Data Selection

Richard Hua

Semester 2, 2021

Introduction

Data selection is closely related to sorting.
Instead of rearranges the input list, data selection aims to find the kth smallest item.
This is also called the item of **rank** k or the kth order statistic.

Continued

We have seen with Selection sort and Heapsort that if we can select (the largest or the smallest item), then we can sort.
The converse is also true. Given a list of n items, we can sort it and then select the desired order static easily.

Question

How would different data structures make a difference in selecting the kth order statistic after the list is sorted?

Continued

Question

How would different data structures make a difference in selecting the kth order statistic after the list is sorted?

Retrieving the kth element in an array is $\Theta(1)$.
$\Theta(k)$ in linked-list.

Continued

Can we do it efficiently without sorting the input array?
Example: We may have a large data base of a large population's income, and we wish to know the median income.
This question is easy to answer for some special cases.
Example: if $k = 1$ or n, we can just build a min- or max-heap and retrieve the root.
Or just do a sequential search.

Exercise

Exercise

Both sequential search and 'Heapselect' takes $\Theta(n)$ time, which would you prefer in practice?

Exercise

Exercise

Both sequential search and 'Heapselect' takes $\Theta(n)$ time, which would you prefer in practice?

Both method have the same asymptotic time complexity, but the constant might be different.
Sequential search does exactly $n-1$ comparisons in all cases.
MAKEHEAP does $2\lfloor n/2 \rfloor$ number of comparisons and 0 swaps in the best case.
What about average and worst case?

Data selection

Not an easy question to answer for general k (e.g. if $k = \frac{n}{2}$, then we have the problem of finding the median).
Building a heap and repeatedly removing the root would be $\Omega(n \log n)$ in this case.

Quickselect

Quickselect works well on this problem.
Very similar to Quicksort.
Both good and bad.

Quickselect

Require: $0 \leqslant l \leqslant r \leqslant n-1$

function QUICKSORT(list $a[0 \cdots n-1]$, integer l, integer r)

if $l < r$ **then**

$i \leftarrow \text{PIVOT}(a, l, r)$

$j \leftarrow \text{PARTITION}(a, l, r, i)$

$\text{QUICKSORT}(a, l, j-1)$

$\text{QUICKSORT}(a, j+1, r)$

return a

Quickselect

Suppose that the pivot is at index j after partition.
The left sublist has j items and the right sublist has $n-j-1$ items.
If $k <= j$, then the item we are looking for is in the left sublist.
If $k = j+1$, then the pivot is the kth order statistic.
If $k > j+1$, then we search the right sublist.

Quickselect

Require: $0 \leqslant l \leqslant r \leqslant n-1$

function QUICKSELECT(list $a[0 \cdots n-1]$, integers l, r, k)

if $l \leqslant r$ **then**

$i \leftarrow \text{PIVOT}(a, l, r)$

$j \leftarrow \text{PARTITION}(a, l, r, i)$

$q \leftarrow j - l + 1$

if $k = q$ **then return** $a[j]$

else if $k < q$ **then return** $\text{QUICKSELECT}(a, l, j-1, k)$

else return $\text{QUICKSELECT}(a, j+1, r, k-q)$

else return 'Not Found'

Note that the search shouldn't fail unless k is outside the correct range (e.g. $k = 2n$).

Best case time complexity

Best case is quite straightforward, the pivot from the first iteration happens to be what we are looking for.
$\Theta(n)$ number of comparisons and swaps (exact number depends on the PARTITION method we use).

Worst case time complexity

Worst case is the same as Quicksort. Very unbalanced left and right sublists.
We can write it as a recurrence formula again.
$T(n) = T(n-1) + n,\ T(0) = 0$. Solving it results in a $\Theta(n^2)$ in the worst case.

Average case time complexity

Again, average-case analysis is quite difficult to do here.
Let's think about how to derive the formula.
Assuming k is within valid range (i.e. $1 \leqslant k \leqslant n$). Suppose the pivot is at index i after partition. Then we need to do a recursive call on a list of size either i or $n-1-i$.
Again, the formula has an expression of the form $\sum_{i=0}^{n-1} T(i) + T(n-i-1)$ similar to Quicksort.
Note that only 1 sublist from each pair is chosen so the average-case running time is

$$T(n) = \frac{1}{n} \sum_{i=0}^{n-1} T(i) + cn.$$

Average case time complexity

Theorem

The average-case time complexity of Quickselect is $\Theta(n)$.

Solving the recurrence will result in a linear function.

Additional notes

Everything we saw with Quicksort also applies to Quickselect (e.g. pivot strategy, larger non-recursive base case, etc.).

Question

Can we do better than $\Theta(n)$ time? No

Optimal time for sequencial search

We will do another analysis.

if the algorithm is less than linear time, then it means it doesn't have time to check all the items in the input list

Exercise

Exercise

To find p keys of fixed ranks, $k_1, k_2, \cdots, k_p$, in an unordered array, we can either:

1. *Run Quickselect p times.*
2. *Or use Quicksort once to order the array and simply fetch the desired p keys.*

Find a condition (in terms of p and n) when on the average the second method is more efficient. Which method is better if $n = 10^6$ and $p = 10$? You can ignore the time to fetch the p items.

Exercise

We will use $T_1(n, p)$ and $T_2(n, p)$ to denote the average running time of the two method.
In average, Quickselect does p selection with linear time complexity so $T_1(n, p) = pc_1 n$.
Similarly, $T_2(n, p) = c_2 n \lg n$.
So Quicksort is faster if $T_2(n, p) < T_1(n, p)$ or $c_2 n \lg n < pc_1 n$.
Rearrange the inequality and we get $p > \frac{c_2}{c_1} \lg n$.
Recall where the constants come from.
We can assume $c_1 \approx c_2$.
When $n = 10^6$ and $p = 10$, $10 < \lg n \approx 20$.
So we should sort the list first only if $p > 20$.