MonkeySort

Keith Gallagher

Florida Institute of Technology

An Introduction....

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] <= a[i + 1].....

a[perm(i)] <= a[perm(i + 1)] for some perm

b = perm(a) and b(i) <= 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

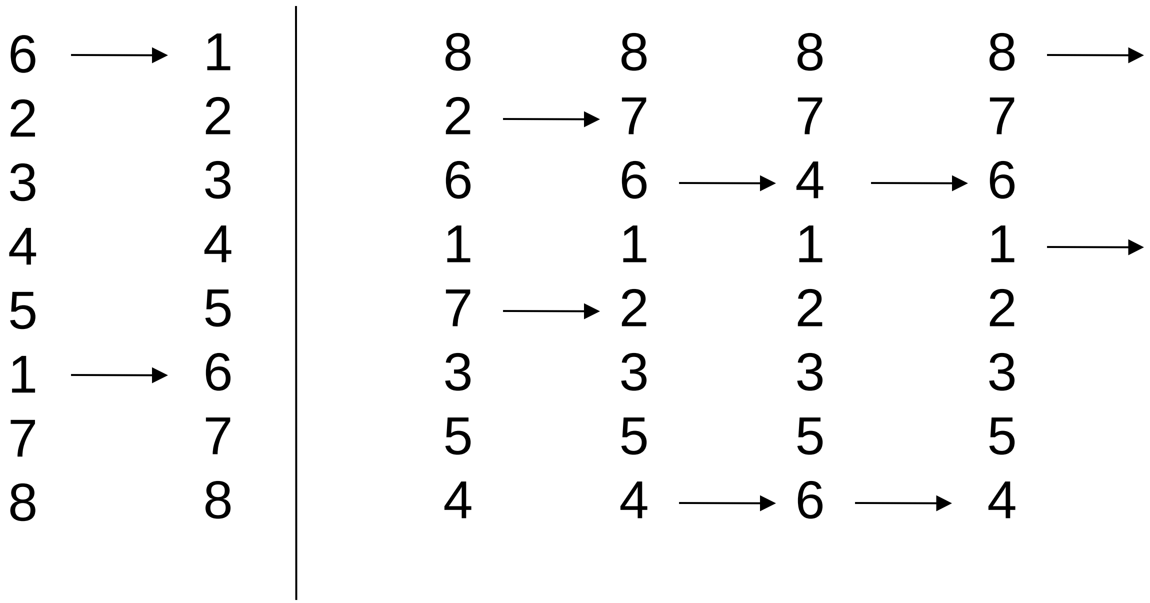
Evolved MonkeySort

- 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!

Not so QED…



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

Code

```
main (int argc , char * argv[])
{
  int i, n, *a, count = 0 ;
  srandom(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);
}
```

```
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;
}
```

