Lecture P8: Divide-and-Conquer

Divide-and-Conquer paradigm.
- Break up problem into one (or more) smaller subproblems of similar structure.
- Solve subproblems recursively using same method.
- Combine results to produce solution to original problem.

Historical origins.
- Julius Caesar (100 BCE - 44 BCE).
  "Divide et impera.”
  "Veni, vidi, vici.”

Many problems have elegant divide-and-conquer solutions.
- Sorting. quicksort
- Dragon curve.

Quicksort.
- Partition array so that:
  - some partitioning element $a[m]$ is in its final position
  - no larger element to the left of $m$
  - no smaller element to the right of $m$

Partition array so that:
- some partitioning element $a[m]$ is in its final position
- no larger element to the left of $m$
- no smaller element to the right of $m$

<table>
<thead>
<tr>
<th>QUICKSORT</th>
<th>QUICKSORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q U I C K S O R T I S C O O L</td>
<td>Q U I C K S O R T I S C O O L</td>
</tr>
<tr>
<td>I C K I C L Q U S O R T S O O</td>
<td>C C I I K L O O O Q R S S T U</td>
</tr>
</tbody>
</table>

Sorting each “half” recursively.
Quicksort

Partition array so that:
- some partitioning element \( a[m] \) is in its final position
- no larger element to the left of \( m \)
- no smaller element to the right of \( m \)

Sort each “half” recursively.

void quicksort(char a[], int left, int right) {
    int m; if (right > left) {
        m = partition(a, left, right);
        quicksort(a, left, m - 1);
        quicksort(a, m + 1, right);
    }
}

quicksort.c (see Sedgewick Program 7.1)

int partition(char a[], int left, int right) {
    int i = left - 1;  // left to right pointer
    int j = right;     // right to left pointer
    while(1) {
        while (a[++i] < a[right])
            ;
        while (a[right] < a[--j])
            if (j == left) break;
        if (i >= j) break; // pointers cross
        swap(a, i, j); // swap two elements
    }
    swap(a, i, right); // swap partition element
    return i;
}

partition (see Sedgewick Program 7.2)

main() {
#include <stdio.h>
define N 14

    char a[] = "pseudomythical";
    printf("Before: %s\n", a);
    quicksort(a, 0, N-1);
    printf("After:  %s\n", a);
    return 0;
}

void swap(char a[], int i, int j) {
    char t;
    t = a[i]; a[i] = a[j]; a[j] = t;
}
Quicksort: Performance

Quicksort vs. Insertion sort.

### Insertion Sort ($N^2$) vs. Quick sort ($N \log N$)

<table>
<thead>
<tr>
<th></th>
<th>Insertion Sort Time</th>
<th>Quicksort Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>computer</td>
<td>thousand</td>
</tr>
<tr>
<td>home</td>
<td>instant</td>
<td>2.8 hours</td>
</tr>
<tr>
<td>super</td>
<td>instant</td>
<td>1 second</td>
</tr>
<tr>
<td></td>
<td>thousand</td>
<td>instant</td>
</tr>
<tr>
<td></td>
<td>million</td>
<td>instant</td>
</tr>
</tbody>
</table>

Stay tuned: Analysis of Algorithms Lecture.

---

### Dragon (Jurassic Park) Curve

Fold a wire in half $n$ times. Unfold to right angles.

$n = 0$

$n = 1$

$n = 2$

$n = 3$

$n = 4$

$n = 5$

$n = 6$

$n = 7$

$n = 8$

$n = 9$

$n = 10$

$n = 11$

$n = 12$

---

### Drawing a Dragon Curve

Use simple “turtle graphics.”
- **F**: move turtle forward one step (pen down).
- **L**: turn left 90°.
- **R**: turn right 90°.

Example.
- **F L F L F R F**
Drawing a Dragon Curve

Use simple "turtle graphics."
- F: move turtle forward one step (pen down).
- L: turn left 90°.
- R: turn right 90°.

Example.
- dragon(0): F
- dragon(1): F L F
- dragon(2): F L F L F R F
- dragon(3): F L F L F R F F L F L F R F F F

Example.
- dragon(0): F
- dragon(1): F L F
- dragon(2): F L F L F R F
- dragon(3): F L F L F R F F L F L F R F F F

Recursive Dragon Curve Program

A dragon curve of order n is:
- Dragon curve of order n-1.
- Turn left.
- Inverted dragon curve of order n-1.
  – backwards, switch L and R

```c
void dragon(int n) {
  if (n == 0)
    F();
  else {
    dragon(n-1);
    L();
    nogard(n-1);
  }
}
```

```c
void nogard(int n) {
  if (n == 0)
    F();
  else {
    dragon(n-1);
    R();
    nogard(n-1);
  }
}
```

Need implementation of nogard().

Drawing in Turtle Graphics

```c
void F(void) { printf("D 10\n"); }
void L(void) { printf("R 90\n"); }
void R(void) { printf("R -90\n"); }
```

Nonrecursive Dragon Curve

To write down the whole dragon curve sequence:
1. Put F in every other space.
2. Put L, R (alternating) in every other remaining space.
3. Repeat Step 2 until done.

Proof.

```
D(0) L N(0) D(0) R N(0) D(0) L N(0) D(0) R N(0)
```

Drawback: requires excessive memory for big dragons.

Drawing a Dragon Curve

Observation: nogard(n) is same as dragon(n), except middle move is R not L

Justification: by definition, nogard(n) is inverted dragon(n).

- dragon(n) = dragon(n-1) L nogard(n-1)
- nogard(n) = dragon(n-1) R nogard(n-1)

"inverted" dragon(3): reverse string, switch L and R
The dragon curve and binary integers.

- The kth turtle turn (ignore Fs) depends on the bit to the left of the rightmost 1 in the binary representation of k.
  - L if bit = 0, R if bit = 1

Proof: (by induction on order of curve)

- Base case: \( \text{dragon}(1) = FLF \).
- Assume true for \( \text{dragon}(n) \), and consider \( \text{dragon}(n+1) \).
- Recall: only difference between top and bottom halves is their middle moves.

Consequence: simple iterative algorithm that requires little storage.

### Finding the Dragon Bit

Logical expression to find bit \( b \) to the left of the rightmost 1.

\[
k \& ((k \wedge (k - 1)) + 1)
\]

<table>
<thead>
<tr>
<th>Dec</th>
<th>Bin</th>
<th>L</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Summary

Why learn recursion?

- New mode of thinking.
- Powerful programming tool to solve a problem.
  - divide-and-conquer

Examples

- Quicksort.
- Dragon curve.

Many other problems have elegant divide-and-conquer solutions.

- Database search.
- Integer arithmetic! (multiplication, division, RSA function)
  - stay tuned for RSA crypto assignment