Quicksort

Richard Hua

Semester 2, 2021

Introduction

Another recursive divide-and-conquer algorithm.
Recursive calls are determined by the input.
Unlike Mergesort where subarrays are predetermined (even though we compute the actual subarrays at runtime).

Continued

Tony Hoare (1934 -). Turing Award (1980).
Published in 1961.
Originally invented as part of a Russian-English machine translation program while Hoare was working in Moscow.
Won a bet of sixpence.
Worked on one of the earliest Object-Oriented Programming (OOP) language (ALGOL W).

Billion-dollar mistake?

I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years. (T. Hoare, 2009)

Quicksort

Quicksort is based on the following basic idea.

- If the size of the list is 0 or 1, return.
- Otherwise, choose one of the items in the list as a **pivot** and **partition** the list.
- Partition moves the items in the list into two disjoint sublists: all elements less than the pivot are placed in the left sublist and all elements greater than the pivot are placed in the right sublist.
- Recursively process the two sublists.

Continued

A few important things to note:

- Need an efficient partition algorithm.
- Pivot selection is very important. Performance of Quicksort heavily depends on the pivot choice. In general, you want the two sublists to be close in size.
- In any implementation, we must specify what happens to items that has the same key as the pivot.

Pseudo-code

Require: $0 \leqslant i \leqslant j \leqslant n-1$

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

if $i < j$ **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

Efficient partition

Require: $0 \leqslant i \leqslant j \leqslant n-1$

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

if $i < j$ **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

Question

What is the time complexity of PARTITION?

Any approximations on the upper- or lower-bounds?

Efficient partition

Require: $0 \leqslant i \leqslant j \leqslant n-1$

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

if $i < j$ **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

PARTITION has to be at least linear, i.e. $\Omega(n)$, in the worst case. Since each element needs to be checked once to determine whether it's greater or less than the pivot.

Efficient partition

function PARTITION(list $a[0 \cdots n-1]$, integers i, j, k)
    SWAP(a,i,k)
    $p \leftarrow a[i]$
    $l \leftarrow i$
    $r \leftarrow j+1$
    while True **do**
        repeat
            $l \leftarrow l+1$
        until $a[l] \geqslant p$
        repeat
            $r \leftarrow r-1$
        until $a[r] \leqslant p$
        if $l < r$: SWAP(a, l, r)
        else SWAP(a, i, r); **return** r

PARTITION example

Run PARTITION on the following list assuming $k = i$.

4	1	5	6	2	7	3

PARTITION example continued

Run PARTITION on the following list assuming $k = i$.

4	1	5	6	2	7	3

Question

Can the QUICKSORT pseudo-code we had use this PARTITION function?

PARTITION time complexity

Question

What is the time complexity of PARTITION?

PARTITION time complexity

Question

What is the time complexity of PARTITION?

$\Theta(n)$ since we need to traverse the entire list.
What is the exact number of comparison/swap of Hoare's partition algorithm in the best/worst case?

QUICKSORT in action

https://www.youtube.com/watch?v=ywWBy6J5gz8

Time complexity

Let's derive a recurrence relation first.
If the two sublist are always of equal size after partition. The recurrence can be written as $T(n) = 2T(\frac{n-1}{2}) + \Theta(n)$.
Clearly $2T(\frac{n-1}{2}) + \Theta(n) \leqslant 2T(\frac{n}{2}) + \Theta(n)$.
Best case is $\Theta(n \log n)$ again.

Continued

Question

Suppose the left sublist is always of size 0 (e.g. if we always pick the smallest element as the pivot), what is the time complexity of QUICKSORT?

A recurrence formula is of the form $T(n) = T(n-1) + n$, $T(0) = 0$.

Solving it results in the tail sum of $1 + 2 + \cdots n = \frac{n(n+1)}{2} \in \Theta(n^2)$.

QUICKSORT is quadratic in the worst case.

Average case

QUICKSORT is very sensitive to input.
Performance varies a lot between the best and worst case.
We want to know the average case performance.

