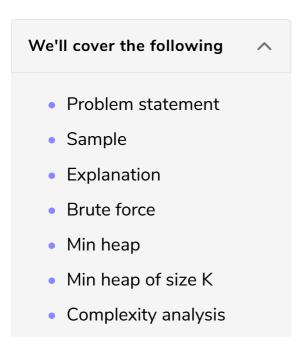
Solved Problem - Kth Largest element

In this lesson, we'll discuss a solved heap problem.



Problem statement

Given an array, A[], consisting of N integers; for a given K, find the K^{th} largest element in the array.

Input format

The first line consists of two integers N and K $(1 \le K \le N \le 10^6)$.

The second line consists of N integers representing the array A[] $(1 \le A[i] \le 10^6)$.

Sample

Input 1

5 1 3 6 5 1 4

Output 1

Input 2

```
5 1
3 6 5 1 4
```

Output 2

3

Explanation

Sample 1: K=1 means the 1^{st} largest in the entire array which is ${\color{red} f 6}$.

Sample 2: K=4 means the 4^{th} element if you sort the array in non-increasing order. i.e 4^{th} element in [6 5 4 3 1], which is 3.

Brute force

Explanation for sample two suggests a very simple solution:

- Sort the array
- ullet Print the K^{th} number from right

When sorting, taking O(NlogN)\$ time and printing the K^{th} element is a constant time operation.

Time complexity - O(NlogN).

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int kth_largest(vector<int> &v, int k) {
    sort(v.begin(), v.end());
    reverse(v.begin(), v.end());
    return v[k - 1];
}

int main() {
    int N = 5;
    vector<int> v = {3, 6, 5, 1, 4};
```

```
cout << kth_largest(v, 1) << "\n";
  cout << kth_largest(v, 4) << "\n";
  return 0;
}</pre>
```







[]

Min heap

Because of the property of binary heaps, we can get the minimum element in the tree in O(1).

Getting any other element is an O(N) operation because there is no true order to the nodes besides the root.

So, creating a heap out of the array to get the K^{th} element doesn't quite work out.

Min heap of size K

What if while iterating over the array, we add those elements in a priority queue (used interchangeably with heap), up to size K so that at each step the size of the heap is <=K and contains the largest K elements so far?

Formally, at index i:

- size(heap) < K, add to heap
- size(heap) = K, add if A[i] > root(heap)

If we do this at every step, the heap has the largest K elements so far in it.

In the end, print the smallest element in the heap (root).

Complexity analysis

Since the size of the heap is <=K, inserting an element or removing a root takes O(logK) time. In the worst-case scenario, each element will go in the heap and come out once. So, we do this 2*N times, we get O(NlogK).

In the end, getting the smallest element in the heap take O(1) time.

Time Complexity O(Nlog K)

Thire Complexity - O(1110911).

Important: We know that log factor in time complexity is a small value and thus log K is not much smaller than log N.

But take an example when K=8 and $N=10^6\,$

- log K = 3
- $logN \approx 20$

 \triangleright

We are talking about a solution that is seven times faster.

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
int kth_largest(vector<int> &v, int K) {
  priority_queue<int, vector<int>, greater<int> > pq;
  for (int i = 0; i < v.size(); i++){
    if (pq.size() < K) {
      pq.push(v[i]);
    else {
      if (v[i] > pq.top()) {
        pq.pop();
        pq.push(v[i]);
  int answer = pq.top();
 while (!pq.empty()) {
    answer = min(answer, pq.top());
    pq.pop();
  return answer;
}
int main() {
  int N = 5;
 vector<int> v = \{3, 6, 5, 1, 4\};
 cout<<kth_largest(v, 1)<<"\n";</pre>
 cout<<kth_largest(v, 4)<<"\n";</pre>
 return 0;
```

In the next chapter, we'll discuss another tree-based data structure - binary search

ti cc.