Kth Largest and smallest element

BRUTE FORCE :

Sorting and finding kth and n-k th element as it is 0based indexing k-1 and n-k-1 th element of array will be returned.

Code :

```
#include <bits/stdc++.h>
vector<int> kthSmallLarge(vector<int> &arr, int n, int k)
{
  // Write your code here.
  vector<int> ans;
  sort(arr.begin(),arr.end());
  int l=n-k;
  ans.push_back(arr[k-1]);
  ans.push_back(arr[l]);
  return ans;
}
```

- Time Complexity : O(NlogN)
- Space Complexity : O(1)

Better Approach :

Using min heap and max heap and popping out k-1 element, then the topmost element will be the smallest and largest element respectively.

Code :

```
#include <bits/stdc++.h>
int ksmaller(vector<int> &arr,int n,int k)
{
  priority_queue<int,vector<int>,greater<int>> pq;
  for(int i=0;i<n;i++)
  {
    pq.push(arr[i]);
  }
  while(k>1)
  {
    pq.pop();
    k--;
  }
  return pq.top();
}
int klarger(vector<int> &arr,int n,int k)
{
  priority_queue<int> pq;
  for(int i=0;i<n;i++)
  {
    pq.push(arr[i]);
  }
  while(k>1)
  {
    pq.pop();
    k--;
  }
  return pq.top();
}
vector<int> kthSmallLarge(vector<int> &arr, int n, int k)
{
  // Write your code here.
  int small=ksmaller(arr,n,k);
  int larger=klarger(arr,n,k);
  vector<int> ans;
  ans.push_back(small);
  ans.push_back(larger);
  return ans;
}
```

- Time Complexity : O(k+(n-k) log(k))
- Space Complexity : O(1)

Optimal Approach : quick sort partition algo