COMP2123/2823/9123
Data structures and Algorithms

Lecture 11: Divide and Conquer

[GT 11]

Dr. Julian Mestre
School of Computer Science

Some content is taken from material provided by the textbook publisher Wiley.

THE UNIVERSITY OF SYDNEY

Divide and Conquer

Divide and Conquer algorithms can normally be broken into these three parts:

1. Divide If it is a base case, solve directly, otherwise break up the problem into several parts.

2. Recur Recursively solve each part [each sub-problem].

3. Conquer Combine the solutions of each part into the overall solution.

Integer multiplication

Given two n-digit integers x and y

Problem compute the product x y

While this seems like recreational mathematics, it does have real applications: Public key encryption is based on manipulating integers with thousands of bits.

Integer multiplication: Naïve approach

Given two n-digit integers x and y

Problem compute the product $x\ y$

Suppose we wanted to do it by hand. We assume that two digits can be multiplied or added in constant time

In primary school we all learn an algorithm for this problem that performs $\Theta(n^2)$ operations

Integer multiplication: Divide and conquer

Let $x = x_1 \, 2^{n/2} + x_0$ and $y = y_1 \, 2^{n/2} + y_0$

Then $x \, y = x_1 \, y_1 \, 2^n + x_1 \, y_0 \, 2^{n/2} + x_0 \, y_1 \, 2^{n/2} + x_0 \, y_0$

We can compute the product of two n-digit numbers by making 4 recursive calls on $n/2$-digit numbers and then combining the solutions to the subproblems.

Integer multiplication: Divide and conquer

```
def multiply(x, y):
  // x and y are positive integers represented in binary
  if x == 0 or y == 0 then return 0
  if x == 1 then return y
  if y == 1 then return x

  // recursive case
  let x1 and x0 be such that x = x1 2^(n/2) + x0
  let y1 and y0 be such that y = y1 2^(n/2) + y0

  return multiply(x1, y1) 2^n +
         (multiply(x1, y0) + multiply(x0, y1)) 2^(n/2) +
         multiply(x0, y0)
```

Integer multiplication: Correctness

Let $x = x_1 \, 2^{n/2} + x_0$ and $y = y_1 \, 2^{n/2} + y_0$

Then $x \, y = x_1 \, y_1 \, 2^n + x_1 \, y_0 \, 2^{n/2} + x_0 \, y_1 \, 2^{n/2} + x_0 \, y_0$

Straight forward application of induction to prove that `multiply(x, y)` = x y

Integer multiplication: Divide and conquer v2.0

Let $x = x_1 \, 2^{n/2} + x_0$ and $y = y_1 \, 2^{n/2} + y_0$

$$x \, y = x_1 \, y_1 \, 2^n + (x_1 \, y_0 + x_0 \, y_1) \, 2^{n/2} + x_0 \, y_0$$

$$(x_1 + x_0)\,(y_1 + y_0) = x_1 \, y_1 + x_1 \, y_0 + x_0 \, y_1 + x_0 \, y_0$$

We can compute the product of two n-digit numbers by making 3 recursive calls on $n/2$-digit numbers and then combining the solutions to the subproblems.

Integer multiplication: Divide and conquer

```
def multiply(x, y):
  // base case
  ⋮
  // recursive case
  let x₁ and x₀ be such that x = x₁ 2^(n/2) + x₀
  let y₁ and y₀ be such that y = y₁ 2^(n/2) + y₀

  first_term ← multiply(x₁, y₁)
  last_term ← multiply(x₀, y₀)
  other_term ← multiply(x₁ + x₀, y₁ + y₀)

  return first_term 2^n +
         (other_term - first_term - last_term) 2^(n/2) +
         last_term
```

Integer multiplication: Complexity analysis

Recall $x\,y = x_1\,y_1\,2^n + (x_1\,y_0 + x_0\,y_1)\,2^{n/2} + x_0\,y_0$

Divide step (produce halves) takes $O(n)$

Recur step (solve subproblems) takes $3\,T(n/2)$

Conquer step (add up results) takes $O(n)$

$$T(n) = \begin{cases} 3\,T(n/2) + O(n) & \text{for } n > 1 \\ O(1) & \text{for } n = 1 \end{cases}$$

This solves to $T(n) = O(n^2)$. No better than naïve!!!

Integer multiplication: Complexity analysis

Divide step (produce halves) takes $O(n)$

Recur step (solve subproblems) takes $3\ T(n/2)$

Conquer step (add up results) takes $O(n)$

$$T(n) = \begin{cases} 3\ T(n/2) + O(n) & \text{for } n > 1 \\ O(1) & \text{for } n = 1 \end{cases}$$

This solves to $T(n) = O(n^{\log_2 3})$, where $\log_2 3 \approx 1.6$

Better than naïve!!!

