# **k largest(or smallest) elements in an array | added Min Heap method**

**Question:** Write an efficient program for printing k largest elements in an array. Elements in array can be in any order.

For example, if given array is [1, 23, 12, 9, 30, 2, 50] and you are asked for the largest 3 elements i.e., k = 3 then your program should print 50, 30 and 23.

**Method 1 (Use Bubble k times)**

Thanks to Shailendra for suggesting this approach.

1) Modify [Bubble Sort](http://en.wikipedia.org/wiki/Bubble_sort) to run the outer loop at most k times.

2) Print the last k elements of the array obtained in step 1.

Time Complexity: O(nk)

Like Bubble sort, other sorting algorithms like [Selection Sort](http://en.wikipedia.org/wiki/Selection_sort) can also be modified to get the k largest elements.

**Method 2 (Use temporary array)**

K largest elements from arr[0..n-1]

1) Store the first k elements in a temporary array temp[0..k-1].

2) Find the smallest element in temp[], let the smallest element be *min*.

3) For each element *x* in arr[k] to arr[n-1]

If *x* is greater than the min then remove *min* from temp[] and insert *x*.

4) Print final k elements of *temp[]*

Time Complexity: O((n-k)\*k). If we want the output sorted then O((n-k)\*k + klogk)

Thanks to nesamani1822 for suggesting this method.

**Method 3(Use Sorting)**

1) Sort the elements in descending order in O(nLogn)

2) Print the first k numbers of the sorted array O(k).

Time complexity: O(nlogn)

**Method 4 (Use Max Heap)**

1) Build a Max Heap tree in O(n)

2) Use [Extract Max](http://www.cs.utsa.edu/~dj/cs3343/lecture7.html) k times to get k maximum elements from the Max Heap O(klogn)

Time complexity: O(n + klogn)

**Method 5(Use Oder Statistics)**

1) Use order statistic algorithm to find the kth largest element. Please [see the topic selection in worst-case linear time](http://www.cse.ust.hk/~dekai/271/notes/L05/L05.pdf) O(n)

2) Use [QuickSort](http://en.wikipedia.org/wiki/Quicksort) Partition algorithm to partition around the kth largest number O(n).

3) Sort the k-1 elements (elements greater than the kth largest element) O(kLogk). This step is needed only if sorted output is required.

Time complexity: O(n) if we don’t need the sorted output, otherwise O(n+kLogk)

Thanks to [Shilpi](http://geeksforgeeks.org/forum/topic/print-k-largest-numbers) for suggesting the first two approaches.

**Method 6 (Use Min Heap)**

This method is mainly an optimization of method 1. Instead of using temp[] array, use Min Heap.

Thanks to [geek4u](http://geeksforgeeks.org/forum/topic/kth-largest-element) for suggesting this method.

1) Build a Min Heap MH of the first k elements (arr[0] to arr[k-1]) of the given array. O(k)

2) For each element, after the kth element (arr[k] to arr[n-1]), compare it with root of MH.

……a) If the element is greater than the root then make it root and call [heapify](http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/heapSort.htm) for MH

……b) Else ignore it.

// The step 2 is O((n-k)\*logk)

3) Finally, MH has k largest elements and root of the MH is the kth largest element.

Time Complexity: O(k + (n-k)Logk) without sorted output. If sorted output is needed then O(k + (n-k)Logk + kLogk)

All of the above methods can also be used to find the kth largest (or smallest) element.