Lecutre 3 Divide and conquer

By Gou Guanglei

1. Introduction

- The *divide and conquer* strategy solves a problem by:
  - Breaking it into *subproblems* that are themselves smaller instances of the same type of problem
  - **Recursively** solving these subproblems
  - Appropriately combining their answers

2 Binary search

- Find an element in a sorted array
  - *Divide:* Check middle element.
  - *Conquer:* Recursively search 1 subarray.
  - *Combine:* Trivial.

```
BinarySearch(key, A[], low, high)
  mid = (low + high)/2
  if A[mid]=key
    return true;
  else if A[mid] > key
    BinarySearch(key, A[], low, mid-1)
  else
    BinarySearch(key, A[], mid+1, high)
  return -1;
```

- Performance
  - $T(n) = 1T(n/2) + \Theta(1)$
  - $T(n) = \Theta(logn)$

3. Powering a number

- Problem : Compute a^n, where $n \in N$
- Naive algorithm: $\Theta(n)$

```
NaivePower(a, n)
  x = a;
  for i=2 to n
    x = x * a;
  return x
```

- Divide and conquer algorithm

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if n is even} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if n is odd} \end{cases}$$

```
FastPower(a, n)
  if n = 1
    return a
  else
    x = FastPower(a, n/2)
    if n is even
      return x * x
    else
      return x * x *a
```

- $T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(logn)$

4. Fibonacci numbers

- Recursive definition

$$F_n = \begin{cases} 0 & \text{n=0} \\ 1 & \text{n=1} \\ F_{n-1} + F_{n-2} & \text{n>=2} \end{cases}$$

- Naive recursive algorithm $\Omega(\phi^n)$, where $\phi = (1+\sqrt{5})/2$ is the **golden ratio**.
- Bottom-up algorithm
  - Compute $F_0, F_1, F_2, \ldots, F_n$ in order, forming each number by summing the two previous
  - $T(n) = \Theta(n)$.
- Theorem. (Recursive squaring algorithm)

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

  - Proof.
    - Base case: $n = 1$

$$\begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1$$

    - Inductive step: $n \geq 2$

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

  - $T(n) = \Theta(logn)$

5. Multiplication

- Naive algorithm: Add one itself to another times. $\Theta(2^n)$ addtions
- Divide and conquer algorithm
  - Suppose $x \times y$. Let x and y be base 2.
  - Split each of them into their left and right halves
    - $x = 2^{n/2}x_L + x_R$
    - $y = 2^{n/2}y_L + y_R$
    - $x \times y = (2^{n/2}x_L + x_R) \times (2^{n/2}y_L + y_R) = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$
    - The prblem $x \times y$ divide into **4** subproblems $x_L y_L, x_L y_R, x_R y_L, x_R y_R$.
  - $T(n) = 4T(n/2) + O(n) \Rightarrow T(n) = \Theta(n^2)$
  - can we do better?
- $x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$
- Therefore, the prblem $x \times y$ divide into **3** subproblems $x_L y_L, (x_L + x_R)(y_L + y_R), x_R y_R$.
- $T(n) = 3T(n/2) + O(n) \Rightarrow T(n) = O(n^{log_2 3})$

```
multiply(x, y)
  n = max(size of x, size of y)

  if n = 1
    return xy

  xL, xR = leftmost(n/2+1), rightmost(n/2) bits of x
  yL, yR = leftmost(n/2+1), rightmost(n/2) bits of y

  P1 = multiply(xL, yL)
  P2 = multiply(xR, yR)
  P3 = multiply(xL+xR, yL+yR)

  return P1*2^n + (P3 - P1 -P2)*2^{n/2} + P2
```

6. Matrix multiplication

- input: $A = [a_{ij}], B = [b_{ij}]$; output $C = [c_{ij}] = A \cdot B$
- Naive algorithm for matrix multiplication $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$

```
for i=1 to n
  for j=1 to n
    c_ij = 0
      for k=1 to n
        c_ij = c_ij + a_ik * b_kj
```

- $T(n) = \Theta(n^3)$
- Divide and conquer algorithm

- Idea: $n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices.

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$C \quad = \quad A \quad \cdot \quad B$$

- The problem $A \times B$ divide into 8 multiply subproblems.
  - $r = ae + bg$, $s = af + bh$, $t = ce + dg$, $u = cf + dh$
- $T(n) = 8T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^3)$

- No better than the ordinary algorithm. Can we do better ?
- **Strassen' algorithm**
  - The problem $A \times B$ divide into 7 multiply subproblems.
    - $P_1 = a(f-h), P_2 = (a+b)h, P_3 = (c+d)e, P_4 = d(g-e),$
    - $P_5 = (a+b)(e+h), P_6 = (b-d)(g+h), P_7 = (a-c)(e+f)$
    - $r = P_5 + P_4 - P_2 + P_6, s = P_1 + P_2, t = P_3 + P_4, u = P_5 + P_1 - P_3 - P_7$
  - $T(n) = 7T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^{log_2 7}) \approx \Theta(n^{2.81})$

```
Strassen(A, B)
  if A is 1*1 then
    return A*B
  P_1 = Strassen(a,(f-h))
  P_2 = Strassen((a+b), h)
  P_3 = Strassen((c+d), e)
  P_4 = Strassen(d, (g-e))
  P_5 = Strassen((a+b), (e+h))
  P_6 = Strassen((b-d), (g+h))
  P_7 = Strassen((a-c), (e+f))

  return r=P_5+P_4-P_2+P_6, s=P_1+P_2, t=P_3+P_4, u=P_5+P_1-P_3-P_7
```

7. Conclusion

- Divide and conquer is just one of several powerful techniques for algorithm design.
- Divide-and-conquer algorithms can be analyzed using recurrences and the master method (so practice this math).
- The divide-and-conquer strategy often leads to efficient algorithms.