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

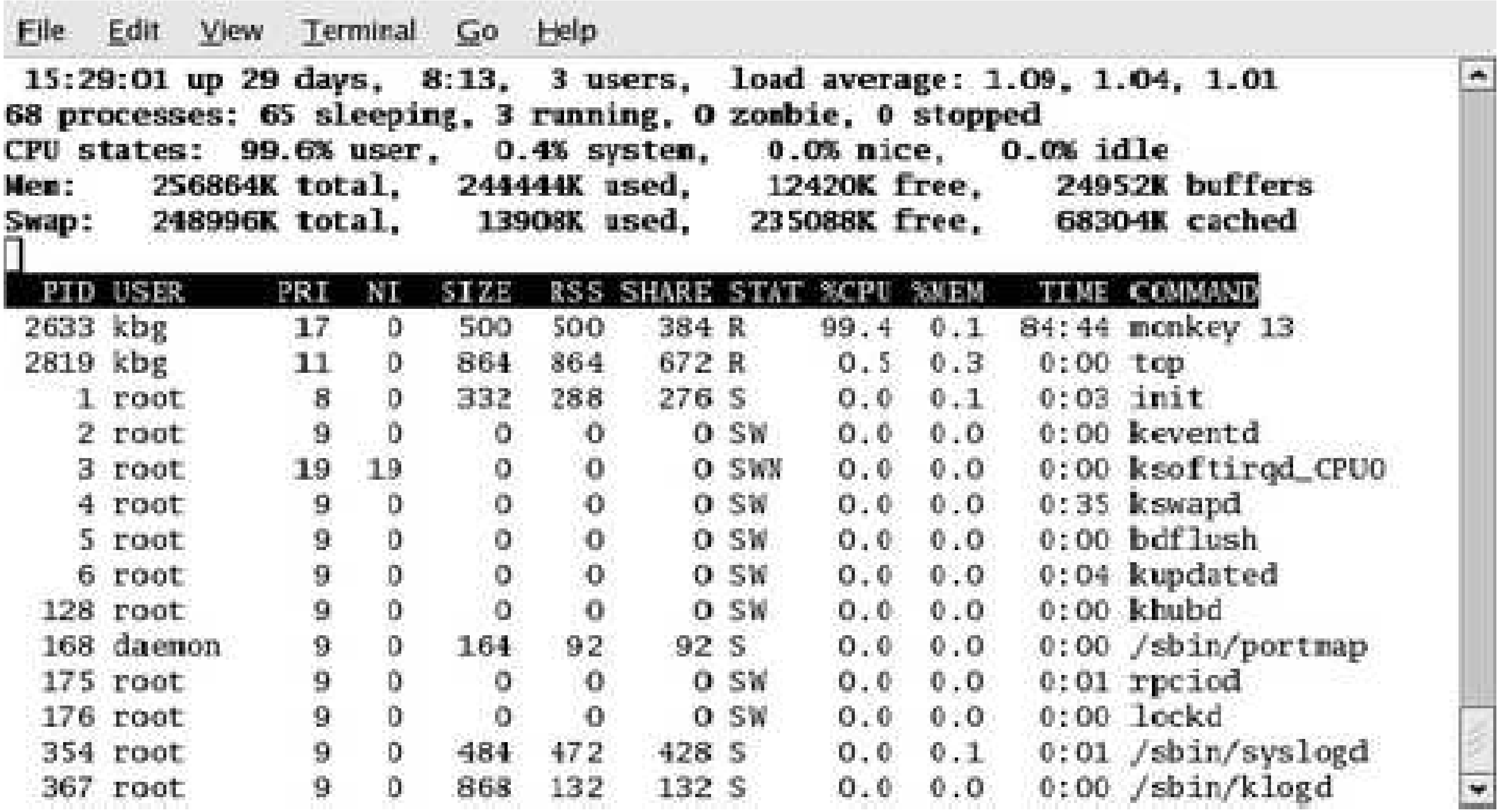
- Simple
- Easy (for non-programmers)

