Divide & Conquer

CISC 380: Algorithms

Dr. Sarah Miracle

Merge Sort: High Level Algorithm

input: $A = [a_1, \ldots, a_n]$ (assume n is a power of 2 for simplicity)
output: $F = [f_1, \ldots, f_n]$ with the same elements as A but in SORTED order.

function MERGESORT(A=$[a_1, \ldots, a_n]$)
    if $n = 1$ **then**
        return (A)
    if $n > 1$ **then**
        B $= [a_1, \ldots, a_{n/2}]$, C$=[a_{n/2+1}, \ldots, a_n]$
        D $=$ MergeSort(B)
        E $=$ MergeSort(C)
        F $=$ Merge(D,E)
        return (F)

A Recursive Version of the Merge Algorithm

Input: array $X = [x_1, \ldots, x_k]$ and $Y = [y_1, \ldots, y_l]$ (which are both sorted, so $x_1 \le x_2 \le \ldots \le x_k$ and $y_1 \le y_2 \le \ldots y_l$)
Output: $Z = X \cup Y$ in sorted order

> **function** $\textsc{Merge}$(X,Y)
>
>   **if** $k = 0$ **then**
>
>   **return** (Y)
>
>   **if** $l = 0$ **then**
>
>   **return** (X)
>
>   **if** $x_1 \le y_1$ **then**
>
>   Z $= [x_1, \textsc{Merge}(\ [x_2, \ldots, x_k], Y)$
>
>   **else if** **then**
>
>   Z $= [y_1, \textsc{Merge}(\ \mathrm{X},[y_2, \ldots, y_l])$
>
>   **return** Z

Write the recurrence relation for the running time $T(n)$ of the Merge algorithm.

A Recursive Version of the Merge Algorithm

Input: array $X = [x_1, \ldots, x_k]$ and $Y = [y_1, \ldots, y_l]$ (which are both sorted, so $x_1 \leq x_2 \leq \ldots \leq x_k$ and $y_1 \leq y_2 \leq \ldots y_l$)
Output: $Z = X \cup Y$ in sorted order

function MERGE(X,Y)
  if $k = 0$ **then**
    return (Y)
  if $l = 0$ **then**
    return (X)
  if $x_1 \leq y_1$ **then**
    Z $= [x_1,$ MERGE($[x_2, \ldots, x_k], Y)$
  else if **then**
    Z $= [y_1,$ MERGE(X,$[y_2, \ldots, y_l]$)
  return Z

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

Binary Search

```
function BinarySearch(A[0...n-1], target, low, high)
    if low > high then
        return -1
    mid = (low + high)/2
    if A[mid] > target then
        return BinarySearch(A, target, low, mid-1)
    else if A[mid] < target then
        return BinarySearch(A, target, mid+1, high)
    else
        return mid
```

Write the recurrence relation for the running time $T(n)$ of the binary search algorithm.

Binary Search

```
function BinarySearch(A[0...n-1], target, low, high)
    if low > high then
        return -1
    mid = (low + high)/2
    if A[mid] > target then
        return BinarySearch(A, target, low, mid-1)
    else if A[mid] < target then
        return BinarySearch(A, target, mid+1, high)
    else
        return mid
```

$$T(n) = T(n/2) + c$$

Master Theorem for Recurrence Relations

Theorem

The recurrence

$$\begin{aligned} T(n) &= aT(n/b) + cn^k \\ T(1) &= c, \end{aligned}$$

where $a, c > 0$; $b > 1$; *and* $k \geq 0$ *are constants, solves to:*

$$\begin{aligned} T(n) &= \Theta(n^k) \text{ if } a < b^k \\ T(n) &= \Theta(n^k \log n) \text{ if } a = b^k \\ T(n) &= \Theta(n^{\log_b a}) \text{ if } a > b^k \end{aligned}$$

Master Theorem Case 3: $r > 1$

Last time we saw that when $r > 1$ ($a > b^k$) then $T(n) = cn^k r^{\log_b n}$

$$
\begin{aligned}
T(n) &= cn^k \left(\frac{a}{b^k}\right)^{\log_b n} \\
&= cn^k \left(\frac{a^{\log_b n}}{(b^k)^{\log_b n}}\right) \\
&= cn^k \left(\frac{a^{\log_b n}}{b^{k \log_b n}}\right) \\
&= cn^k \left(\frac{a^{\log_b n}}{(b^{\log_b n})^k}\right) \\
&= cn^k \left(\frac{a^{\log_b n}}{n^k}\right) \\
&= ca^{\log_b n} \\
&= c(b^{\log_b a})^{\log_b n} \\
&= c(b^{\log_b n})^{\log_b a} = cn^{\log_b a} = \Theta(n^{\log_b a})
\end{aligned}
$$