Average case

Suppose we have a list of size n different keys, what could the size of the left and right sublist be after partition?
Think about in how many places can the pivot end up in after partition.
If pivot is at index i after the partition, then there are i and $n-1-i$ elements in the left and right sublist respectively. And we have to do two recursive calls on these list (i.e. $T(i)$, $T(n-1-i)$).
We need to add up all the possible cases and then divide by the number of cases we have.
Average case recurrence is of the form
$T(n) = \frac{1}{n}\sum_{i=0}^{n-1}(T(i) + T(n-1-i)) + cn = \frac{2}{n}\sum_{i=0}^{n-1} T(i) + cn$,
$T(n) = 0$,assuming each element has an equal chance of being picked as the pivot.

Continued

$T(n) = \frac{2}{n}\sum_{i=0}^{n-1} T(i) + cn.$

$nT(n) = 2\sum_{i=0}^{n-1} T(i) + cn^2.$

If we replace n by $n-1$ we get

$(n-1)T(n-1) = 2\sum_{i=0}^{n-2} T(i) + c(n-1)^2.$

$nT(n) - (n-1)T(n-1) = 2T(n-1) + cn^2 - c(n-1)^2.$

Rearrange it and we get $nT(n) = (n+1)T(n-1) + c(2n-1).$

Divide both sides by $n(n+1)$ and we get

$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{c(2n-1)}{n(n+1)} = \frac{T(n-1)}{n} + \frac{3c}{n+1} - \frac{c}{n}.$

Continued

$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{3c}{n+1} - \frac{c}{n}$. What is $\frac{T(n-1)}{n}$?
$\frac{T(n-1)}{n} = \frac{T(n-2)}{n-1} + \frac{3c}{n} - \frac{c}{n-1}$.
Substitute it back and we get
$\frac{T(n)}{n+1} = \frac{T(n-2)}{n-1} + \frac{3c}{n+1} + \frac{3c}{n} - \frac{c}{n} - \frac{c}{n-1} =$
$\frac{T(n-2)}{n-1} + 3c(\frac{1}{n+1} + \frac{1}{n}) - c(\frac{1}{n} + \frac{1}{n-1})$.
Repeat the same process and we get
$\frac{T(n)}{n+1} = \frac{T(n-3)}{n-2} + 3c(\frac{1}{n+1} + \frac{1}{n} + \frac{1}{n-1}) - c(\frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2})$.
$\frac{T(n)}{n+1} = \frac{T(0)}{1} + 3c\sum_{i=2}^{n+1}\frac{1}{i} - c\sum_{i=1}^{n}\frac{1}{i}$.
The term $\sum_{i=1}^{n} 1/i$ is known as the nth **Harmonic** number, denoted by H_n. $H_n \in \Theta(\log n)$.
$\frac{T(n)}{n+1} = 3c(H_{n+1} - 1) - cH_n = 3c(H_n + \frac{1}{n+1} - 1) - cH_n =$
$2cH_n + \frac{3c}{n+1} - 3c$.
$\frac{T(n)}{n+1} \in \Theta(\log n)$ so $T(n) \in \Theta(n \log n)$.

Pivot selection

We have to be very careful when choosing pivot.
Optimal choice is the median, but this is hard to do in practice.
Two approaches in general:

- Passive pivot strategy.
- Active pivot strategy.

Passive strategies always choosing a fixed index (e.g. first, last or middle) as the pivot.
Not a good idea in general (e.g. large and almost sorted input, algorithmic complexity attack, etc.)

Active strategy

Median-of-three method.
Random selection. Can be used to avoid complexity attack.
Median-of-medians method. Very good approximation. Need $O(n)$ time to compute. Not suited for our purpose.

In-place

Question

Is QUICKSORT in-place?

In-place

Question

Is QUICKSORT in-place?

No, unfortunately. Recursion calls require $\Theta(\log n)$ space. Not much but not constant.

Stable

Question

Is MERGESORT stable?

Obviously nothing gets moved around during the 'split' phase. What about MERGE?

Stable

Question

Is QUICKSORT stable?

Stable

Question

Is QUICKSORT stable?

Exercise

Create an example that shows that QUICKSORT is not stable.

Additional notes

Can we implement QUICKSORT with linked-list?
Technically, yes. But people normally don't do it for a reason...
Often implemented slightly differently in practice.
For example, when n is quite small (e.g. $n < 10$), sorting the list recursively may take more time than using, say, insertion sort.