ABSTRACT

In computer science, a **selection algorithm** is an algorithm for finding the *k*th smallest number in a list or array; such a number is called the *k*th *order statistic*. This includes the cases of finding the minimum, maximum, and median elements. There are O(*n*) (worst-case linear time) selection algorithms, and sublinear performance is possible for structured data; in the extreme, O(1) for an array of sorted data. Selection is a subproblem of more complex problems like the nearest neighbor and shortest path problems. Many selection algorithms are derived by generalizing a sorting algorithm, and conversely some sorting algorithms can be derived as repeated application of selection.

This new algorithm although has worst case of O(n^2), the average case is of near linear time for an unsorted list.

ALGORITHM

Legend:

arr: Unsorted array of numbers.

k: Kth smallest element in the array.

n: Total number of elements in the unsorted array.

Ns: Closest elememt smaller than Kth element encountered.

Nl: Closest element greater than Kth element encountered.

smallcount: Keeps count of elements smaller than arr[j].

largecount: Keeps count of elements larger than arr[j].

smalllimit: Number of possible elements smaller than Kth element.

largelimit: Number of possible elements greater than Kth element.

Algorithm starts here:

1. Start

2. Input the unsorted array (arr[]), its size (n) and value of k (Kth smallest element)

3. smalllimit=k-1

   largelimit-n-k

   Ns= -infinity (minimum value possible)

   Nl= +infinity (maximum value possible)

4. Note: Keep count of smallcount, largecount, smalllimit, largelimit, Ns and Nl

  from i=0 till i<n:

   {

```
smallcount=largecount=0
j=i+1

if i==n-1 then:
  print the element (this is the Kth element)

if arr[i] doesn't lie between Ns and Nl, then:
{
    if arr[i]<Ns then:
      decrement smalllimit by 1
    else:
      decrement largelimit by 1
    i++
    continue the loop
}

from j=i+1 till j<n:
{
    if arr[j] < arr[i] then:
      increment smallcount by 1
    else:
      increment largecount by 1

    if smallcount>smalllimit then:
      Decrement largelimit by 1
      Nl= arr[i]
      break out of the loop

    else if largecount > largelimit:
```

Decrement smalllimt by 1

Ns= arr[i]

break out of the loop

Increment j by 1

}

if smallcount is equal to smalllimit and largecount is equal to largelimit

print the element in the array (ie. print arr[i])

}

5. End

Analysis of the Algorithm

Best case:

When the Kth element is present in the **first position in the unsorted array**. For example,

Arrangement: **6** 4 7 1 9 0 11 (unsorted)

n=7, k=4

0 1 4 **6** 7 9 11 (sorted)

Time complexity- O(n)

Worst case:

When Kth element is present at **the end of the unsorted array** and elements are arranged in alternate order smaller and greater than Kth element (from greatest to smallest).

For example,

Arrangement: 11 0 9 1 7 4 **6** (unsorted)

n=7, k=4

0 1 4 **6** 7 9 11 (sorted)

Time complexity: O(n^2)

Conclusion:

The algorithm works well if either the element is present in **the first position** of the unsorted array or when elements **just smaller and greater than Kth element** is present in the initial positions of the unsorted array.

Example: When elements just smaller and greater than Kth element is in 1st and 2nd position

Arrangement: 7 4 11 0 9 1 **6** (unsorted)

n=7, k=4

0 1 4 **6** 7 9 11 (sorted)

In such a case also complexity is O(n).

