Count Reverse Pairs

- We will be having 2 nested For loops the outer loop having i as pointer. The inner loop with j as pointer and we will make sure that arr[i] > 2*arr[j] condition must be satisfied when we increment count.

```
int reversePairs(vector < int > & arr) {
    int Pairs = 0;
    for (int i = 0; i < arr.size(); i++) {
      for (int j = i + 1; j < arr.size(); j++) {
        if (arr[i] > 2 * arr[j]) Pairs++;
      }
    }
  return Pairs;
}
```

- Time Complexity : O(N^2)
- Space Complexity : O(1)

Optimal Solution : Using Merge Sort

In merge sort when we Sort for left and right we have two sorted arrays we make use of these arrays and check whether elements in left array are ≥ 2* element in right array if they are we move the pointer. As we know both arrays are sorted if arr[0] in left is greater than 2*arr[right] then all the elements after that element will be forming pair so what we do is When pointer is at an element where the condition is not satisfied we add all the elements to the count before it using cnt+=right-(mid+1); When it goes for next iteration we do not reset the cnt rather we start from counting from the current right pointing element and then again add right-mid+1 making sure that all the previous elem which satisfied the condition again satisfies the condition for the new window and get counted again if we reset the count then we have to again start iterating from the starting of right array which we don’t want so count is thus updated only when inner loop breaks.

```
#include <bits/stdc++.h>
int merge(vector<int> &arr,int low,int mid,int high)
```

```
{
    int cnt=0;
    int left=low,right=mid+1;
    vector<int> temp;
    while(left<=mid && right<=high)
    {
        if(arr[left]<arr[right])
        {
            temp.push_back(arr[left]);
            left++;
        }
        else
        {
            temp.push_back(arr[right]);
            right++;
        }
    }
    while(left<=mid)
    {
        temp.push_back(arr[left]);
        left++;
    }
    while(right<=high)
    {
        temp.push_back(arr[right]);
        right++;
    }
    for(int i=low;i<=high;i++)
    {
        arr[i]=temp[i-low];
    }
}
int countPairs(vector<int>& arr,int low,int mid,int high)
{
  int right=mid+1,cnt=0;
  for(int i=low;i<=mid;i++)
  {
    while(right<=high&& arr[i]>2*arr[right])
    {
      right++;
    }
    cnt+=right-(mid+1);
  }
  return cnt;
}
int mergeSort(vector<int> &arr,int low,int high)
{
    int cnt=0;
    if(low>=high)
        return 0;
    int mid=(low+high)/2;
    cnt+=mergeSort(arr,low,mid);
    cnt+=mergeSort(arr,mid+1,high);
    cnt+=countPairs(arr,low,mid,high);
```

```
    merge(arr,low,mid,high);
    return cnt;
}
int reversePairs(vector<int> &arr, int n){
  // Write your code here.
  return mergeSort(arr,0,n-1);
}
```

- Time Complexity - O(2NlogN) because countPairs also traverses the both two arrays that needs to be merged which adds NlogN to complexity.
- Space Complexity : O(N) merge array .