

## Divide and Conquer

Chapter 5, Algorithm Design

Hantao Zhang  
<http://www.cs.uiowa.edu/~hzhang/c31/>

---

---

---

---

---

---

---

---

## Divide and Conquer

- ♦ **Divide** the problem into sub-problems that are similar to the original but smaller in size
  - ♦ Subproblems are about the same size: **divide equally**
  - ♦ One subproblem has one less in size: **divide one less**
- ♦ **Conquer** the sub-problems by solving them **recursively**. If they are small enough, just solve them in a straightforward manner.
- ♦ **Combine** the solutions to create a solution to the original problem

dc - 2

---

---

---

---

---

---

---

---

## Recurrence Relation

- ♦ (5.4) For any function  $T()$  satisfying
$$T(n) \leq qT(n/2) + cn$$
where  $q > 2$ ,  $n > 2$ , and  $T(2) \leq c$  (a constant), we have
$$T(n) = O(n^{\log_2 q})$$

dc - 3

---

---

---

---

---

---

---

---

## Ten Largest Known Primes

rank	prime	digits	when
<u>1</u>	$2^{32582657}-1$	<a href="#">9808358</a>	2006
<u>2</u>	$2^{30402457}-1$	<a href="#">9152052</a>	2005
<u>3</u>	$2^{25964951}-1$	<a href="#">7816230</a>	2005
<u>4</u>	$2^{24036583}-1$	<a href="#">7235733</a>	2004
<u>5</u>	$2^{20996011}-1$	<a href="#">6320430</a>	2003
<u>6</u>	$2^{13466917}-1$	<a href="#">4053946</a>	2001
<u>7</u>	$19249 \cdot 2^{13018586}+1$	3918990	2007
<u>8</u>	$27653 \cdot 2^{9167433}+1$	2759677	2005
<u>9</u>	$28433 \cdot 2^{7830457}+1$	2357207	2004
<u>10</u>	$33661 \cdot 2^{7031232}+1$	2116617	2007

dc - 4

---

---

---

---

---

---

---

---

---

---

## Large Integer Multiplication 1

- ◆ Represent each large integer by a list of 32-bit unsigned integers,  $x = [a_{n-1}, a_{n-2}, \dots, a_0]$ , where
$$\text{val}(x) = a_{n-1}2^{32(n-1)} + a_{n-2}2^{32(n-2)} + \dots + a_0,$$
- ◆ Let  $x = [a_{n-1}, a_{n-2}, \dots, a_0]$ ,  $y = [b_{n-1}, b_{n-2}, \dots, b_0]$
- ◆ Compute  $z = x*y$  as normal multiplication in base- $2^{32}$
- ◆ Result:  $z$  is a list of  $2n$  32-bit unsigned integers.
- ◆ Computing time:  $O(n^2)$ .

dc - 5

---

---

---

---

---

---

---

---

---

---

## Large Integer Multiplication 2

- ◆ Divide  $x$  and  $y$  into two lists of  $n/2$  integers:
$$x = x_1 \cdot x_0 \text{ and } y = y_1 \cdot y_0$$
- ◆ Compute  $z_1 = x_1 * y_1$ ,  $v = x_1 * y_0$ ,  $u = x_0 * y_1$ , and  $z_0 = x_0 * y_0$ , recursively.
- ◆ Result:  $z = (z_1) \cdot (u + v) \cdot (z_0)$
- ◆ Computing time:
  - ◆ Let  $T(n)$  be the time of computing  $x*y$ , where  $x$  and  $y$  are lists of  $n$  integers.
  - ◆  $T(n) = 4T(n/2) + cn$

dc - 6

---

---

---

---

---

---

---

---

---

---

## Large Integer Multiplication 3

- ◆ Divide  $x$  and  $y$  into two lists of  $n/2$  integers:  
 $x = x_1 \cdot x_0$  and  $y = y_1 \cdot y_0$
- ◆ Compute  $z_1 = x_1 * y_1$ ,  $p = (x_1 + x_0) * (y_1 + y_0)$ , and  $z_0 = x_0 * y_0$ , recursively.
- ◆ Result:  $z = (z_1) \cdot (p - z_1 - z_0) \cdot (z_0)$
- ◆ Computing time:
  - ◆ Let  $T(n)$  be the time of computing  $x * y$ , where  $x$  and  $y$  are lists of  $n$  integers.
  - ◆  $T(n) = 3T(n/2) + cn$

dc - 7

---

---

---

---

---

---

---

---

## The Master Method

- ◆ The “Cookbook” approach for solving recurrences of the form  
 $T(n) = aT(n/b) + f(n)$ 
  - ◆  $a \geq 1, b > 1$  are constants.
  - ◆  $f(n)$  is asymptotically positive.
  - ◆  $n/b$  may not be an integer, but we ignore floors and ceilings.
- ◆ Requires memorization of three cases.

