Divide And Conqueror

Topics Covered

Divide and Conqueror approach

Linear Search

Binary Search

Naive Divide and Conqueror Algorithm

Karatsuba approach

Divide And Conquer

Date
Page

- Break into non-overlapping subproblems of the same type
- Solve subproblems
- Combine results

⇒ (Linear Search)

Input: An array A with n elements. A key k

Output: An index, i, where A[i] = k. If there is no such i, then NOT_FOUND

LinearSearch (A, low, high, key)

Recursive Solution:

```
if high < low:
    return NOT_FOUND
if A[low] = key:
    return low
return LinearSearch (A, low+1, high, key)
```

Recursion defining worst case time

$T(n) = T(n-1) + c$

```
LinearSearch (A, low, high, key)
For i from low to high:
    if A[i] = key:
        return i
return NOT_FOUND
```

⇒ (Binary Search)

Input: A sorted array A[low ---- high]

$(\forall\ low \le i < high : A[i] \le A[i+1])$.

A key k.

Output: An index, i, $(low \le i \le high)$ where A[i] = k

otherwise, the greatest index i, where A[i] < k.

otherwise ($k < A[low]$), the result is low - 1

```
BinarySearch(A, low, high, key)
if high < low:
    return low - 1
mid ← [low + (high - low)/2]
if key = A[mid]
    return mid
else if key < A[mid]:
    return BinarySearch(A, low, mid - 1, key)
else:
    return BinarySearch(A, mid + 1, high, key)
```

Runtime:

$T_n = T(\lceil \frac{n}{2} \rceil) + c$

$T(0) = c$

* Runtime of Binary Search

$\sum_{i=0}^{\log_2 n} c = \Theta(\log_2 n)$

Iteration Version (BinarySearch(A, low, high, key))

```
while low ≤ high
    mid ← [low + (high - low)/2]
    if key = A[mid]:
        return mid
    else if key < A[mid]:
        high = mid - 1
    else
        low = mid + 1
return low - 1
```

Use of multiplying Polynomial

ex $A(x) = 3x^2 + 2x + 5$

$B(x) = 5x^2 + x + 2$

$A(x) B(x) = 15x^4 + 13x^3 + 33x^2 + 9x + 10$

Input: Two $n-1$ degree polynomials

$a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_1 x + a_0$

$b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \dots + b_1 x + b_0$

Output: The product polynomial

$c_{2n-2}x^{2n-2} + c_{2n-3}x^{2n-3} + \dots + c_1 x + c_0$

where:

$c_{2n-2} = a_{n-1} b_{n-1}$

$c_{2n-3} = a_{n-1} b_{n-2} + a_{n-2} b_{n-1}$

Example

Input: $n=3$, $A=[3,2,5]$ $B=(5,1,2)$

$C=[15,13,33,9,10]$

Naive Algorithm MultPoly(A,B,n)

product ← Array[2n-1]

for i from 0 to 2n-2:

product[i] ← 0

for i from 0 to n-1:

for j from 0 to n-1:

product[i+j] ← product[i+j] + A[i] * B[j]

return product

Runtime ⇒ $O(n^2)$

* Naive devide and Conquer Algorithm

e.g. $A(x) = 4x^3 + 3x^2 + 2x + 1$

$B(x) = x^3 + 2x^2 + 3x + 4$

$D_1(x) = 4x + 3$ $D_0(x) = 2x + 1$

$E_1(x) = x + 2$ $E_0(x) = 3x + 4$

$D_1E_1 = 4x^2 + 11x + 6$ $D_1E_0 = 12x^2 + 25x + 12$

$D_0E_1 = 2x^2 + 5x + 2$ $D_0E_0 = 6x^2 + 11x + 4$

$AB = (D_1E_1)x^4 + (D_1E_0 + D_0E_1)x^2 + D_0E_0$

$\Rightarrow 4x^6 + 11x^5 + 20x^4 + 30x^3 + 20x^2 + 11x + 4$

Pseudocode: Function Mult2(A, B, n, a_1, b_1)

a_1 → coefficient of first a

R = array[0 ... 2n-1]

if n = 1

R[0] = R[a_1] * B[b_1]; return R

R[0 ... n-2] = Mult2(A, B, n/2, a_1, b_1)

R[n ... 2n-2] = Mult2(A, B, n/2, a_1 + n/2, b_1 + n/2)

D_0E_1 = Mult2(A, B, n/2, a_1, b_1 + n/2)

D_1E_0 = Mult2(A, B, n/2, a_1 + n/2, b_1)

R[n/2 ... n + n/2 - 2] += $D_1E_0 + D_0E_1$

return R

Time: $\log_2 n$

Runtime same as Naive ⇒ $O(n^2)$

Karatsuba approach

$A(x) = a_1 x + a_0$

$B(x) = b_1 x + b_0$

$C(x) = a_1 b_1 x^2 + (a_1 b_0 + a_0 b_1) x + a_0 b_0$

Needs 4 multiplication

Rewrite as:

$C(x) = a_1 b_1 x^2 + ((a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0) x + a_0 b_0$

3 multiplication

Ex $A(x) = 4x^3 + 3x^2 + 2x + 1$

$B(x) = x^3 + 2x^2 + 3x + 4$

$D_1(x) = 4x + 3$ $\quad D_0(x) = 2x + 1$

$E_1(x) = x + 2$ $\quad E_0(x) = 3x + 4$

$D_1E_1 = 4x^2 + 11x + 6$ $\quad D_0E_0 = 6x^2 + 11x + 4$

$(D_1 + D_0)(E_1 + E_0) = (6x + 4)(4x + 6)$

$= 24x^2 + 52x + 24$

$\Theta(n^{1.58})$

Better than previous

$\Theta(n^2)$

Master Problem:

If $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ (for constants $a > 0$, $b > 1$, $d \geq 0$), then:

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

$T(n) = 4T\left(\frac{n}{2}\right) + O(n^1)$

$a = 4$

$b = 2$

$d = 1$

Since $d < \log_b a$,

$T(n) = O(n^{\log_b a}) = O(n^2)$

If a were 9, then $\log_2 9$ or 3 $\quad O(n^3)$