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

March 1, 2010

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 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 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 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 O(n)
2) Use 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 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 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 for MH
b) Else ignore it.
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.

Please write comments if you find any of the above explanations/algorithms incorrect, or find better ways to solve the same problem.

References:
http://jonah.cs.elon.edu/sduvall2/courses/csc331/2006spring/Lectures/Order_statistics.ppt
http://en.wikipedia.org/wiki/Selection_algorithm

Asked by geek4u

36 comments so far

1. *Kasim* says:
   October 12, 2011 at 12:16 AM

   ```
   class ThreeBiggestDemo{
   public static void main (String args[]){
   int ar [] = new int[10];
   int big1, big2, big3, temp;

   ar[0] = 29;
   ar[1] = 2;
   ar[2] = 43;
   ar[3] = 8;
   ar[4] = 72;
   ar[5] = 17;
   ar[6] = 92;
   ar[7] = 113;
   ar[8] = 11;
   ar[9] = 0;

   big1 = ar[0];
   big2 = ar[1];
   big3 = ar[2];

   if (big1 > big2){
   if (big1> big3){
   big1 = big1;
   if (big2> big3) {
   big2 = big2;
   ```

```
big3 = big3;
}
else{
temp = big2;
big2 = big3;
big3 = temp;
}
}
else{
temp = big1;
big1 = big3;
big3 = big2;
big2 = temp;
}
}
else if (big2 > big3){
if (big1 > big3){
temp = big1;
big1 = big2;
big2 = temp;
big3 = big3;
}
else{
temp = big1;
big1 = big2;
big2 = big3;
big3 = temp;
}
}
else{
temp = big1;
big1 = big3;
big2 = big2;
big3 = temp;
}
for (int i=3; i big3){
if (ar[i] > big2){
if (ar[i] > big1){
temp = big1;
big1 = ar[i];
big2 = temp;
big3 = big3;
}
else{
big1= big1;
temp = big2;
big2 = ar[i];
big3 = temp;
}
}
else
```

```
big3 = ar[i];
}
}
System.out.println ("Big 1 " + big1);
System.out.println ("Big 2 " + big2);
System.out.println ("Big 3 " + big3);
}
}
```

Reply

2. *Nithish* says:
   September 12, 2011 at 12:10 AM

   How about randomized QuickSort with a small tweak?

   Choose the pivot for the QuickSort as any number from the given array and say after doing a single iteration of QuickSort, it was found the pivot element belonged to index 'X' of the say N array elements given.

   If we had to find the Kth smallest element, then if X was greater an K - 1, apply randomized QuickSort on the element 0 to X - 1 because we know that all element from 0 to X - 1 are smaller than the element X and X + 1 to N are greater than X.

   If we had to find the Kth largest element, then the element we have to find the element that belongs in the index N - K - 1 and apply the same logic.

   Reply

3. *geek* says:
   July 5, 2011 at 2:17 AM

   Building a heap of n elements require nLogn operations . How can you build a heap in O(n) in your examples ?

   Reply

4. *shanky* says:
   June 29, 2011 at 1:28 AM

   in method 4 how can we build a max heap in O(n).it requires O(nlogn)

   Reply

   - *Sandeep* says:
     June 29, 2011 at 10:03 AM

     @shanky: Build Heap takes O(n) time. See this G-Fact

     Reply

5. *Imran* says:
   May 4, 2011 at 8:29 AM

```
/* Based on Method 5 of Order Statistics. It finds kth Smallest element.
* Java Code using Partition as the first element.
* Comments and suggestions are appreciated.
* We can enhance this code by calculating Median of Medians to find pivot eac
```

```
* For reference consult this explanation.
* http://www.comp.dit.ie/rlawlor/Prob_Solv/Imperative_Algs/Quick%20Sort%20Exp
*/
public static int quickSelect( int a[], int l, int r, int x)
    {
                // if x is outOfRange return -1
        if(x < 1 || x > r) return -1;

        if(l==r) return a[l];
        if(l < r)
        {
          // divide and conquer
           int j = partition( a, l, r);
           int k = j-l+1;
           if(k == x)
               return a[j];
           else if(x < k)
               return quickSelect( a, l, j-1, x);
           else
               return quickSelect( a, j+1, r, x-k);
        }
        return -1;
    }

    public static int partition(int a[], int l, int r)
    {
        System.out.print("left = " + l +", right = " + r);
        int pivot = a[l], i, j, temp;
        i = l; j = r+1;
        System.out.print(", pivot = " + pivot +", ");
        while(true)
        {
          do ++i; while( i <= r && a[i] <= pivot);
          do --j; while( a[j] > pivot );
          if( i >= j ) break;
          temp = a[i];  a[i] = a[j];  a[j] = temp;
        }
       temp = a[l];  a[l] = a[j];  a[j] = temp;
       System.out.print("j = " + j);
       System.out.print(", a[j] = " + a[j] + ", ");
       System.out.println(Arrays.toString(a));
       return j;
    }
```

