Median-finding Algorithm

Median-finding algorithms (also called linear-time selection algorithms) use a divide and conquer strategy to efficiently comput the i^{th} smallest number in an unsorted list of size n, where i is an integer between 1 and n. Selection algorithms are often used part of other algorithms; for example, they are used to help select a pivot in quicksort and also used to determine i^{th}-order statistics such as the maximum, minimum, and median elements in a list.

Contents

Median-finding Algorithm

The Median-of-medians Algorithm

Implementation of the Median-finding Algorithm

Complexity of the Median-of-medians Algorithm

See Also

References

Median-finding Algorithm

The problem a median-finding algorithm solves is the following:

> Given an array $A = [1, \ldots, n]$ of n numbers and an index i, where $1 \leq i \leq n$, find the i^{th} smallest element of A.

This problem can certainly be solved using a sorting algorithm to sort a list of numbers and return the value at the i^{th} index. However, many sorting algorithms can't go faster than $n \log n$ time. A median-finding algorithm can find the i^{th} smallest elemer a list in $O(n)$ time.

In practice, median-finding algorithms are implemented with randomized algorithms that have an expected linear running time. However, this wiki will focus on the median-of-medians algorithm, which is a deterministic algorithm that runs in linear time.

The Median-of-medians Algorithm

The median-of-medians algorithm is a deterministic linear-time selection algorithm. The algorithm works by dividing a list into sublists and then determines the approximate median in each of the sublists. Then, it takes those medians and puts them into a and finds the median of that list. It uses that median value as a **pivot** and compares other elements of the list against the pivot. I element is less than the pivot value, the element is placed to the left of the pivot, and if the element has a value greater than the pivot, it is placed to the right. The algorithm recurses on the list, honing in on the value it is looking for.

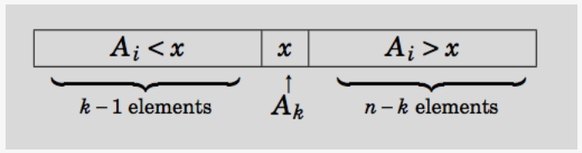
Median-of-medians Algorithm[1]

The algorithm takes in a list and an index— `median-of-medians(A, i)` . Assume that all elements of A are distinct (though the algorithm can be further generalized to allow for duplicate elements).

1. Divide the list into sublists each of length five (if there are fewer than five elements available for the last list, that is fine).
2. Sort each sublist and determine the median. Sorting very small lists takes linear time since these sublists have five elements, and this takes $O(n)$ time. In the algorithm described on this page, if the list has an even number of elements,

take the floor of the length of the list divided by 2 to find the index of the median.

3. Use the median-of-median algorithm to recursively determine the median of the set of all the medians.
4. Use this median as the pivot element, x. The pivot is an approximate median of the whole list and then each recursive step hones in on the true median.
5. Reorder A such that all elements less than x are to the left of x, and all elements of A that are greater than x are to the right. This is called partitioning. The elements are in no particular order once they are placed on either side of x. For example, if x is 5, the list to the right of x maybe look like $[8, 7, 12, 6]$ (i.e. not in sorted order). This takes linear time since $O(n)$ comparisons occur—each element in A is compared against x only.



[1]

6. Let k be the "rank" of x, meaning, for a set of numbers S, x is the k^{th} smallest number in S.

7a. If $i = k$, then return x.
7b. If $i < k$, then recurse using median-of-medians on $(A[1, \ldots, k-1], i)$.
7c. If $i > k$, recurse using the median-of-medians algorithm on $(A[k+1, \ldots, i], i-k)$.

EXAMPLE

Show the steps for the median-of-medians algorithm to find the third lowest score in the list, A, of exam scores.

Note: Some implementations of this algorithm, like the one below, are zero-indexed, meaning that the 0^{th} lowest score will be the lowest score in the list. In this example, use zero-indexing—so the third lowest score will be the 4^{th} element from the left in a sorted list.

$$A = [25, 21, 98, 100, 76, 22, 43, 60, 89, 87]$$

Show Answer

First, we break the list into lists of five elements:

$$A_1 = [25, 21, 98, 100, 76] \quad \text{and} \quad A_2 = [22, 43, 60, 89, 87].$$

Sort each list:

$$A_1 = [21, 25, 76, 98, 100] \quad \text{and} \quad A_2 = [22, 43, 60, 87, 89].$$

Then, get the median out of each list and put them in a list of medians, M :

$$M = [76, 60].$$

Sort this: $M = [60, 76]$.

Pick the median from that list—since the length of the list is 2, and we determine the index of the median by the length of the list divided by two: we get $\frac{2}{2} = 1$, the index of the median is 1, and M[1] = 76.

Use this as the pivot element and put all elements in A that are less than 76 to the left and all elements greater than 76 to the right:

$$A' = [25, 22, 43, 60, 21, 76, 100, 89, 87, 98].$$

Find the index of 76, which is 5. How does 5 compare with 3? Since $5 > 3$, we must recurse on the left half of the list A', which is $[25, 22, 43, 60, 21]$.

This list is only five elements long, so we can sort it and find what is at index 3: $[21, 22, 25, 43, 60]$ and 43 is at index three.

So, 43 is the fourth smallest number of A. □

Implementation of the Median-finding Algorithm

Here is a Python implementation of the median-of-medians algorithm[2]. This implementation works on lists that have unique elements (no repeated elements).