SOURCE CODE

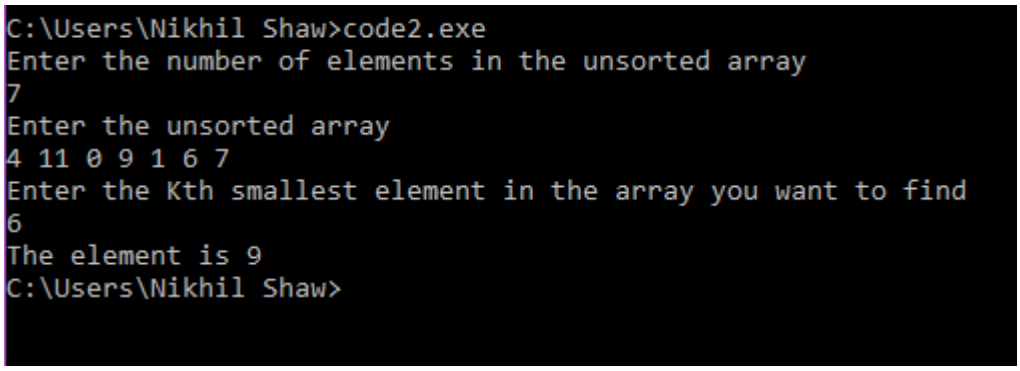
```
/*
Created by: Nikhil Shaw
Date of creation: 29 Sep 2016
Aim: To find the Kth smallest element in an unsorted array of numbers
*/

/*
arr: Unsorted array of numbers.
k: Kth smallest element in the array.
n: Total number of elements in the unsorted array.
Ns: Closest elememt smaller than Kth element encountered.
Nl: Closest element greater than Kth element encountered.
smallcount: Keeps count of elements smaller than arr[j].
largecount: Keeps count of elements larger than arr[j].
smalllimit: Number of possible elements smaller than Kth element.
largelimit: Number of possible elements greater than Kth element.
*/


/*
Following is the code in C++
*/


#include<iostream>

using namespace std;

int main()
{

  int arr[20], smalllimit, largelimit, smallcount, largecount, Ns, Nl, k, n, i, j;
  cout<<"Enter the number of elements in the unsorted array"<<endl;
  cin>>n;
  cout<<"Enter the unsorted array"<<endl;
  for(i=0;i<n;i++)
    cin>>arr[i];
  cout<<"Enter the Kth smallest element in the array you want to find"<<endl;
  cin>>k;
  while((k<1)||(k>n))                    // condition for valid value of k
  {
    cout<<"Enter K between 1 and "<<n<<endl;
    cin>>k;
  }
  smalllimit=k-1;
  largelimit=n-k;
  Ns=-30000;                             // very low value initially
  Nl=30000;                              // very high value initially
  for(i=0;i<n;i++)
  {
```

```
    smallcount=0;
    largecount=0;
     j=i+1;

    if(i==n-1)                          // Only one element left in the array to process
    {
      cout<<"The element is "<<arr[i];
      break;
    }
    else if( (arr[i]>Nl)||(arr[i]<Ns) )      // if element doesn't lie between Ns and Nl
    {
      if(arr[i]<Ns)
       {
        smalllimit--;
       }
      else
        largelimit--;
      continue;
    }

    else
    {
      for(j=i+1;j<n;j++)
      {
        if(arr[j]<arr[i])
          smallcount++;
        else
          largecount++;

        if(smallcount>smalllimit)
        {
          largelimit--;
          Nl=arr[i];
          break;
        }
        else if(largecount>largelimit)
        {
          smalllimit--;
          Ns=arr[i];
          break;
        }
        else{}
      }
    }

    if( (smallcount==smalllimit)&&(largecount==largelimit) )     // It is the Kth elements
with exactly k-1 elements smaller than it and n-k elements greter than it.
    {
      cout<<"The element is "<<arr[i];
      break;
    }
  }

return 0;
}
```

TEST RUN

The algorithm was tested by generating random numbers between 0 and 1000 for more than 100 times. Each time the code output the correct answer (crosschecked using sorting and printing kth element).

Following is a screenshot of a test run:



Explaination:

Unsorted array:

4 11 0 9 1 6 7

Ns=-infinity, Nl= +infinity

n=7, k=6

largelimit= 1 (ie. n-k), smalllimit= 5 (ie. k-1)

1. considering arr[i]= 4

    4 lies between Ns(=-infinity) and Nl(=+infinity),

    smallcount=largecount=0

    arr[j]=11 > 4

    smallcount= 0, largecount= 1

    arr[j]=0 <4

    smallcount=1 , largecount=1

arr[j]=9 >4

smallcount=1 , largecount=2

arr[j]=1 <4

smallcount=2, largecount=2

largecount > largelimit

Ns=4

smalllimit= smalllimit-1 (ie. 4)

2. considering arr[i]=11

11 lies between Ns(=4) and Nl(=+infinity)

smallcount=largecount=0

arr[j]=0 <11

smallcount=1, largecount=0

arr[j]=9 < 11

smallcount=2 , largecount=0

arr[j]=1 <11

smallcount=3 , largecount=0

arr[j]= 6<11

smallcount=4 , largecount=0

arr[j]=7<11

smallcount=5 , largecount=0

smallcount>samllimit

Nl=11

largelimit= largelimit-1 (ie. 0)

3. considering arr[i]=0

0 doesn't lies between Ns(=4) and Nl(=11)

0<Ns

smalllimit=smalllimit-1(ie 3)

4. considering arr[i]=9

9 lies between Ns(=4) and Nl(=11)

arr[j]=1<9

smallcount=1 , largecount=0

arr[j]=6<9

smallcount=2 , largecount=0

arr[j]=7<9

smallcount=3 , largecount=0

Now, smallcount==smalllimit and largecount==largelimit

print 9 (9 is kth element, k=6)

REFERENCES

wikipedia.org-
https://en.wikipedia.org/wiki/Selection_algorithm

ABOUT THE AUTHOR

Nikhil Shaw is currently persuing undergraduate degree in Infornmation Technology at SRM University, Kattankulathur campus, Chennai, India.

Contact :

nikhilshaw_ra@srmuniv.edu.in

shaw.nikhil05@gmail.com