Reply

6. *WgpShashank* says:
   March 17, 2011 at 5:01 AM

   here you can also get Min-Max Heap

   http://forestofcode.blogspot.com/2010/12/c-min-max-heap-implementation.html

   Reply
7. *laxman* says:

March 16, 2011 at 7:25 PM

@sandeep...maderator,, venki

hi geeks please provide the working code fro the 6th method..this is highly in demand..plz..plz..try to post solution asap...everyone looking forward.

Thanks
Rahul

Reply

- *Sandeep* says:
  March 17, 2011 at 2:43 AM

  @Rahul: Following is the code for method 6.

```
#include<iostream>
#include<stdio.h>
using namespace std;

void swap(int *x, int *y) {
  int temp = *x;
  *x = *y;
  *y = temp;
}

class Heap
{
  int *arr; // pointer to array of elements in heap
  int heap_size;
public:
  Heap(int a[], int size);
  void buildminHeap();
  void minHeapify(int );
  void heapSort();
  void changeRoot(int x);
  int getRoot() {return arr[0];}
  int parent(int i){ return (i-1)/2; } ;
  int left(int i)  { return (2*i + 1); } ;
  int right(int i) { return (2*i + 2); } ;
};

Heap::Heap(int a[], int size) {
  heap_size = size;
  arr = a;
}

void Heap::changeRoot(int x)
{
   int root = arr[0];
   if (root < x)
   {
      arr[0] = x;
   }
```

```
    minHeapify(0);
}

void Heap::minHeapify(int i) {
  int l = left(i);
  int r = right(i);
  int largest;
  if (l < heap_size && arr[l] < arr[i])
    largest = l;
  else
    largest = i;
  if (r < heap_size && arr[r] < arr[largest])
    largest = r;
  if (largest != i)
  {
    swap(&arr[i], &arr[largest]);
    minHeapify(largest);
  }
}

void Heap::buildminHeap() {
  int i = (heap_size - 1)/2;
  while (i >= 0)
  {
    minHeapify(i);
    i--;
  }
}

int kthLargest(int arr[], int n, int k)
{
   Heap hp(arr, k);
   hp.buildminHeap();
   int i;
   for(i = k; i < n; i++)
   {
     hp.changeRoot(arr[i]);
   }
   return hp.getRoot();
}

int main()
{
    int k = 4;
    int arr[] = {12, 34, 10, 8, 9, 4, 56};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf(" %d ", kthLargest(arr, n, k));
    getchar();
    return 0;
}
```

I haven't tested it much. Once I test it for some significant number of cases and add some error handing code, I will add it to the original post.

Reply

- *wgpshashank* says:
  March 21, 2011 at 11:20 PM

  @sandeep u haven't done any boundary checking its not a good programming .in c no excpetion but in java u will get exception..at fisrt step itsel...hope u will rerposet it with correct boundary checing & with some test case as well

  Reply

8. *Algoseekar* says:
   March 12, 2011 at 5:13 AM

   @geeksforgeeks,venki,,all geeks everyone know that method 6 using heap is best method can anyone provide the exact implementation of that..

   Thank
   Algoseekar

   Reply
9. *reg_frenzy* says:
   February 17, 2011 at 12:15 PM

   We could use Winner trees approach. At each time, we compare two adjacent elements, thus reducing the comparisons by 2 at each iteration.

   Eg:
   12 5 8 1 78 90

   Outcome of First iteration:
   (Compare adjacent elements, winner is the bigger of the 2 elements)

   If it is odd, retain the last element.

