24, 45, 25};

TC: (maxAge - minAge + 1) + n = RANGE + N

|  |
| --- |
| void sortAges(int ages[], int n) {  int minAge = findMinAge(ages, n); // 23  int maxAge = findMaxAge(ages, n); // 45  *vector*<int> hashTable(maxAge - minAge + 1);  for (int i = 0; i < n; i++) {  int idx = ages[i] - minAge;  hashTable[idx]++;  }  int k = 0;  for (int i = minAge; i <= maxAge; i++) {  while (hashTable[i] != 0) {  ages[k] = i;  hashTable[i]--; // Keep Decrementing freq  k++; // Next Index to Fill  }  }  } |

|  |
| --- |
| void sortAges(int ages[], int n) {  int minAge = findMinAge(ages, n); // 23  int maxAge = findMaxAge(ages, n); // 45  *vector*<int> hashTable(maxAge - minAge + 1);  for (int i = 0; i < n; i++) {  int idx = ages[i] - minAge;  hashTable[idx]++;  }  int k = 0;  for (int i = 0; i < maxAge - minAge + 1; i++) {  while (hashTable[i] != 0) {  ages[k] = i + minAge;  hashTable[i]--; // Keep Decrementing freq  k++; // Next Index to Fill  }  }  } |

## GENERALIZATIONS

When we have a range [L, R] to store in an array

* Size of array: R - L +1
* Each value maps to index = value - L
* Given index, we want to know the value = idx + L

**[PROBLEM]** Sort the given array in LINEAR TIME which can have max 106 elements, but elements of the array are in range -104 to 104

A = {-1000, -10000, 2, 3, 1, -10000, 10000, -10000, 10000, 10, 23, 34, 56, 100, 9999}

R = 104

L = -104

R-L+1=104-(-104)+1=1+2\*104

**FREQUENCY TABLE (Used a lot in online coding)** Faster than using a MAP.

* If Range of values is <= 106: apply this technique to get rid of the **TLE** when we use a MAP.
* Frequency table is useful when elements are in contiguous range.

Built-In Maps are of two types:

1. HashTable - Table like an array: O(1) : unordered\_map, HashMap
2. TreeMap - Uses BalancedTree: O(LogN): map, TreeMap

**[PROBLEM]** Given a string, find out the frequency of every character from 'a' to 'z' in it.

*string* S = "abracadabra32998923ACASDFASDFASDFASD";

Can we use array to store frequencies of 'a' to 'z'?

What size of array do we need? 26 (L='a'=97, R='z'= 122, size=R-L+1=26)

idx = char - L

And idx in freqTable maps to which character: idx + L

FreqTable problems can be solved with the help of MAP also.

**[PROBLEM]** Given an array having two repeated numbers, rest all occur one time.

**Arrays can be used like a SET and MAP if the range of values can be mapped to indices 0 to 106-1 max.** This will be our own HASHTABLE.