5 Divide & Conquer

11 November 2024

Prof. Dr. Sebastian Wild

CS566 (Wintersemester 2024/25)
Philipps-Universität Marburg

version `2024-10-24 23:46 H`

Learning Outcomes

Unit 5: ***Divide & Conquer***

1. Know the steps of the Divide & Conquer paradigm.
2. Be able to design and analyze new algorithms using the Divide & Conquer paradigm.

Outline

Divide and conquer

Divide and conquer *idiom* (Latin: *divide et impera*)
to make a group of people disagree and fight with one another
so that they will not join together against one (Merriam-Webster Dictionary)

$\rightsquigarrow$ in politics as in algorithms, many independent, small problems are better than a big one!

Divide-and-conquer algorithms:

1. Break problem into smaller, independent subproblems. (Divide!)
2. Recursively solve all subproblems. (Conquer!)
3. Assemble solution for original problem from solutions for subproblems.

Examples:

- Mergesort
- Quicksort
- Binary search
- (arguably) Tower of Hanoi

5.1 Order Statistics

Selection by Rank

- Standard data summary of numerical data: (Data scientists, listen up!)
  - mean, standard deviation
  - min/max (range)
  - histograms

  easy to compute in $\Theta(n)$ time

  - median, quartiles, other quantiles (a.k.a. order statistics)

  computable in $\Theta(n)$ time?

General form of problem: **Selection by Rank**

- **Given:** array $A[0..n)$ of numbers and number $k \in [0..n)$.
- **Goal:** find element that would be in position k if A was sorted (kth smallest element).

  but 0-based & counting dups

- $k = \lfloor n/2 \rfloor \rightsquigarrow$ median; $k = \lfloor n/4 \rfloor \rightsquigarrow$ lower quartile
  $k = 0 \rightsquigarrow$ minimum; $k = n - \ell \rightsquigarrow$ ℓth largest

Quickselect

- Key observation: Finding the element of rank k seems hard.
  But computing the rank of a given element is easy!
  count smaller elements

$\rightsquigarrow$ Pick any element $A[b]$ and find its rank j.

- $j = k$? $\rightsquigarrow$ Lucky Duck! Return chosen element and stop
- $j < k$? $\rightsquigarrow$ ...not done yet. But: The $j + 1$ elements smaller than $\le A[b]$ can be excluded!
- $j > k$? $\rightsquigarrow$ similarly exclude the $n - j$ elements $\ge A[b]$

- partition function from Quicksort:
  - returns the rank of pivot
  - separates elements into smaller/larger

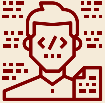
$\rightsquigarrow$ can use same building blocks

(recursion can be replaced by loop)

```
procedure quickselect(A[l..r), k)
    if r − ℓ ≤ 1 then return A[l]
    b := choosePivot(A[l..r))
    j := partition(A[l..r), b)
    if j == k
        return A[j]
    else if j < k
        quickselect(A[j + 1..n), k − j − 1)
    else // j > k
        quickselect(A[0..j), k)
```

Quickselect Discussion

- $\Theta(n^2)$ worst case (like Quicksort)
- can prove: expected cost $\Theta(n)$
- no extra space needed
- adaptations possible to find several order statistics

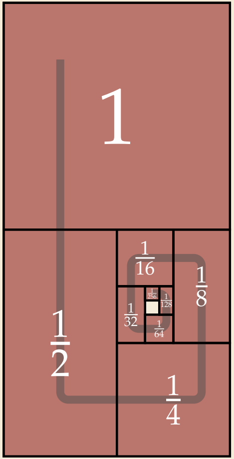

For practical purposes, Quickselect is fine.


Yeah . . . maybe. But can we select by rank in $O(n)$ **worst case**?

Better Pivots

It turns out, we can!

- All we need is better pivots!
  - If pivot was the exact median,
    we would at least halve #elements in each step
  - Then the total cost of all partitioning steps is $\le 2n = \Theta(n)$.



