Find the largest multiple of 3

Given an array of non-negative integers. Find the largest multiple of 3 that can be formed from array elements.

For example, if the input array is {8, 1, 9}, the output should be "9 8 1", and if the input array is {8, 1, 7, 6, 0}, output should be "8 7 6 0".

Method 1 (Brute Force)

The simple & straight forward approach is to generate all the combinations of the elements and keep track of the largest number formed which is divisible by 3.

Time Complexity: O(n x 2^n). There will be 2^n combinations of array elements. To compare each combination with the largest number so far may take O(n) time.
Auxiliary Space: O(n) // to avoid integer overflow, the largest number is assumed to be stored in the form of array.

Method 2 (Tricky)

This problem can be solved efficiently with the help of O(n) extra space. This method is based on the following facts about numbers which are multiple of 3.

1) A number is multiple of 3 if and only if the sum of digits of number is multiple of 3. For example, let us consider 8760, it is a multiple of 3 because sum of digits is 8 + 7+ 6+ 0 = 21, which is a multiple of 3.

2) If a number is multiple of 3, then all permutations of it are also multiple of 3. For example, since 6078 is a multiple of 3, the numbers 8760, 7608, 7068, ….. are also multiples of 3.

3) We get the same remainder when we divide the number and sum of digits of the number. For example, if divide number 151 and sum of it digits 7, by 3, we get the same remainder 1.

What is the idea behind above facts?
The value of 10%3 and 100%3 is 1. The same is true for all the higher powers of 10, because 3 divides 9, 99, 999, … etc.
Let us consider a 3 digit number n to prove above facts. Let the first, second and third digits of n be 'a', 'b' and 'c' respectively. n can be written as

```
n = 100.a + 10.b + c
```

Since (10^x)%3 is 1 for any x, the above expression gives the same remainder as following expression

```
 1.a + 1.b + c
```

So the remainder obtained by sum of digits and 'n' is same.

Following is a solution based on the above observation.

1. Sort the array in non-decreasing order.

2. Take three queues. One for storing elements which on dividing by 3 gives remainder as 0.The second queue stores digits which on dividing by 3 gives remainder as 1. The third queue stores digits which on dividing by 3 gives remainder as 2. Call them as queue0, queue1 and queue2

3. Find the sum of all the digits.

4. Three cases arise:
……**4.1** The sum of digits is divisible by 3. Dequeue all the digits from the three queues. Sort them in non-increasing order. Output the array.

……**4.2** The sum of digits produces remainder 1 when divided by 3.
Remove one item from queue1. If queue1 is empty, remove two items from queue2. If queue2 contains less than two items, the number is not possible.

……**4.3** The sum of digits produces remainder 2 when divided by 3.
Remove one item from queue2. If queue2 is empty, remove two items from queue1. If queue1 contains less than two items, the number is not possible.

5. Finally empty all the queues into an auxiliary array. Sort the auxiliary array in non-increasing order. Output the auxiliary array.

Based on the above, below is C implementation:

The below code works only if the input arrays has numbers from 0 to 9. It can be easily extended for any positive integer array. We just have to modify the part where we sort the array in decreasing order, at the end of code.

```c
/* A program to find the largest multiple of 3 from an array of elements */
#include <stdio.h>
#include <stdlib.h>

// A queue node
typedef struct Queue
{
    int front;
    int rear;
    int capacity;
    int* array;
} Queue;

// A utility function to create a queue with given capacity
Queue* createQueue( int capacity )
{
    Queue* queue = (Queue *) malloc (sizeof(Queue));
    queue->capacity = capacity;
    queue->front = queue->rear = -1;
    queue->array = (int *) malloc (queue->capacity * sizeof(int));
    return queue;
}

// A utility function to check if queue is empty
int isEmpty (Queue* queue)
{
    return queue->front == -1;
}

// A function to add an item to queue
void Enqueue (Queue* queue, int item)
{
    queue->array[ ++queue->rear ] = item;
    if ( isEmpty(queue) )
        ++queue->front;
}

// A function to remove an item from queue
int Dequeue (Queue* queue)
{
    int item = queue->array[ queue->front ];
    if( queue->front == queue->rear )
        queue->front = queue->rear = -1;
    else
        queue->front++;

    return item;
}

// A utility function to print array contents
void printArr (int* arr, int size)
{
    int i;
    for (i = 0; i< size; ++i)
        printf ("%d ", arr[i]);
}

/* Following two functions are needed for library function qsort().
   Refer following link for help of qsort()
   http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ */
int compareAsc( const void* a, const void* b )
{
    return *(int*)a > *(int*)b;
}
int compareDesc( const void* a, const void* b )
{
    return *(int*)a < *(int*)b;
```

```c
}

// This function puts all elements of 3 queues in the auxiliary array
void populateAux (int* aux, Queue* queue0, Queue* queue1,
                            Queue* queue2, int* top )
{
    // Put all items of first queue in aux[]
    while ( !isEmpty(queue0) )
        aux[ (*top)++ ] = Dequeue( queue0 );

    // Put all items of second queue in aux[]
    while ( !isEmpty(queue1) )
        aux[ (*top)++ ] = Dequeue( queue1 );

    // Put all items of third queue in aux[]
    while ( !isEmpty(queue2) )
        aux[ (*top)++ ] = Dequeue( queue2 );
}

// The main function that finds the largest possible multiple of
// 3 that can be formed by arr[] elements
int findMaxMultupleOf3( int* arr, int size )
{
    // Step 1: sort the array in non-decreasing order
    qsort( arr, size, sizeof( int ), compareAsc );

    // Create 3 queues to store numbers with remainder 0, 1
    // and 2 respectively
    Queue* queue0 = createQueue( size );
    Queue* queue1 = createQueue( size );
    Queue* queue2 = createQueue( size );

    // Step 2 and 3 get the sum of numbers and place them in
    // corresponding queues
    int i, sum;
    for ( i = 0, sum = 0; i < size; ++i )
    {
        sum += arr[i];
        if ( (arr[i] % 3) == 0 )
            Enqueue( queue0, arr[i] );
        else if ( (arr[i] % 3) == 1 )
            Enqueue( queue1, arr[i] );
        else
            Enqueue( queue2, arr[i] );
    }

    // Step 4.2: The sum produces remainder 1
    if ( (sum % 3) == 1 )
    {
        // either remove one item from queue1
        if ( !isEmpty( queue1 ) )
            Dequeue( queue1 );

        // or remove two items from queue2
        else
        {
            if ( !isEmpty( queue2 ) )
                Dequeue( queue2 );
            else
                return 0;

            if ( !isEmpty( queue2 ) )
                Dequeue( queue2 );
            else
                return 0;
        }
    }

    // Step 4.3: The sum produces remainder 2
    else if ((sum % 3) == 2)
    {
        // either remove one item from queue2
        if ( !isEmpty( queue2 ) )
```

```
            Dequeue( queue2 );

        // or remove two items from queue1
        else
        {
            if ( !isEmpty( queue1 ) )
                Dequeue( queue1 );
            else
                return 0;

            if ( !isEmpty( queue1 ) )
                Dequeue( queue1 );
            else
                return 0;
        }
    }

    int aux[size], top = 0;

    // Empty all the queues into an auxiliary array.
    populateAux (aux, queue0, queue1, queue2, &top);

    // sort the array in non-increasing order
    qsort (aux, top, sizeof( int ), compareDesc);

    // print the result
    printArr (aux, top);

    return top;
}

// Driver program to test above functions
int main()
{
    int arr[] = {8, 1, 7, 6, 0};
    int size = sizeof(arr)/sizeof(arr[0]);

    if (findMaxMultupleOf3( arr, size ) == 0)
        printf( "Not Possible" );

    return 0;
}
```

The above method can be optimized in following ways.
1) We can use Heap Sort or Merge Sort to make the time complexity O(nLogn).

2) We can avoid extra space for queues. We know at most two items will be removed from the input array. So we can keep track of two items in two variables.

3) At the end, instead of sorting the array again in descending order, we can print the ascending sorted array in reverse order. While printing in reverse order, we can skip the two elements to be removed.

Time Complexity: O(nLogn), assuming a O(nLogn) algorithm is used for sorting.