Python

```python
def median_of_medians(A, i):

    #divide A into sublists of len 5
    sublists = [A[j:j+5] for j in range(0, len(A), 5)]
    medians = [sorted(sublist)[len(sublist)/2] for sublist in sublists]
    if len(medians) <= 5:
        pivot = sorted(medians)[len(medians)/2]
    else:
        #the pivot is the median of the medians
        pivot = median_of_medians(medians, len(medians)/2)

    #partitioning step
    low = [j for j in A if j < pivot]
    high = [j for j in A if j > pivot]

    k = len(low)
    if i < k:
        return median_of_medians(low,i)
    elif i > k:
        return median_of_medians(high,i-k-1)
    else: #pivot = k
        return pivot

#Here are some example lists you can use to see how the algorithm works
#A = [1,2,3,4,5,1000,8,9,99]
#B = [1,2,3,4,5,6]
#print median_of_medians(A, 0) #should be 1
#print median_of_medians(A,7) #should be 99
#print median_of_medians(B,4) #should be 5
```

Note the source cited here does not have a completely correct implementation but did inspire this (better) implementation.

TRY IT YOURSELF

Using the algorithm described for the median-of-medians selection algorithm, determine what the list of medians will be on the following input:

$$A = [1, 2, 3, 4, 5, 1000, 8, 9, 99].$$

`median_of_medians(A,7)`

- ○ $[5]$
- ○ $[3, 99]$
- ○ $[5, 1000]$
- ○ $[8, 9]$

Hint: In the code, `medians` is the list of these medians.

Bonus: Feel free to run the code and add some strategic print messages.

EXAMPLE

Say you wanted to use the above implementation to find the i^{th} *largest* element in A instead of the i^{th} smallest. What change could you make to the implementation (not to the function's inputs) to achieve this?

Show Answer

Swap `low` and `high` in the partitioning step so that

```
Python
high = [j for j in A if j < pivot]
 low = [j for j in A if j > pivot]
```

Make the changes and test it out with the following test cases:

```
Python
B = [1,2,3,4,5,6]
print median_of_medians(B,0) #the first largest element should be 6
print median_of_medians(B,5) #the fifth largest element should be 1 (remember 0 indexing)
```

Now try the next example to see how you can find the largest element by carefully selecting an i value.

EXAMPLE

What could you input to the original implementation above to find the largest element in a list?

Show Answer

The largest element of a list will always be the "least smallest" element. For example, in a list of length 10, the least smallest element in the list is the ninth smallest (remember zero-indexing where the zeroth smallest is the smallest element). More generally, to find the largest element in the list, call `median_of_medians(A, len(A)-1)`.

Try this out with the following test cases:

```
Python
D = [1,2,3,4,5,6] # 6 is the largest (least small) element in D
print median_of_medians(D, len(D)-1)

E = [9,5,4,3] #9 is the largest (least small) element in E
print  median_of_medians(E, len(E)-1)
```

EXAMPLE

How could you select an i value so that you can find the i^{th} largest value in A without modifying the original implementation itself?

Show Answer

The value you input for the i variable would be `len(A) - x - 1\)`, where x is the number x^{th} largest value you want to find. □

Complexity of the Median-of-medians Algorithm

The median-of-medians algorithm runs in $O(n)$ time. n is divided into $\frac{n}{5}$ sublists of five elements each. If M is the list of all of t
medians from these sublists, then M has $\frac{n}{5}$—one median for each of the $\frac{n}{5}$ sublists. Let's call the median of this list (the media
the medians) p. Half of the $\frac{n}{5}$ elements in M are less than p. Half of $\frac{n}{5} = \frac{n}{10}$. For each of these $\frac{n}{10}$ elements, there are two
elements that are smaller than it (since these elements were medians in lists of five elements—two elements were smaller and tw
elements were larger). Therefore, there are $\frac{3n}{10} < p$ and, in the worst case, the algorithm may have to recurse on the remaining
elements. The time for dividing lists, finding the medians of the sublists, and partitioning takes $T(n) = T\left(\frac{n}{5}\right) + O(n)$ time, a
with the recursion factored in, the overall recurrence to describe the median-of-medians algorithm is

$$T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n).$$

The master theorem can be used to show that this recurrence equals $O(n)$.

Why 5?

The median-of-medians divides a list into sublists of length five to get an optimal running time. Remember, finding the median o
small lists by brute force (sorting) takes a small amount of time, so the length of the sublists must be fairly small. However, adjus
the sublist size to three, for example, does change the running time for the worse.

If the algorithm divided the list into sublists of length three, p would be greater than approximately $\frac{n}{3}$ elements and it would be
smaller than approximately $\frac{n}{3}$ elements. This would cause a worst case $\frac{2n}{3}$ recursions, yielding the recurrence
$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n)$, which by the master theorem is $O(n \log n)$, which is slower than linear time.

In fact, for any recurrence of the form $T(n) \leq T(an) + T(bn) + cn$, if $a + b < 1$, the recurrence will solve to $O(n)$, and if
$a + b > 1$, the recurrence is usually equal to $\Omega(n \log n)$. [3]

The median-of-medians algorithm could use a sublist size greater than 5—for example, 7—and maintain a linear running time.
However, we need to keep the sublist size as small as we can so that sorting the sublists can be done in what is effectively
constant time.

See Also

- Quicksort
- Order Statistics

References

1. Moshkovitz, D., & Tidor, B. *Introduction & Median Finding*. Retrieved May 30, 2016, from http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-design-and-analysis-of-algorithms-spring-2012/lecture-notes/MIT6_046JS12_lec01.pdf

2. , g. *python Implementing median of medians algorithm*. Retrieved May 30, 2016, from https://www.reddit.com/r/learnprogramming/comments/3ld88o/pythonimplementing_median_of_medians_algorithm/
3. Trevisan, L. *W4231: Analysis of Algorithms*. Retrieved May 31, 2016, from http://people.eecs.berkeley.edu/~luca/w4231/fall99/slides/l3.pdf