Geometric series facts

Let r be a positive real and k a positive integer then

$$1 + r + r^2 + \ldots + r^k = (r^{k+1} - 1)/(r-1)$$

Consequently if $r > 1$ then

$$1 + r + r^2 + \ldots + r^k < r^{k+1} / (r-1)$$

and if $r < 1$ then

$$1 + r + r^2 + \ldots + r^k < 1 / (1-r)$$

Logarithms facts

Base exchange rule:

$$\log_a x = (\log_b x)/(\log_b a)$$

Product rule:

$$\log_a (xy) = (\log_a x) + (\log_a y)$$

Power rule:

$$\log_a x^b = b \log_a x$$

Master Theorem

Let f(n) and T(n) be defined as follows:

$$T(n) = \begin{cases} a\,T(n/b) + f(n) & \text{for } n > 1 \\ c & \text{for } n < d \end{cases}$$

Depending on a, b and $f(n)$ the recurrence solves to:

1. if $f(n) = O(n^{\log_b a - \varepsilon})$ for $\varepsilon > 0$ then $T(n) = \Theta(n^{\log_b a})$,
2. if $f(n) = \Theta(n^{\log_b a} \log^k n)$ for $k > 0$ then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$,
3. if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ and $af(n/b) \leq \delta$ for $\varepsilon > 0$ and $\delta > 0$ then $T(n) = \Theta(f(n))$,

Note: You should be able to solve all recurrences in this class using unrolling, but if you are comfortable using the Master Theorem, go for it.

Selection

Given an unsorted array A holding n numbers and an integer k, find the kth smallest number in A

Trivial solution: Sort the elements and return kth element

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

Yes, with divide and conquer!

First attempt

1. Divide find the median and split array on the halves,
   $\leq$ and $>$ than the median
2. Recur if $k \leq n/2$ find kth element on smaller half
   if $k > n/2$ find $(k-n/2)$th element on larger half
3. Conquer return value of the recursive call

1	12	5	16	19	7	23	6	13	20

$k = 6$
$n = 10$

1	12	5	7	6

12

16	19	23	13	20

Divide

16	19	23	13	20

Recur

$k = 1$

13

Conquer

Selection time complexity

Divide step (find median and split) takes at least O(n)

Recur step (solve left or right subproblem) takes T(n/2)

Conquer step (return recursive result) takes O(1)

If we could compute the median in O(n) time then:

$$T(n) = \begin{cases} T(n/2) + O(n) & \text{for } n > 1 \\ O(1) & \text{for } n = 1 \end{cases}$$

This solves to $T(n) = O(n)$ but only if we can solve the median problem, which is in fact a special case of selection with $k=n/2$

Second attempt: Approximating the median

We don't need the exact median. Suppose we could find in $O(n)$ time an element x in A such that

$$|A| / 3 \leq \mathrm{rank}(A, x) \leq 2 |A| / 3$$

Then we get the recurrence

$$T(n) = \begin{cases} T(2n/3) + O(n) & \text{for } n > 1 \\ O(1) & \text{for } n = 1 \end{cases}$$

Which again solves to $T(n) = O(n)$

To approximate the median we can use a recursive call!

Median of 3 medians

Consider the following procedure

- Partition A into $|A| / 3$ groups of 3
- For each group find the median
- Let x be the median of the medians

We claim that x has the desired property

$$|A| / 3 \leq \text{rank}(A, x) \leq 2|A| / 3$$

Half of the groups have a median that is smaller/larger than x, and each group has two elements smaller/larger than x, thus

$$\#\text{ elements smaller than } x > 2\,(|A| / 6) = |A| / 3$$
$$\#\text{ elements greater than } x > 2\,(|A| / 6) = |A| / 3$$

Median of 3 medians

Let x be the median of the medians, then

$$|A| \ / \ 3 \leq \text{rank}(A, x) \leq 2|A| \ / \ 3$$

1	12	5	16	19	7	23	6	13

1	12	5		16	19	7		23	6	13
1	5	12		7	16	19		6	13	23
1	5	12		6	13	23		7	16	19

\# elements smaller than $x > 2\,(|A| \ / \ 6) = |A| \ / \ 3$

\# elements greater than $x > 2\,(|A| \ / \ 6) = |A| \ / \ 3$

Median of 3 median time complexity

We don't need the exact median. With a recursive call on $n/3$ elements, we can find x in A such that

$$|A|/3 < \text{rank}(A, x) < 2|A|/3$$

Then we get the recurrence

$$T(n) = T(2n / 3) + T(n / 3) + O(n)$$

Which solves to $T(n) = O(n \log n)$

No better than sorting!