   12 8 90

   Proceeding similarly,

   Outcome of Second iteration:

   12 90

   Outcome of Third iteration:

   90

   This is based on the concept of tournament trees. Now, the first largest element is 90. To find the second largest element, we play a tournament again with the elements with which the largest element(90) was compared, before becoming largest.

   In this example, the lit shrinks to:

   78 12

   So, when we compare these 2 elements, after first iteration, the winner is 78.

Complexity analysis:
This algorithm takes O(n + k log n) complexity, which for any fixed k independent of n is O(n).

Reply

10. *bunty* says:
[September 5, 2010 at 12:56 AM]

Another algorithm which will take O(n(1+k))
- Scan through the original array and create a temp array, "temp", of k elements, such that temp elements are in ascending order. This temp array will have our k largest elements.
- Arr[n]
- Make and array temp[k] with k (here 3) elements:
temp[i] = 0 for i = 0 to 1
max = temp[2] = Arr[0] = 0

temp[] = {0,0, Arr[0]);
// Comparing each element of Arr with temp[k-1] and place the //larger one in temp[k-1], maintaining temp in ascending order.
for (count=0;count<n;count++)
{// n comparison
if (temp[k-1] < Arr[count])
{
// Assigning Arr[count] to temp[k-1] and keeping array in
// order and over writing the lowest element in temp.
count2=0;
while(count2<temp[k-1]) // (k-2) shifts.
temp[count2] = temp[count2+1];

temp[count2] = Arr[count];

}

let us take and example of worst case;
Arr[] = {0,1,2,3,4,5,6,7};
and k = 3

1) temp = 0,0,0
on comparing temp[2]<Arr[1]
so new temp will be
temp = 0,0,1
2) now temp[2]<Arr[2]
new temp = 0,1,2
.....
when temp = 4,5,6
and temp[2]<Arr[7]
then new temp will be
temp = 5,6,7....which are the largest 3 numbers of Arr.

But it is the worst case, probably avg case would be little bit better.
Please do let me know for nay issue with the algo.

Reply

11. *RK* says:
August 26, 2010 at 1:17 AM

@ Method 5

I am not sure why we need to sort the elements(step 3)once we have already found the kth largest element. The question says the elements larger then kth largest element can be in any order.

In my opinion, the running time be O(n).

Please correct me if I am wrong.

Reply

- *GeeksforGeeks* says:
August 26, 2010 at 11:11 AM

@RK: Thanks for sharing your thoughts, we have added a note for this.

Reply

12. *Mahesh* says:
August 8, 2010 at 6:43 AM

Link in method 5 broken.

Reply

- *GeeksforGeeks* says:
August 8, 2010 at 12:21 PM

@Mahesh: Thanks for pointing this out. We ave fixed it.

Reply

13. *Ashish* says:
June 21, 2010 at 4:39 PM

what about forming a binary tree (as a preprocessing step) with each node storing the number of elements on its left child side? this is the solution i gave in my interview

Reply

14. *Virender* says:
May 14, 2010 at 2:58 PM

You can use the **Median of Median** method for this problem to reduce the Time complexity to **O(n)**. Median of Median modifies the partition method of quick sort to find "Good" pivot.

Explained here

Reply

- *kartik* says:
May 19, 2010 at 3:27 PM

I thing the method that you are suggesting and Method 5 in the above post are same.

Reply

- *dejected* says:
  May 23, 2010 at 8:40 PM

  Not really. There needs no partitioning as mentioned under method 5.

  Just pick the values which is greater than or equal to k-th largest element. Simple O(n) solution.

  Reply

15. *Anand* says:
    April 13, 2010 at 3:29 PM

    Hi
    I am interested in knowing how we gonna approach if the array contains , say billion integers. Obviously, we cannot put them all in memory and apply sorting due to memory constraint.

    suggestions, any?

    Reply
    - *kartik* says:
      April 13, 2010 at 3:42 PM

      You can use method 2 or method 6 because these methods do not require all the billion integers to be present in memory.

      Among these two methods, method 6 is a better choice.

      Reply
16. *Shailendra* says:
    March 29, 2010 at 12:23 AM

    We could apply Bubble Sort for K times so the largest/smallest k elements will be sorted.

