CMSC351-2023-10-11

Wednesday, October 11, 2023 8:19 AM

Quicksort

① Intro: Quicksort is a fast divide-and-conquer sorting algorithm

② Idea: The idea is we pick a value in the list called the pivot value (pv) then we rearrange the list so all values ≤ pv are to the left of the pv and all values > are to the right of the pv. ⊛
we then do the same thing to the left and right sublists.
At the bottom of the recursion a list of length 1 is sorted.
DONE!
note: pivot value $\underline{\underline{\text{is}}}$ an entry in the list, often called the pivot.
note: ⊛ is called the partitioning process.

③ Partitoning

(A) Intuition: Sps we have a list
Choose the pv (don't worry about how for now)
Find leftmost value > pv
Find first subsequent value ≤ pv.
Swap those
Repeat until done

ex $2, \underline{\underline{5}}, 4, \underline{\underline{1}}, 0, 3$ choosed pv = 3
>pv ≤pv swap!

$2, 1, \underline{\underline{4}}, 5, \underline{\underline{0}}, 3$
>pv ≤pv swap!

$2, 1, 0, \underline{\underline{5}}, 4, \underline{\underline{3}}$ swap
>pv ≤pv

$2, 1, 0, 3, \underline{\underline{4}}, 5$
>pv but no subseq. value ≤ pv. STOP

2, 1, 0, 3, 4, 5

$>$pv w/ no subseq. value $\le$ pv. Stop

$\underbrace{2, 1, 0}_{\le pv}, \boxed{3}, \underbrace{4, 5}_{> pv}$

This is one step of the partitioning done!

(B) <u>Pseudocode</u>

```
fctn partition(A, L, R)
    pv = A[R]
    t = L
    for i = L to R-1
        if A[i] <= pv
            A[t] <-> A[i]
            t++
        end if
    end for
    A[t] <-> A[R]
    return t   <- returns index of where pv. ends up.
end fctn
```

A = List. L, R = start + ending indices

Here pv = rightmost value

<u>Note</u>: b/c there is just one loop w/ constant time inside and before and after, this is Θ(length of list)

(C) Why does the pseudocode do what we discussed?!

What the alg does is:

t is "hunting for" the Leftmost value > pv.

i is looking for the first subsequent value ≤ pv.

note: if t encounters values ≤ pv we will do some useless swapping. That's okay.

<u>ex</u> 2 5 4 1 0 3 partition(A, 0, 5)

t, i ← A[i] ≤ pv ↳ pv = 3

useless swap! t++, i++

2 5 4 1 0 3

t, i ← A[i] ≰ pv

no swap! i++

2 5 4 1 0 3

```
        no swap!   i++
2     5     4     1     0     3
      t     i     ← A[i] ≰ pv
        no swap!   i++
2     5     4     1     0     3
      t           i     ← A[i] ≤ pv
        swap!    t++, i++
2     1     4     5     0     3
            t           i     ← A[i] ≤ pv
        swap!    t++, i++
2     1     0     5     4     3
                  t           i  ← loop ended at i=4 (previous index)
                                    exit for loop.
2     1     0     3     4     5
                  ↑___________↑  swap after loop!

        DONE!
```

(D) Choosing the pivot value

- in theory we could choose any value in the list.
- the pseudocode requires it to be the last value. to use a different value, swap that value w/ the last value right at the start of Partition.
- ideally we'd use the median b/c then, at the end, the pv would essentially be in the middle which is nice for divide and conquer. However - finding the median is not obvious!
- In practice the pv is chosen randomly.

④ Now on to Quicksort!

At this point quicksort is easy!

```
fctn quicksort(A,L,R)                    ← Assm A=global list
    if L<R
        resultingpivotindex = partition(A,L,R)
        quicksort(A,L,resultingpivotindex-1)
        quicksort(A,resultingpivotindex+1,R)
    end
end
```

to start the whole process:

```
quicksort(A, 0, length(A)-1)
```

⑤ Time Complexity! Aux Space? In-Place? Stable