But: finding medians is (basically) our original problem!

It totally suffices to find an element of rank αn for $\alpha \in (\varepsilon, 1-\varepsilon)$
to get overall costs $\Theta(n)$!

The Median-of-Medians Algorithm

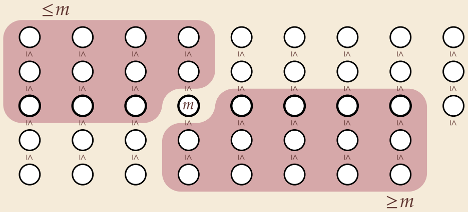
```
procedure choosePivotMoM(A[l..r])
    m := ⌊n/5⌋
    for i := 0, ..., m − 1
        sort(A[5i..5i + 4])
        // collect median of 5
        Swap A[i] and A[5i + 2]
    return quickselectMoM(A[0..m), ⌊(m−1)/2⌋)

procedure quickselectMoM(A[l..r), k)
    if r − ℓ ≤ 1 then return A[l]
    b := choosePivotMoM(A[l..r))
    j := partition(A[l..r), b)
    if j == k
        return A[j]
    else if j < k
        quickselectMoM(A[j + 1..n), k − j − 1)
    else // j > k
        quickselectMoM(A[0..j), k)
```

Analysis:

- Note: 2 mutually recursive procedures
  $\leadsto$ effectively 2 recursive calls!

1. recursive call inside choosePivotMoM on $m \le \frac{n}{5}$ elements
2. recursive call inside quickselectMoM



$\leadsto$ partition excludes $\sim 3 \cdot \frac{m}{2} \sim \frac{3}{10}n$ elem.

$$
\begin{aligned}
\leadsto \quad C(n) &\le \Theta(n) + C(\tfrac{1}{5}n) + C(\tfrac{7}{10}n) \\
&\le \Theta(n) + C(\tfrac{1}{5}n + \tfrac{7}{10}n) \\
&= \Theta(n) + C(\tfrac{9}{10}n) \quad \leadsto \quad C(n) = \Theta(n)
\end{aligned}
$$

ansatz: overall cost linear

5.2 Further D&C Algorithms

Majority

- **Given:** Array $A[0..n)$ of objects
- **Goal:** Check of there is an object x that occurs at $> \frac{n}{2}$ positions in A
  if so, return x

- Naive solution: check each $A[i]$ whether it is a majority $\rightsquigarrow$ $\Theta(n^2)$ time

Can be solved faster using a simple Divide & Conquer approach:

- If A has a majority, that element must also be a majority of at least one half of A.

$\rightsquigarrow$ Can find majority (if it exists) of left half and right half recursively

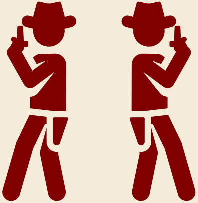
$\rightsquigarrow$ Check these ≤ 2 candidates.

- Costs similar to mergesort $\Theta(n \log n)$

Majority – Linear Time

We can actually do much better!

```
def MJRTY(A[0..n))
    c := 0
    for i := 1, ..., n − 1
        if c == 0
            x := A[i]; c := 1
        else
            if A[i] == x then c := c + 1 else c := c − 1
    return x
```



- MJRTY($A[0..n)$) returns *candidate* majority element
- either that candidate is the majority element or none exists(!)

Clearly $\Theta(n)$ time

Closest pair

- **Given:** Array $P[0..n)$ of points in the plane
  each has x and y coordinates: $P[i].x$ and $P[i].y$
- **Goal:** Find pair $P[i]$, $P[j]$ that is closest in (Euclidean) distance

- Naive solution: compute distance of each pair $\rightsquigarrow$ $\Theta(n^2)$ time
- Can be done in $O(n \log n)$ time using a clever divide & conquer algorithm.
  (Details not part of the module material.)