    Reply
    - *GeeksforGeeks* says:
      March 29, 2010 at 2:38 PM

      Thanks for suggesting a new method. We have included it to the original post.

      Reply
17. *GeeksforGeeks* says:
    February 27, 2010 at 3:46 PM

    @ankit: This is quite interesting. Please see http://en.wikipedia.org/wiki/Binary_heap#Building_a_heap for proof that heap can be built in O(n) time.

    Reply
18. *ankit* says:
    February 27, 2010 at 3:37 PM

    How can you build a max heap tree in O(n) time? It should be O(nlogn).

    Reply

19. *GeeksforGeeks* says:
[February 27, 2010 at 1:31 AM]

@nesamani1822: Thanks for suggesting a new approach. We have added it to the original post. Keep it up!!

[Reply]

20. *nesamani1822* says:
[February 26, 2010 at 1:04 PM]

@Sandeep:

Here is the explanation for your example.

[1, 23, 12, 9, 30, 2, 50, 3]

1)Array creation
int *max=malloc(N,sizeof(int));
in your case N is 3

2) Initialize the array with first N elements
max[0] = 1
max[1] = 23
max[2] = 12

3) compare from the index 3 to 7

i=3
max[0] = 9
max[1] = 23
max[2] = 12

i=4
max[0] = 9
max[1] = 23
max[2] = 30

i=5
No change (Not greater than any element)
max[0] = 9
max[1] = 23
max[2] = 30

i=6
max[0] = 50
max[1] = 23
max[2] = 30

i=7
No change (Not greater than any element)
max[0] = 50
max[1] = 23
max[2] = 30

Reply

- *nikhil jain* says:
  March 20, 2011 at 3:30 PM

  it should be after each iteration find the min again in temp and replace this with the next larger number in arr[]

  Reply

21. *Sandeep* says:
    February 25, 2010 at 7:45 PM

    @nesamani1822: Could you please explain the approach with below example.

    Find 3 largest elements of the array [1, 23, 12, 9, 30, 2, 50, 3]

    I could easily understand first two steps, just have doubts about the third step.

    Reply

22. *nesamani1822* says:
    February 25, 2010 at 6:29 PM

    We can do in other way also.
    Here is the way how to do it.
    1) Create one array based on the N provided.
    2) Initialize that array with the first N elements of original array.
    3) then compare the next elements of the array with the newly initialized array and replace with lowest element of that newly initialized array.

    Reply

    - *Sam* says:
      November 10, 2010 at 4:21 AM

      Here is the implementation

```
int[] array = { 1, 23, 12, 9, 30, 2, 50, 3 };
            int k = 5;

            for (int i = k; i < array.Length - 1; i++)
            {
                //Find Min
                int min_index = 0;
                for (int j = 1; j  array[j])
                    {
                        min_index = j;
                        array[min_index] = array[j];
                    }
                }

                //Swap item if min < array[i]
                if (array[min_index] < array[i])
                {
                    int temp = array[min_index];
                    array[min_index] = array[i];
                    array[i] = temp;
```

```
            }
        }

        //Print output
        foreach (int item in array)
        {
            Console.Write("{0} ", item);
        }
```

Reply

23. *Sandeep* says:
February 24, 2010 at 7:46 AM

@Madhav: Method 1 talks about same. i.e., sort the elements and get the k largest elements.

@duke87: Could you please explain how the given code find k largest elements. Also, there seems to be typos in below lines.

```
for(h=p;hn) /* What is hn ?*/
search(arr,p,q-1,n,o);
else  /* ---> Else without if*/
 search(arr,q+1,r,n-q,o);
```

Reply

24. *Madhav* says:
February 23, 2010 at 10:55 PM

quick sort thrice nd u get d 3 largest/smallest elements ..

