Lecture P7: 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 $m$

Sort 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.

```c
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)

```c
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)

### Quicksort In Action

```c
#include <stdio.h>
#define N 14

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

`main()`

```c
void swap(char a[], int i, int j) {
    char t;
    t = a[i]; a[i] = a[j]; a[j] = t;
}
```

`swap()`
Quicksort: Performance

Quicksort vs. Insertion sort.

<table>
<thead>
<tr>
<th></th>
<th>computer</th>
<th>thousand</th>
<th>million</th>
<th>billion</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>home</td>
<td>instant</td>
<td>2.8 hours</td>
<td>317 years</td>
</tr>
<tr>
<td></td>
<td>super</td>
<td>instant</td>
<td>1 second</td>
<td>1.6 weeks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>thousand</th>
<th>million</th>
<th>billion</th>
</tr>
</thead>
<tbody>
<tr>
<td>instant</td>
<td>instant</td>
<td>instant</td>
<td></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.

Insertion Sort \( (N^2) \)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
<td>Instant</td>
<td>2.8 hours</td>
<td>317 years</td>
</tr>
<tr>
<td>Super</td>
<td>Instant</td>
<td>1 second</td>
<td>1.6 weeks</td>
</tr>
</tbody>
</table>

QuickSort \( (N \log N) \)

Stay tuned: Analysis of Algorithms Lecture.

Dragon (Jurassic Park) Curve

n = 0  n = 4

n = 1

n = 2  n = 5

n = 3

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 F
- dragon(3): F L F L F F L F L F F F
- dragon(4): F L F L F F L F L F F F L F L F F F L F L F F F

"inverted" dragon(3): reverse string, switch L and R

**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);
    }
}
```

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) F F F F
D(1) L F
D(2) L L
D(3) L L L L
```

**Drawback:** requires excessive memory for big dragons.
The dragon curve and binary integers.

- The $k$th 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: dragon(1) = FLF.
- Assume true for dragon(n), and consider 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.

$$k \& ((k \uparrow (k-1)) + 1)$$

```
0 1 1 b 1 0 0 0
   k

0 1 1 b 0 1 1 1
   (k \uparrow (k-1)) + 1

0 0 0 0 1 1 1 1
   (k \uparrow (k-1))

0 0 0 b 0 0 0 0
   k \& ((k \uparrow (k-1)) + 1)
```

---

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