  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

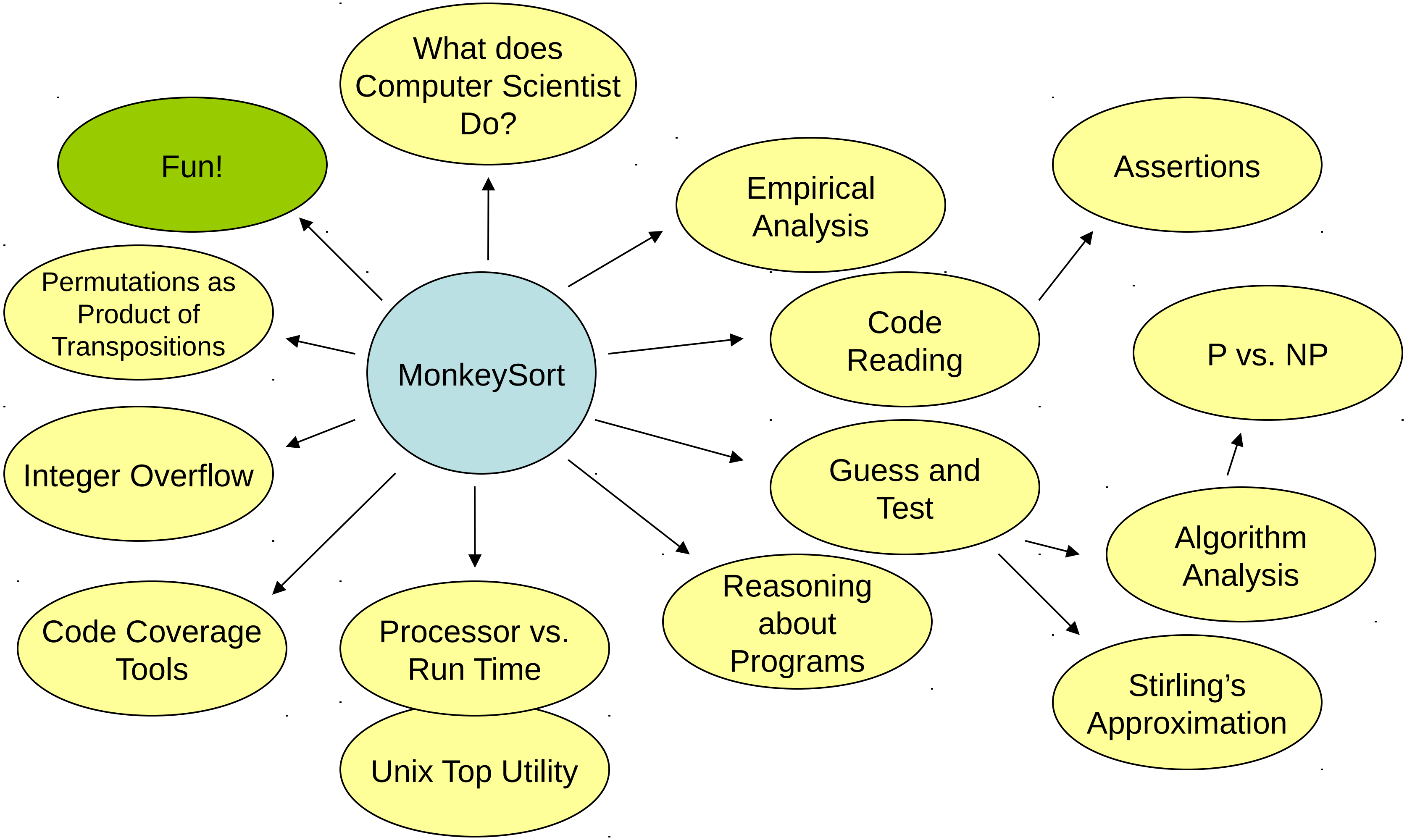


```
 15:29:01 up 29 days,  8:13,  3 users,  load average: 1.09, 1.04, 1.01
68 processes: 65 sleeping, 3 running, 0 zombie, 0 stopped
CPU states:  99.6% user,   0.4% system,   0.0% nice,   0.0% idle
Mem:    256864K total,   244444K used,    12420K free,    24952K buffers
Swap:   248996K total,    13908K used,   235088K free,    68304K cached

  PID USER      PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
 2633 kbg        17   0   500  500   384 R    99.4  0.1  84:44 monkey 13
 2819 kbg        11   0   864  864   672 R     0.5  0.3   0:00 top
    1 root        8   0   332  288   276 S     0.0  0.1   0:03 init
    2 root        9   0     0    0     0 SW    0.0  0.0   0:00 keventd
    3 root       19  19     0    0     0 SWN   0.0  0.0   0:00 ksoftirqd_CPU0
    4 root        9   0     0    0     0 SW    0.0  0.0   0:35 kswapd
    5 root        9   0     0    0     0 SW    0.0  0.0   0:00 bdflush
    6 root        9   0     0    0     0 SW    0.0  0.0   0:04 kupdated
  128 root        9   0     0    0     0 SW    0.0  0.0   0:00 khubd
  168 daemon      9   0   164   92    92 S     0.0  0.0   0:00 /sbin/portmap
  175 root        9   0     0    0     0 SW    0.0  0.0   0:01 rpciod
  176 root        9   0     0    0     0 SW    0.0  0.0   0:00 lockd
  354 root        9   0   484  472   428 S     0.0  0.1   0:01 /sbin/syslogd
  367 root        9   0   868  132   132 S     0.0  0.0   0:00 /sbin/klogd
```

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



MonkeySort

What does Computer Scientist Do?

Fun!

Permutations as Product of Transpositions

Integer Overflow

Code Coverage Tools

Processor vs. Run Time

Unix Top Utility

Reasoning about Programs

Guess and Test

Code Reading

Empirical Analysis

Assertions

P vs. NP

Algorithm Analysis

Stirling’s Approximation

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/Postcondion

Assertion

Guard

Indices

Addresses

Algebraically

Permutation

Correctness

thanks for listening!