Median of 5 medians

We don't need the exact median. With a recursive call on $n/5$ elements, we can find x in A such that

$$3|A|/10 < \mathrm{rank}(A, x) < 7|A|/10$$

Then we get the recurrence

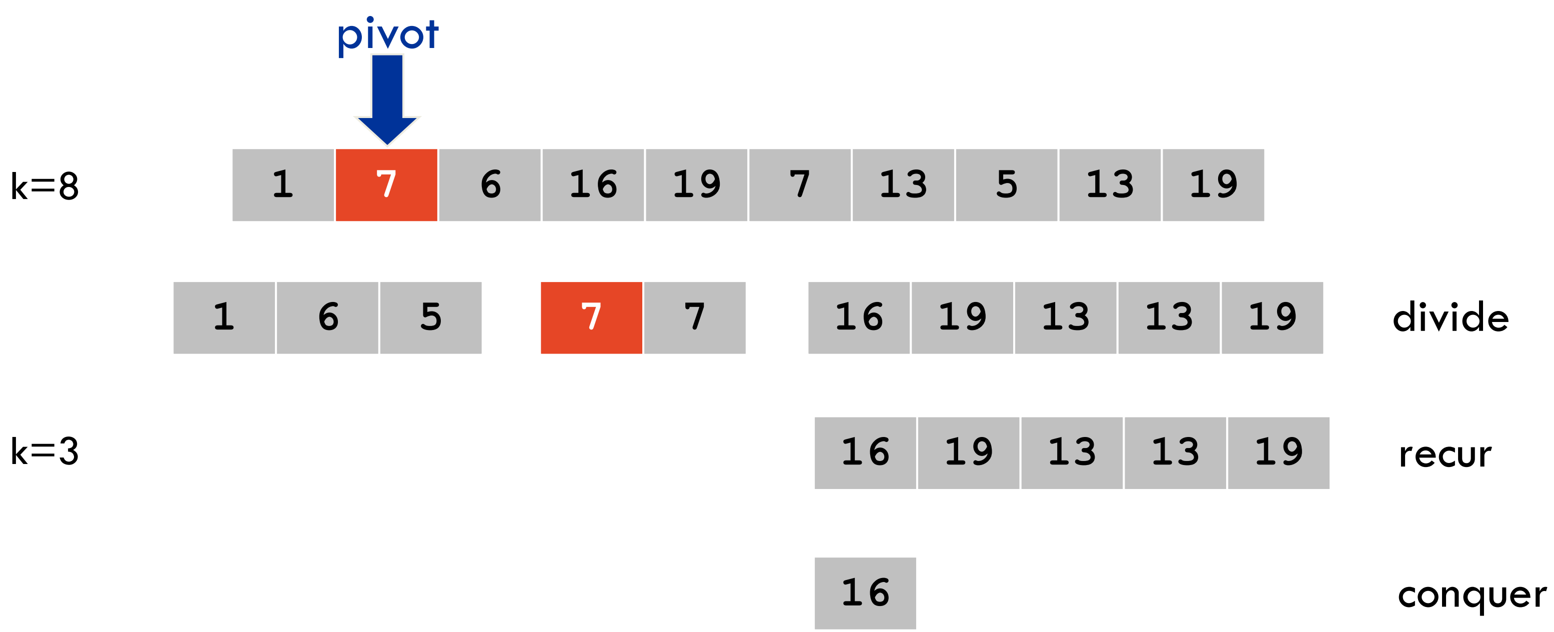
$$T(n) = T(7n / 10) + T(n / 5) + O(n)$$

Which solves to $T(n) = O(n)$

Asymptotically faster than sorting!

Quick selection

1. Divide Choose a random element from the list as the pivot
   Partition the elements into 3 lists:
   (i) less than, (ii) equal to and (iii) greater than the pivot
2. Recur Recursively select right element from correct list
3. Conquer Return solution to recursive problem



Quick selection complexity analysis

Divide step (pick pivot and split) takes $O(n)$

Recur step (solve left and right subproblem) takes $T(n')$

Conquer step (merge subarrays) takes $O(n)$

Now we can set up the recurrence for $T(n)$:

$$E[T(n)] = \begin{cases} E[T(n')] + O(n) & \text{for } n > 1 \\ O(1) & \text{for } n = 1 \end{cases}$$

This solves to $E[T(n)] = O(n)$

(details available on the textbook but not examinable)