Running Median

Naive Approach :

For each new iteration store the array till index i in a new array sort it and then find the median for every iteration

Code :

```
#include<bits/stdc++.h>
void findMedian(int *arr, int n)
{
    // Write your code here
    int s=0;

    if(n==0)
    {
        return;
    }
   cout<<arr[0]<<" ";
    for(int i=1;i<n;i++)
    {
        int temp[i+1],k=i+1;
        for(int j=0;j<=i;j++)
        {
            temp[j]=arr[j];
        }
       sort(temp,temp+(i+1));
       if(i%2==0)
       {
           cout<<temp[k/2]<<" ";
       }
       else
       {
           cout<<(temp[k/2-1]+temp[k/2])/2<<" ";
       }
    }
}
```

- Time Complexity : O(N*(klog(k)+k))
- Space Complexity : O(N^2)

Optimal Approach :

For median if we can divide the whole array into two halves left and right and then pick the max element from left half and min element from right half then we can solve this problem. For left half we will make a maxheap and for right a minheap. We always make sure that we have maximum elements in maxheap because when array length is odd we only need the max element from right and when length is even then we need max element from right as well as min element from left.

So always size of maxheap needs to be greater.

Now when max heap is empty then we push the element into maxheap as well as when maxheap top element is greater than current element we push in maxheap else in minheap because we need all smaller numbers on left.

When maxheap contains element more than it needs to then we simply pop the top element from maxheap and push into minheap i.e when difference between sizes of both heaps exceed 1 but if maxheap size is less than minheap size that is maxheap contains less elements than minheap then we pop the top element of minheap and push into maxheap.

Finally the top element of both heaps are added if array size is even or both heaps are of equal size, else only top of maxheap is the median.

Code :

```
#include<bits/stdc++.h>
void findMedian(int *arr, int n)
{
    // Write your code here
    priority_queue<int,vector<int>,greater<int>> minheap;
    priority_queue<int> maxheap;
    for(int i=0;i<n;i++)
    {
        if(maxheap.size()==0 || maxheap.top()>=arr[i])
        {
            maxheap.push(arr[i]);
        } else {
          minheap.push(arr[i]);
        }
        if (maxheap.size() > minheap.size() + 1)
        {
            minheap.push(maxheap.top());
            maxheap.pop();
        }
        else if(maxheap.size()<minheap.size())
        {
```

```
            maxheap.push(minheap.top());
            minheap.pop();
        }
        if(maxheap.size()==minheap.size())
        {
            cout<<(maxheap.top()+minheap.top())/2<<" ";
        }
        else
        {
            cout<<maxheap.top()<<" ";
        }

    }
}
```

- Time Complexity : O(n(log(m) + log(k)))
- Space Complexity : O(N)