Week 4 Notes

Wednesday, February 15, 2023 8:12 AM

Linear Search

Linear search works by checking the first element in a array, if it doesn't match it moves to the next slot until it finds a match.

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

Recursive Solution

```
LinearSeach(A, low, high, key)
If high < low
    Return NOT_FOUND
If A[low] = key
    Return low
Return LinearSearch(A, low + 1, high, key)
```

A recurrence relation is an equation recursively defining a sequence of values.

Binary Search

Searching in a sorted array
Input: A sorted array A[low…high]
(low <= I <high: A[i] <= A[i+1]
A key k
Output: A index, I, (low <= I <= high) where A[i] = k
Otherwise, the greater index I, where A[i] < k
Otherwise (k < A[low]), the result is ow - 1
filep
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)
```

Polynomials

Uses of multiplying polynomials

- Error correcting codes
- Large integer multiplication
- Generating functions
- Convolution in signal processing

A(x) = 3x^2 + 2x + 5
B(x) = 5x^2 + x + 2
A(x)B(x) = 15x^4 + 13x^3 + 33x^2 + 9x + 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:
		Product[I + j<- product[i+j] + A[i] * B[j]
	Retun product
```

Runtime: O(n^2)

Naïve divide and conquer

Multiplying Polynomials

- Let $A(x) = D_1(x)x^{\frac{n}{2}} + D_0(x)$ where
  $D_1(x) = a_{n-1}x^{\frac{n}{2}-1} + a_{n-2}x^{\frac{n}{2}-2} + \ldots + a_{\frac{n}{2}}$
  $D_0(x) = a_{\frac{n}{2}-1}x^{\frac{n}{2}-1} + a_{\frac{n}{2}-2}x^{\frac{n}{2}-2} + \ldots + a_0$
- Let $B(x) = E_1(x)x^{\frac{n}{2}} + E_0(x)$ where
  $E_1(x) = b_{n-1}x^{\frac{n}{2}-1} + b_{n-2}x^{\frac{n}{2}-2} + \ldots + b_{\frac{n}{2}}$
  $E_0(x) = b_{\frac{n}{2}-1}x^{\frac{n}{2}-1} + b_{\frac{n}{2}-2}x^{\frac{n}{2}-2} + \ldots + b_0$
- $AB = (D_1x^{\frac{n}{2}} + D_0)(E_1x^{\frac{n}{2}} + E_0)$
  $= (D_1E_1)x^n + (D_1E_0 + D_0E_1)x^{\frac{n}{2}} + D_0E_0$
- Calculate D_1E_1, D_1E_0, D_0E_1, and D_0E_0

Recurrence: $T(n) = 4T(\frac{n}{2}) + kn.$

```
Function Muti2(A, B, n, a1, b1)
	R = array[0…2n-1]
	If n = 1:
		R[0] = A[a1] * B[b1] ; return R
	R[0…n-2] = Mult2(A, B, n/2, a1, b1)
	R[n..2n-2] = Mult2(A, B, n/2, a1 + n/2, b1 + n/2)
	D0E1 = Mult2(A, B, n/2, a1, b1 + n/2)
	D1E0 = Mult2(A, B, n/2, a1+ n/2, b1 )
	R[n/2…n + n/2 -2] += D1E0 + D0E1
```

Faster Divide and Conquer

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 multiplications

Rewrite as:

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

Needs 3 multiplications

Karatsuba Example

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

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

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

$D_1 E_1 = 4x^2 + 11x + 6$ $\qquad$ $D_0 E_0 = 6x^2 + 11x + 4$

$$(D_1 + D_0)(E_1 + E_0) = (6x + 4)(4x + 6) = 24x^2 + 52x + 24$$

$$\begin{aligned} AB = {} & (4x^2 + 11x + 6)x^4 + \\ & (24x^2 + 52x + 24 - (4x^2 + 11x + 6) \\ & - (6x^2 + 11x + 4))x^2 + \\ & 6x^2 + 11x + 4 \\ = {} & 4x^6 + 11x^5 + 20x^4 + 30x^3 + 20x^2 + 11x + 4 \end{aligned}$$

Master Theorem

Theorem

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:

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

Example:
T(n) = 4T(n/2) + O(n)
A = 4, B = 2, d = 1
Since d <logba, T(n) = O(n^logba) = O(n^2)

<u>Sorting Problem</u>

Sorting

Input: Sequence $A[1 \ldots n]$.
Output: Permutation $A'[1 \ldots n]$ of $A[1 \ldots n]$ in non-decreasing order.

Why sorting?
- Sorting data is an important step of many efficient algorithms
- Sorted data allows for more efficient queries

Selection Sort

SelectionSort($A[1 \ldots n]$)

```
for i from 1 to n:
  minIndex ← i
  for j from i+1 to n:
    if A[j] < A[minIndex]:
      minIndex ← j
  {A[minIndex] = min A[i...n]}
  swap(A[i], A[minIndex])
  {A[1...i] is in final position}
```

$\texttt{MergeSort}(A[1 \dots n])$

```
if n = 1:
  return A
m ← ⌊n/2⌋
B ← MergeSort(A[1 ... m])
C ← MergeSort(A[m + 1 ... n])
A′ ← Merge(B, C)
return A′
```