CSE 421

Divide and Conquer: Integer Multiplication

Shayan Oveis Gharan
Median
Selecting k-th smallest

Problem: Given numbers $x_1, \ldots, x_n$ and an integer $1 \leq k \leq n$ output the $k$-th smallest number
\[
\text{Sel}([x_1, \ldots, x_n], k)
\]

A simple algorithm: Sort the numbers in time $O(n \log n)$ then return the $k$-th smallest in the array.

Can we do better?

Yes, in time $O(n)$ if $k = 1$ or $k = 2$.

Can we do $O(n)$ for all possible values of $k$?

Assume all numbers are distinct for simplicity.
An Idea

Choose a number \( w \) from \( x_1, \ldots, x_n \)

Define
\[
\begin{align*}
S_< (w) &= \{ x_i : x_i < w \} \\
S_\leq (w) &= \{ x_i : x_i = w \} \\
S_> (w) &= \{ x_i : x_i > w \}
\end{align*}
\]

Can be computed in linear time

Solve the problem recursively as follows:
\[
\begin{align*}
&\text{If } k \leq |S_< (w)|, \text{ output } Sel(S_< (w), k) \\
&\text{Else if } k \leq |S_< (w)| + |S_\leq (w)|, \text{ output } w \\
&\text{Else output } Sel(S_> (w), k - |S_< (w)| - |S_\leq (w)|)
\end{align*}
\]

Ideally want \(|S_< (w)|, |S_> (w)| \leq n/2\). In this case ALG runs in \( O(n) + O \left( \frac{n}{2} \right) + O \left( \frac{n}{4} \right) + \cdots + O(1) = O(n). \)
How to choose w?

Suppose we choose w uniformly at random similar to the pivot in quicksort. Then, $\mathbb{E}[|S_<(w)|] = \mathbb{E}[|S_>(w)|] = n/2$. Algorithm runs in $O(n)$ in expectation.

Can we get $O(n)$ running time deterministically?

- Partition numbers into sets of size 3.
- Sort each set (takes $O(n)$)
- $w = \text{Sel(midpoints, } n/6)$
How to lower bound $|S_{<}(w)|$, $|S_{>}(w)|$?

- $|S_{<}(w)| \geq 2 \left( \frac{n}{6} \right) = \frac{n}{3}
- |S_{>}(w)| \geq 2 \left( \frac{n}{6} \right) = \frac{n}{3}.

So, what is the running time?
Asymptotic Running Time?

- If $k \leq |S_<(w)|$, output $Sel(S_<(w), k)$
- Else if $k \leq |S_<(w)| + |S_=(w)|$, output $w$
- Else output $Sel(S_>(w), k - S_<(w) - S_=(w))$

Where $\frac{n}{3} \leq |S_<(w)|, |S_>(w)| \leq \frac{2n}{3}$

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n) \Rightarrow T(n) = O(n \log n)$$
An Improved Idea

Partition into \( \frac{n}{5} \) sets. Sort each set and set \( w = \text{Sel(midpoints, } \frac{n}{10}) \)

- \( |S_{<}(w)| \geq 3 \left( \frac{n}{10} \right) = \frac{3n}{10} \)
- \( |S_{>}(w)| \geq 3 \left( \frac{n}{10} \right) = \frac{3n}{10} \)

\[
T(n) = T \left( \frac{n}{5} \right) + T \left( \frac{7n}{10} \right) + O(n) \Rightarrow T(n) = O(n)
\]
An Improved Idea

Sel(S, k) {
    n ← |S|
    If (n < ??) return ??
    Partition S into n/5 sets of size 5
    Sort each set of size 5 and let M be the set of medians, so
    |M|=n/5
    Let w=Sel(M,n/10)
    For i=1 to n{
        If $x_i < w$ add x to $S_{<}(w)$
        If $x_i > w$ add x to $S_{>}(w)$
        If $x_i = w$ add x to $S_{\leq}(w)$
    }
    If ($k \leq |S_{<}(w)|$)
        return Sel($S_{<}(w),k$)
    else if ($k \leq |S_{<}(w)| + |S_{\geq}(w)|$)
        return w;
    else
        return Sel($S_{>}(w),k - |S_{<}(w)| - |S_{\geq}(w)|$)
}
D&C Summary

Idea:

“Two halves are better than a whole”
  • if the base algorithm has super-linear complexity.

“If a little's good, then more's better”
  • repeat above, recursively

Applications: Many.
  • Binary Search, Merge Sort, (Quicksort),
  • Root of a Function
  • Closest points,
  • Integer multiplication
  • Median
  • Matrix Multiplication
Approximation Algorithms
How to deal with NP-complete Problem

Many of the important problems in real world are NP-complete.
SAT, Set Cover, Graph Coloring, TSP, Max IND Set, Vertex Cover, …

So, we cannot find optimum solutions in polynomial time. What to do instead?

• Find optimum solution of special cases (e.g., random inputs)

• Find near optimum solution in the worst case
Approximation Algorithm

Polynomial-time Algorithms with a guaranteed approximation ratio.

\[ \alpha = \frac{\text{Cost of computed solution}}{\text{Cost of the optimum}} \]

worst case over all instances.

Goal: For each NP-hard problem find an approximation algorithm with the best possible approximation ratio.
Given a graph $G=(V,E)$, Find smallest set of vertices touching every edge
Greedy algorithms are typically used in practice to find a (good) solution to NP-hard problems

**Strategy (1):** Iteratively, include a vertex that covers most new edges

Q: Does this give an optimum solution?
A: No,
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most
Greedy (1): Pick vertex that covers the most

Greedy Vertex cover = 20
OPT Vertex cover = 8
Greedy (1): Pick vertex that covers the most vertices.

Each vertex has one edge into each $B_i$

$|B_i| = n/i$

Greedy pick bottom vertices $= n + \frac{n}{2} + \frac{n}{3} + \cdots + 1 \approx n \ln n$

OPT pick top vertices $= n$
Greedy 2: Iteratively, pick both endpoints of an uncovered edge.

Vertex cover = 6
Greedy 2: Pick Both endpoints of an uncovered edge

Greedy vertex cover = 16
OPT vertex cover = 8
Greedy (2) gives 2-approximation

**Thm:** Size of greedy (2) vertex cover is at most twice as big as size of optimal cover

**Pf:** Suppose Greedy (2) picks endpoints of edges $e_1, \ldots, e_k$. Since these edges do not touch, every valid cover must pick one vertex from each of these edges!

i.e., $OPT \geq k$.

But the size of greedy cover is $2k$. So, Greedy is a 2-approximation.