Reply

25. *duke87* says:
February 23, 2010 at 11:17 AM

algo called quick select which u write in method 3rd

```
#include
int partation(int [], int ,int);
void search(int[],int ,int ,int,int);
int main()
{
    int n;
    printf("enter the n for nth larest element\n");
    scanf("%d",&n);

    int arr[]={5,2,7,1,8,9,6};

    search(arr,0,6,n-1,n-1);
    //printf("%d ",partation(arr,0,6));
    getchar();
    getchar();
    return 0;
}

void search(int arr[],int p,int r, int n,int o)
```

```
{

  if(r>p)
  {
            int h;
            int q=partation(arr,p,r);
            for(h=p;hn)
                search(arr,p,q-1,n,o);
                else
                search(arr,q+1,r,n-q,o);
            }
  }
}

int partation(int a[],int p,int r)
{
              int i,j;
              j=p;
              i=j-1;
              while(j<=r)
              {
                      if(a[j]<a[r])
                      {
                                  i++;
                                  int temp=a[j];
                                  a[j]=a[i];
                                  a[i]=temp;
                      }
                      j++;
              }
              int temp=a[i+1];
              a[i+1]=a[r];
              a[r]=temp;

return i+1;
}
```

Reply

Comment

Name (*Required*) Email (*Required*) Website URI

Your Comment (Writing code? please paste your code between sourcecode tags)

```
[sourcecode language="C"]
/* Paste your code here (You may delete these
lines if not writing code) */
[/sourcecode]
```

Have Your Say

Notify me of followup comments via e-mail

Pingbacks/Trackbacks

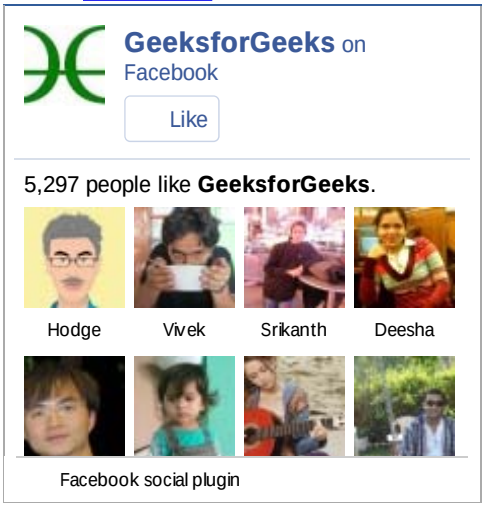
1. Applications of Heap Data Structure | GeeksforGeeks

Subscribe without commenting
E-Mail: Subscribe

- Search

Popular Tags

- GATE
- Java
- Dynamic Programming
- Divide & Conquer
- Backtracking
- Pattern Searching
- Operating Systems
- Recursion



Forum Latest Discussion

- Google interview question
  Last Post By: Gautam
  Inside: Interview Questions
- Recursion concept
  Last Post By: kartik
  Inside: Miscellaneous
- knights tour
  Last Post By: kartik
  Inside: Interview Questions
- Microsoft Interview Question for Software Testing (2-5 Years) about Data Structu [...]

Last Post By: Aashish Barnwal
Inside: Interview Questions

- Amazon Interview Question for Software Engineer/Developer about Trees
  Last Post By: camster
  Inside: Interview Questions
- how to debug the c code.....?
  Last Post By: DJ
  Inside: Interview Questions
- Count of paths
  Last Post By: codinglearner
  Inside: Interview Questions
- what is the output ?? and why
  Last Post By: kartik
  Inside: C/C++ Programming Questions

Popular Posts

- Sorted Linked List to Balanced BST
- Tree traversal without recursion and without stack!
- All permutations of a given string
- Memory Layout of C Programs
- Longest substring without repeating characters
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Understanding “extern” keyword in C
- Median of two sorted arrays

Forum Categories

- Interview Questions
- C/C++ Programming Questions
- Algorithms
- Trees specific questions
- Linked List specific questions
- Multiple Choice Questions
- Object oriented queries
- GPuzzles
- Operating Systems
- Miscellaneous
- Java specific Questions
- Perl specific Questions

Add Interview Questions

If you have attended an interview recently or have good interview questions to share with the fellow

geeks and get the best solution, please add your questions to Add Interview Questions page

Recent Comments

- kartikaditya on Length of the longest substring without repeating characters
- kartikaditya on Find the maximum element in an array which is first increasing and then decreasing
- Avinash Kumar on Write a function to reverse a linked list
- Avinash Kumar on Write a function to reverse a linked list
- kartikaditya on Minimum number of jumps to reach end
- kartikaditya on Find the largest BST subtree in a given Binary Tree
- kartikaditya on Find the largest BST subtree in a given Binary Tree
- Frederic on Print all interleavings of given two strings