Prune-and-search

- Divide the problem into several subproblems
- Decide which subproblem contains a solution
- Solve this subproblem recursively

Example: binary search in a sorted array
- does array contain x=4? 0 1 1 3 5 7 8 9 9
- Compute \( p = \left\lfloor \frac{\text{left} + \text{right}}{2} \right\rfloor \)
- If \( x < A[p] \): search recursively \( A[\text{left} .. p-1] \)
- If \( x = A[p] \): done
- If \( x > A[p] \): search recursively \( A[p+1 .. \text{right}] \)

Computing i-th smallest number

- Input:
  - unsorted array of \( n \) numbers \( A[1 .. n] \)
  - fixed integer \( i \in \{1, ..., n\} \)
- Output: i-th smallest number 3 4 5 7 2 9 1 8 1

- Naïve solution:
  - sort array \( A \) - O(\( n \log n \)) time
  - return \( A[i] \) 1 1 2 3 4 5 7 8 9

- Can we do better than O(\( n \log n \))?

Computing i-th smallest number

```c
void Select(int left, int right, int i)
```

- \( \text{left} \leq i \leq \text{right} \)
- Rearrange input array so that the (\( i-\text{left}+1 \))-smallest number is at position \( i \)
  - result = \( A[i] \)
  
  3 4 5 7 2 9 1 8 1

  \( \text{left} \quad i \quad \text{right} \)
Computing i-th smallest number

```
void Select(int left, int right, int i)
q = rSplit(left, right);
if i < q Select(left, q - 1, i);
else if i > q Select(q + 1, right, i);
```

Average complexity

- \( T(n) \) : expected runtime for \( n \) elements
- Assumption: all elements are distinct

\[
T(n) \leq n + \frac{1}{n} \sum_{m=0}^{n-1} \max\{T(m), T(n - m - 1)\}
\]

\[
T(n) \leq n + \frac{2}{n} \sum_{m=\frac{n}{2}}^{n-1} T(m)
\]

if \( n \) is even
Average complexity

Let’s prove \( T(n) \leq cn \) for some constant \( c \)

Use induction!

Base case: \( n = 1 \)

\( T(1) \leq c \cdot 1 \) : true if \( c \) is large enough

Average complexity

Induction step: assume \( T(m) \leq cm \) for all \( m < n \)

\[
T(n) \leq n + \frac{2}{n} \cdot \sum_{m=\frac{n}{2}}^{n-1} T(m)
\]

\[
\leq n + \frac{2c}{n} \cdot \sum_{m=\frac{n}{2}}^{n-1} m
\]

\[
= \left( 1 + \frac{3c}{4} \right) n - \frac{c}{2} \leq cn \quad \text{if } c \geq 4
\]

Computing i-th smallest number

Randomized algorithm:
- Worst-case: \( O(n^2) \)
- Average: \( O(n) \) [assuming distinct elements]

Deterministic algorithm with \( O(n) \) worst-case?
Deterministic algorithm

- Idea: replace randomized split with deterministic split that is guaranteed to give a sufficiently balanced partition

<table>
<thead>
<tr>
<th>rSplit</th>
<th>dSplit</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
</tr>
<tr>
<td><img src="image3" alt="Image" /></td>
<td><img src="image4" alt="Image" /></td>
</tr>
<tr>
<td><img src="image5" alt="Image" /></td>
<td><img src="image6" alt="Image" /></td>
</tr>
<tr>
<td><img src="image7" alt="Image" /></td>
<td><img src="image8" alt="Image" /></td>
</tr>
<tr>
<td><img src="image9" alt="Image" /></td>
<td><img src="image10" alt="Image" /></td>
</tr>
<tr>
<td><img src="image11" alt="Image" /></td>
<td><img src="image12" alt="Image" /></td>
</tr>
<tr>
<td><img src="image13" alt="Image" /></td>
<td><img src="image14" alt="Image" /></td>
</tr>
</tbody>
</table>

Deterministic algorithm

1. Partition \( n \) items into \( \left\lceil \frac{n}{5} \right\rceil \) groups of size at most 5 each
2. Find the median of each group
3. Find the median of medians recursively
4. Split the array using the result as the pivot element
5. Recurse on one side of the pivot

Select(\( left, right, i \))

- set \( k = \left\lceil \frac{right - left + 1}{5} \right\rceil \) and \( m = left + 2k + \left\lceil \frac{k}{2} \right\rceil \)
- for \( j = 0..k - 1 \) call Sort(\( left + j, k, right \))
- call Select(\( left + 2k, left + 3k - 1, m \))
- swap \( A[left] \) and \( A[m] \), call \( q = \text{Split}(left, right) \)
- if \( q = i \) return, otherwise call Select(\( left', right', i \)) with appropriate arguments

![Image](image15)
How balanced is the split?

• At least $\frac{3n}{10}$ items are smaller than pivot
• At least $\frac{3n}{10}$ items are greater than pivot

$$T(n) \leq n + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right)$$

Worst-case complexity

• Let’s prove $T(n) \leq cn$ for some constant $c$
• Induction step: assume $T(m) \leq cm$ for all $m < n$

$$T(n) \leq n + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right)$$

$$\leq n + c \cdot \frac{n}{5} + c \cdot \frac{7n}{10}$$

$$= \left(1 + \frac{9c}{10}\right)n \leq cn \quad \text{if } c \geq 9$$

Prune-and-search approach: Summary

• Very similar to divide-and-conquer
  - Recursion takes some fraction $\alpha < 1$ of the input

$$T(n) \leq S(n) + T(\alpha \cdot n) \quad \Rightarrow \quad T(n) = O(S(n))$$
Randomized algorithms

- Randomized vs. deterministic algorithm:
  - Often much simpler & faster, but requires a source of randomness
  - Can be slow (with low probability)
    - can be fixed - see introsort / introselect
  - Repeating algorithm for the same input:
    - deterministic: always same steps
    - randomized: steps may be different
  - Complexity:
    - deterministic: worst-case
    - randomized: average case

- Las Vegas algorithm:
  - always correct answer, runtime may vary

- Monte Carlo algorithm:
  - answer may be incorrect [with small prob.], deterministic runtime

Asymptotic growth

\[ f(n) = O(g(n)) : \exists c > 0, N \text{ s.t. } f(n) \leq c \cdot g(n) \quad \forall n > N \]

\[ f(n) = \Theta(g(n)) : \exists c_1, c_2 > 0, N \text{ s.t. } c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \quad \forall n > N \]

\[ f(n) = \Omega(g(n)) : \exists c > 0, N \text{ s.t. } f(n) \geq c \cdot g(n) \quad \forall n > N \]

Asymptotic growth

- Algorithm’s complexity is \( O(n^2) \): ...
- Algorithm’s worst-case complexity is \( O(n^2) \): ...
- Algorithm’s complexity is \( \Theta(n^2) \): ...
- Algorithm’s worst-case complexity is \( \Theta(n^2) \): ...
Asymptotic growth

- $\Theta(n^2)$ alg. may be faster than $\Theta(n \log n)$ for small inputs... but eventually gets slower

- “Median of medians” algorithm:
  - for small sizes, switch to naive solution (sorting)
  - sorting 5 elements: use insertion sort ($O(n^2)$)

Insertion sort

- First sort $A[left..left+0]$
- Then sort $A[left..left+1]$
- Then sort $A[left..left+2]$
- ...

- Worst-case complexity: $O(n^2)$
- But: quite efficient
  - for small arrays
  - already (almost) sorted arrays