MonkeySort

Keith Gallagher

Florida Institute of Technology
An Introduction....

HAVE YOU TRIED TURNING IT OFF

AND FLING FECES AT IT?

The Quark
Infinite monkey theorem

A monkey hitting keys at random on a typewriter keyboard for an infinite amount of time will *almost surely* type a given text, such as the complete works of William Shakespeare.

1 July 2003 .. Sometime around February of 2005 (the last documented total of) characters 24 characters matched from Henry IV part 2. 2,737 billion billion billion billion monkey-years
Infinite monkeysort theorem

A *monkey* hitting keys at random on a typewriter keyboard for an infinite amount of time will *almost surely* sort an array of integers!
Specification of a sorted array

\[ a[i] \leq a[i + 1] \ldots \]

\[ a[\text{perm}(i)] \leq a[\text{perm}(i + 1)] \text{ for some perm} \]

\[ b = \text{perm}(a) \text{ and } b(i) \leq b(i + 1) \]
A simple version for sorting a deck of cards

- Early MonkeySort
  - throw cards in tub
  - stir
  - pick up cards
  - until sorted
  - this may take a while...

Bathtub of the USS Maine (raised 1911, Havana Harbor)
Source: http://www.roadsideamerica.com/attract/OHFINbathtub.html
Evolved MonkeySort

- Guessing two array elements to swap
  - could be the same one
- **Do Not Compare**, just exchange
  - equivalent to “throw/stir/pick-up”
- Will it ever stop?
  - Almost surely!
Sort Examples

QED!

6 → 1
2 → 2
3 → 3
4 → 4
5 → 5
1 → 6
7 → 7
8 → 8

Not so QED…

8 → 8 → 8 → 8
2 → 7 → 7 → 7
6 → 6 → 4 → 6
1 → 1 → 1 → 1
7 → 2 → 2 → 2
3 → 3 → 3 → 3
5 → 5 → 5 → 5
4 → 4 → 6 → 4
void transpose ( int a[], int n)
{
    int i, j, temp;
    i = (int) random() % n;
    j = (int) random() % n;
    temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}

int checksort (int a[], int n )
{
    int i,j;
    for(i = 0, j = 1; j < n ; i++, j++)
        if (a[i] > a[j]) return 0;
    return 1;
}

main (int argc , char * argv[])
{
    int i, n, *a, count = 0 ;
    srand(time((time_t *)0));

    n = atoi(argv[1]);
a = (int *) malloc(n*sizeof(int));

    for( i = 0 ; i < n ; i++)
    {  
a[i] = (int)random() ;
    }
    while (!checksort(a,n))
    { count++;
      transpose (a, n);
    }
    printf("%d\n",count);
}
The Program Itself

• Uses **system time** and **command line arguments**

• Is **Partially Correct**
  – discuss reasoning about programs

• **NP**, as solution is “**guess and test**”
MonkeySort
Observations

• Simple
• Easy (for non-programmers) to understand
• NP
• Partially correct
• Fun!
Results and Observations: Things to Talk About

• It **does** halt
• Can you guess beforehand *about* how guesses it will take?
• Time to halt varies
  – larger sets may sort faster than smaller
• Best-known technique to solve the “garbage truck problem” ie. shortest Hamiltonian circuit.
Screen Shot of “top” Utility

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PRI</th>
<th>NI</th>
<th>SIZE</th>
<th>RSS</th>
<th>SHARE</th>
<th>STAT</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>2633</td>
<td>kbg</td>
<td>17</td>
<td>0</td>
<td>500</td>
<td>500</td>
<td>384</td>
<td>R</td>
<td>99.4</td>
<td>0.1</td>
<td>84:44</td>
<td>monkey 13</td>
</tr>
<tr>
<td>2819</td>
<td>kbg</td>
<td>11</td>
<td>0</td>
<td>864</td>
<td>864</td>
<td>672</td>
<td>R</td>
<td>0.5</td>
<td>0.3</td>
<td>0:00</td>
<td>top</td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>8</td>
<td>0</td>
<td>332</td>
<td>288</td>
<td>276</td>
<td>S</td>
<td>0.0</td>
<td>0.1</td>
<td>0:03</td>
<td>init</td>
</tr>
<tr>
<td>2</td>
<td>root</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SW</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00</td>
<td>keventd</td>
</tr>
<tr>
<td>3</td>
<td>root</td>
<td>19</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SW</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00</td>
<td>ksoftirqd_CPU0</td>
</tr>
<tr>
<td>4</td>
<td>root</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SW</td>
<td>0.0</td>
<td>0.0</td>
<td>0:04</td>
<td>kswapd</td>
</tr>
<tr>
<td>5</td>
<td>root</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SW</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00</td>
<td>bdfflush</td>
</tr>
<tr>
<td>6</td>
<td>root</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SW</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00</td>
<td>kupdated</td>
</tr>
<tr>
<td>128</td>
<td>root</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SW</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00</td>
<td>khubd</td>
</tr>
<tr>
<td>168</td>
<td>daemon</td>
<td>9</td>
<td>0</td>
<td>164</td>
<td>92</td>
<td>92</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00</td>
<td>/sbin/portmap</td>
</tr>
<tr>
<td>175</td>
<td>root</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SW</td>
<td>0.0</td>
<td>0.0</td>
<td>0:01</td>
<td>rpciod</td>
</tr>
<tr>
<td>176</td>
<td>root</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SW</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00</td>
<td>lockd</td>
</tr>
<tr>
<td>354</td>
<td>root</td>
<td>9</td>
<td>0</td>
<td>484</td>
<td>472</td>
<td>428</td>
<td>S</td>
<td>0.0</td>
<td>0.1</td>
<td>0:01</td>
<td>/sbin/syslogd</td>
</tr>
<tr>
<td>367</td>
<td>root</td>
<td>9</td>
<td>0</td>
<td>868</td>
<td>132</td>
<td>132</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00</td>
<td>/sbin/klogd</td>
</tr>
</tbody>
</table>
Some of Our Big Ideas

• NP Hard
  – the ones with **best** known solutions equivalent to “Guess and Test”

• Partial Correctness
  – the program is correct **if it stops**!

• Algorithmic and Empirical Analysis
Some Bigger Ideas

- Stirling’s approximation
- Code coverage tools
- Integer overflow
- Permutations as products of transpositions
- Is P == NP?
- Comparison of analytical results with empirical results
What Do Computer Scientists Do All Day?

• Look for “better” solutions
  – build
• Experimentally determine program properties
• Must carefully consider all solution properties (overflow, timing, etc)
• CPU cycles are cheap; people are expensive: “work smart, not hard”
Words

- Rearrangement
- Criteria
- Functional
- Specification
- Implementation
- Pre/Postcondition
- Assertion
- Guard
- Indices
- Addresses
- Algebraically
- Permutation
- Correctness
thanks for listening!