Rearrange an array in maximum minimum form

Given a sorted array of positive integers, rearrange the array alternately i.e first element should be maximum value, second minimum value, third second max, fourth second min and so on.

Examples:

```
Input  : arr[] = {1, 2, 3, 4, 5, 6, 7}
Output : arr[] = {7, 1, 6, 2, 5, 3, 4}

Input  : arr[] = {1, 2, 3, 4, 5, 6}
Output : arr[] = {6, 1, 5, 2, 4, 3}
```

Expected time complexity is O(n).

We strongly recommend that you click here and practice it, before moving on to the solution.

This problems becomes very easy if extra space is allowed. We can copy the array to another array and then place elements at alternate positions.

How to do in-place? The idea is to use the fact that numbers are positive. One by one place numbers at their correct positions and mark them negative.
How to find correct or output position for an element arr[i]? we can observe that the output follows below pattern for an input array.

```
// Output index j for an element arr[i]
If (i < n/2)
  j = 2*i + 1
Else
  j = 2*(n-1-i);
```

Below is C++ program based on above facts.

```
// C++ program to rearrange an array in minimum maximum form
// using O(1) space.
#include <bits/stdc++.h>
using namespace std;

// Puts max at first position, min at second position
// second max at third position, second min at fourth
// position and so on.
void rearrange(int arr[], int n)
{
   // Traverse through all numbers
   for (int i=0; i<n; i++)
   {
      int temp = arr[i];

      // If number is negative then we have already
      // processed it. Else process all numbers which
      // are to be replaced by each other in cyclic way
      while (temp > 0)
      {
         // Find the index where arr[i] should go
         int j = (i < n/2)? 2*i + 1 : 2*(n-1-i);

         // If arr[i] is already at its correct
         // position, mark it as negative
         if (i == j)
         {
            arr[i] = -temp;
            break;
         }

         // Swap the number 'temp' with the current number
         // at its target position
         swap(temp, arr[j]);

         // Mark the number as processed
         arr[j] = -arr[j];

         // Next process the previous number at target position
         i = j;
      }
   }

   // Change the number to original value
   for (int i=0; i<n; i++)
      arr[i] = -arr[i];
}

// Driver program to test above function
int main()
{
   int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
   int n = sizeof(arr)/sizeof(arr[0]);

   cout << "Original Array\n";
   for (int i=0; i<n; i++)
     cout << arr[i] << " ";

   rearrange(arr, n);

   cout << "\nModified Array\n";
   for (int i=0; i<n; i++)
     cout << arr[i] << " ";
   return 0;
}
```

Output:

```
Original Array
1 2 3 4 5 6 7 8 9
Modified Array
9 1 8 2 7 3 6 4 5
```

Time complexity of above solution is O(n) as every element is traversed at most

twice.