dc - 8

---

---

---

---

---

---

---

---

## The Master Theorem

- Let  $a \geq 1$  and  $b > 1$  be constants, let  $f(n)$  be a function, and Let  $T(n)$  be defined on nonnegative integers by the recurrence  $T(n) = aT(n/b) + f(n)$ , where we can replace  $n/b$  by  $\lfloor n/b \rfloor$  or  $\lceil n/b \rceil$ .  $T(n)$  can be bounded asymptotically in three cases:
1. If  $f(n) = O(n^{\log_b a - \epsilon})$  for some constant  $\epsilon > 0$ , then  $T(n) = \Theta(n^{\log_b a})$ .
  2. If  $f(n) = \Theta(n^{\log_b a})$ , then  $T(n) = \Theta(n^{\log_b a} \lg n)$ .
  3. If  $f(n) = \Omega(n^{\log_b a + \epsilon})$  for some constant  $\epsilon > 0$ , and if, for some constant  $c < 1$  and all sufficiently large  $n$ , we have  $a \cdot f(n/b) \leq c f(n)$ , then  $T(n) = \Theta(f(n))$ .

dc - 9

---

---

---

---

---

---

---

---

## Divide one less: 1 and $n - 1$

- ♦ **Divide:** Divide the data of size  $n$  into 1 and  $n - 1$
- ♦ **Conquer:** Solve the sub-problem of size  $n - 1$ .
- ♦ **Combine:** Compute the global solution from the remaining element and the solution of the sub-problem.

dc - 10

---

---

---

---

---

---

---

---

## Divide one less: Insertion Sort

**Sorting Problem:** Sort a sequence of  $n$  elements into non-decreasing order.

- ♦ **Divide:** Divide the  $n$ -element sequence to be sorted into one subsequence of  $n-1$  elements
- ♦ **Conquer:** Sort the subsequence recursively using insertion sort.
- ♦ **Combine:** Insert the remaining element into the sorted subsequence to produce the sorted answer.

dc - 11

---

---

---

---

---

---

---

---

## Divide one less: Selection Sort

**Sorting Problem:** Sort a sequence of  $n$  elements into non-decreasing order.

- ♦ **Divide:** Divide the  $n$ -element sequence to be sorted into one subsequence of  $n-1$  elements by removing the maximal element from the input.
- ♦ **Conquer:** Sort the subsequence recursively using selection sort.
- ♦ **Combine:** Put the maximal element at the end of the sorted subsequence to produce the sorted answer.

dc - 12

---

---

---

---

---

---

---

---

## Example: Maximum Subvector

Given an array  $A[1..n]$  of numeric values (can be positive, zero, and negative) determine the subvector  $A[i..j]$  ( $1 \leq i \leq j \leq n$ ) whose sum of elements is maximum over all subvectors.

- ♦ **Divide:** Divide  $A[1..n]$  into  $A[1..n-1]$  and  $A[n]$ .
- ♦ **Conquer:** Solve  $A[1..n-1]$  recursively using the same method and save the result in *maxsum*.
- ♦ **Combine:** Compute the sums of all the subvectors containing  $A[n]$ , and compare it with *maxsum*.

dc - 13

---

---

---

---

---

---

---

---

## Example: Maximum Subvector 1

```
MaxSubvector(A, n)
  if n = 1 return max(0, A[1])
  maxsum ← MaxSubvector(A, n-1)
  sum ← 0
  for k ← n downto 1
    do sum += A[k]
    maxsum ← max(sum, maxsum)
  return maxsum
```

- ♦  $T(n) = T(n - 1) + n$

dc - 14

---

---

---

---

---

---

---

---

## Example: Maximum Subvector 2

```
MaxSubvector(A, n)
  // return (maxsum, maxtail), where maxtail is
  // the maxsum of subvectors
  // containing the last element.
  if (n = 1)
    if (A[1] > 0) return (0, A[1])
    else return (0, 0)
  (maxsum, maxtail) ← MaxSubvector(A, n-1)
  maxtail += A[n]
  if (maxtail > maxsum) return (maxtail, maxtail)
  if (maxtail > 0) return (maxsum, maxtail)
  return (maxsum, 0)
```

- ♦  $T(n) = T(n - 1) + 1$

dc - 15

---

---

---

---

---

---

---

---