Fibonacci

function $\textsc{Fibonacci}$(num)
    if num $=0$ **then**
        return 0
    if num $=1$ **then**
        return 1
    return $\textsc{Fibonacci}$(num-1) $+$ $\textsc{Fibonacci}$(num-2)

Fibonacci

function FIBONACCI(num)
  if num $=$ 0 **then**
    return 0
  if num $=$ 1 **then**
    return 1
  return FIBONACCI(n-1) $+$ FIBONACCI(n-2)

$$T(n) = T(n-1) + T(n-2) + c$$

n-bit Multiplication

function $\textsc{FastMultiply}(X, Y)$
  if n = 1 **then** return XY
  x_L = leftmost $n/2$ bits of X
  x_R = rightmost $n/2$ bits of X
  y_L = leftmost $n/2$ bits of Y
  y_R = rightmost $n/2$ bits of Y
  $P_1 = \textsc{FastMultiply}(x_L, y_L)$
  $P_2 = \textsc{FastMultiply}(x_R, y_R)$
  $P_3 = \textsc{FastMultiply}(x_L + x_R, y_L + y_R)$
  return $2^n P_1 + 2^{n/2}(P_3 - P_1 - P_2) + P_2$

Counting Inversions: High Level Algorithm

input: $A = [a_1, \ldots, a_n]$
output: the number of inversions AND a sorted A

1. Break A into A_L = first $n/2$ items
   A_R = last $n/2$ items
2. Recursively find # of inversions within A_L AND sort A_L
3. Recursively find # of inversions within A_R AND sort A_R
4. Scan through sorted A_L and A_R to:
   - find # of inversions between A_L and A_R
   - Merge the two sorted lists A_L and A_R

Counting Inversions: Count & Sort

<u>input:</u> $A = [a_1, \ldots, a_n]$ where n is a power of 2
<u>output:</u> the number of inversions AND a sorted A

function $\textsc{Count\_and\_Sort}$(A)
    if $n = 1$ **then**
        return $(0, A)$
    $A_L = [a_1, \ldots, a_{n/2}]$
    $A_R = [a_{n/2+1}, \ldots, a_n]$
    (count1, B) = $\textsc{Count\_and\_Sort}(A_L)$
    (count2, C) = $\textsc{Count\_and\_Sort}(A_R)$
    (count3, D) = $\textsc{Count\_and\_Merge}$(B,C)
    return (count1 + count2 + count3,D)

Counting Inversions: Count & Merge

input: sorted $B = [b_1, \dots, b_k]$ and $C = [c_1, \dots, c_l]$
output: the number of inversions between B & C AND a sorted $B \cup C$

```
function COUNT_AND_MERGE(B,C)
    if k = 0 then
        return (0, C)
    if l = 0 then
        return (0, B)
    if b_1 < c_1 then
        (count, D) = COUNT_AND_MERGE([b_2, ... b_k],C)
        return (count, [b_1,D])
    else
        (count, D) = COUNT_AND_MERGE(B,[c_2, ... c_l])
        return (k+count, [c_1,D])
```

Randomized Median Algorithm

Select(A,k):
input: unsorted $A = [a_1, \dots, a_n]$ (where n is a power of 5)
output: kth smallest element of A

1. Choose a random pivot p.
2. Partition A into $A_{<p}, A_{=p}, A_{>p}$
3. If $|A_{<p}| > (3/4)n$ OR $|A_{>p}| > (3/4)n$ go back to step 1.
4. Recurse on the appropriate set:
   - If $k \leq |A_{<p}|$, then return (Select($A_{<p}, k$))
   - If $|A_{<p}| < k \leq |A_{<p}| + |A_{=p}|$ then return p
   - If $k > |A_{<p}| + |A_{=p}|$ then return Select($A_{>p}, k - |A_{<p}| - |A_{=p}|$)

Deterministic Median Algorithm

Select(A,k):

input: unsorted $A = [a_1, \ldots, a_n]$ (where n is a power of 5)

output: kth smallest element of A

1. Break A into $n/5$ groups of 5 elements each. Call these groups $G_1, G_2, \ldots, G_{n/5}$.
2. For $i = 1 \to n/5$, sort G_i
3. Let $m_i = \text{median}(G_i)$, $S = \{m_1, m_2, \ldots, m_{n/5}\}$
4. p $=$ Select$(S, n/10)$ (so p is the median of S)
5. Partition A into $A_{<p}, A_{=p}, A_{>p}$
6. Recurse on the appropriate set:
   - If $k \leq |A_{<p}|$, then return (Select$(A_{<p}, k)$)
   - If $|A_{<p}| < k \leq |A_{<p}| + |A_{=p}|$ then return p
   - If $k > |A_{<p}| + |A_{=p}|$ then return Select$(A_{>p}, k - |A_{<p}| - |A_{=p}|)$