Merge Without Extra Space

```
//{ Driver Code Starts
#include <bits/stdc++.h>
using namespace std;



// } Driver Code Ends

class Solution {
  public:
    void mergeArrays(vector<int>& a,
vector<int>& b) {
        // code here
        int n = a.size(), m = b.size();
        int gap = (n + m + 1) / 2;  // Initial gap

        auto nextGap = [](int gap) {
            return (gap <= 1) ? 0 : (gap + 1) / 2;
        };

        while (gap > 0) {
            int i, j;

            for (i = 0; i + gap < n; i++) {
                if (a[i] > a[i + gap]) swap(a[i], a[i +
gap]);
            }

            for (j = gap > n ? gap - n : 0; i < n && j <
m; i++, j++) {
                if (a[i] > b[j]) swap(a[i], b[j]);
            }

            for (j = 0; j + gap < m; j++) {
                if (b[j] > b[j + gap]) swap(b[j], b[j +
gap]);
            }
            gap = nextGap(gap);
        }
    }
};



//{ Driver Code Starts.

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int t;
    cin >> t; // Inputting the test cases

    while (t--) {
        vector<int> a, b;

        // Reading the first array as a space-
separated line
        string arr1;
        getline(cin >> ws, arr1); // Use ws to
ignore any leading whitespace
        stringstream ss1(arr1);
        int num;
        while (ss1 >> num) {
```

```
        a.push_back(num);
    }

    // Reading the second array as a space-separated line
    string arr2;
    getline(cin, arr2);
    stringstream ss2(arr2);
    while (ss2 >> num) {
        b.push_back(num);
    }

    Solution ob;
    ob.mergeArrays(a, b);

    // Output the merged result
    for (int i = 0; i < a.size(); i++) {
        cout << a[i] << " ";
    }
    cout << endl;
    for (int i = 0; i < b.size(); i++) {
        cout << b[i] << " ";
    }
    cout << "\n";
    cout << "~\n";
  }

  return 0;
}

// } Driver Code Ends
```

Binary Search

```
class Solution {
public:
  int search(vector<int>& nums, int target) {
    int low = 0, high = nums.size() - 1;

    while (low <= high) {
      int mid = low + (high - low) / 2;

      if (nums[mid] == target)
        return mid;

      if (nums[mid] < target)
        low = mid + 1;
      else
        high = mid - 1;
    }

    return -1;
  }
};
```

Find Missing and Repeated Values

```
class Solution {
public:
  vector<int> findMissingAndRepeatedValues(vector<vector<int>>& grid) {
        int n = grid.size();
    int N = n * n;
```

```
    long long expectedSum = (N * (N + 1)) / 2;
    long long expectedSquareSum = (N * (N + 1) * (2 * N + 1)) / 6;

    long long actualSum = 0, actualSquareSum = 0;
    unordered_map<int, int> freq;
    int repeated, missing;

    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        int val = grid[i][j];
        actualSum += val;
        actualSquareSum += 1LL * val * val;
        freq[val]++;
        if (freq[val] == 2) {
          repeated = val;
        }
      }
    }

    long long sumDiff = actualSum - expectedSum;
    long long squareSumDiff = actualSquareSum - expectedSquareSum;

    missing = repeated - sumDiff;

    return {repeated, missing};
  }
};
```

Power of Three

```
class Solution {
public:
  bool isPowerOfThree(int n) {
    if (n <= 0) return false;
    while (n % 3 == 0) {
      n /= 3;
    }
    return (n == 1);
  }
};
```

Linear search

```
// Online C++ compiler to run C++ program online
#include <iostream>
using namespace std;

int linearsearch(int size, int arr[], int target){
  for (int i=0;i<size;i++){
    if(arr[i]==target){
      return i;
    }
  }
  return -1;
}
```

```
int main() {

   int n,target;

   cout<<"enter the size of the array";
   cin>>n;
   int arr[n];

   for(int i=0; i<n; i++){
      cin>>arr[i];
   }

   for(int i=0; i<n; i++){
      cout<<arr[i]<<" ";
   }

   cout<<"enter the array target";
   cin>>target;

   int result= linearsearch(n, arr, target);

   if(result!=-1){
      cout<<"target found"<<result;
   }
   else{
      cout<<"target not found"<<target;
   }
}
```

Merge sort code

jump game

```
class Solution {
public:
   bool canJump(vector<int>& nums) {
      int maxReach = 0;
      for (int i = 0; i < nums.size(); i++) {
         if (i > maxReach) return false;
         maxReach = max(maxReach, i +
nums[i]);
      }
      return true;
   }
};
```

Plus One

```
class Solution {
public:
   vector<int> plusOne(vector<int>& digits) {
      int n = digits.size();

      for (int i = n - 1; i >= 0; i--) {
         if (digits[i] < 9) {
            digits[i]++;
            return digits;
         }
         digits[i] = 0;
      }

      digits.insert(digits.begin(), 1);
      return digits;
```

```
    }
};
```

Sqrt(x)

```
class Solution {
public:
    int mySqrt(int x) {
        long long left=0;
        long long right =x;
        while(left<right){
            long long mid=left+((right-left+1)>>1);
            if(mid<=x/mid){
                left=mid;
            }
            else{
                right=mid-1;
            }
        }
        return static_cast<int>(left);
    }
};
```