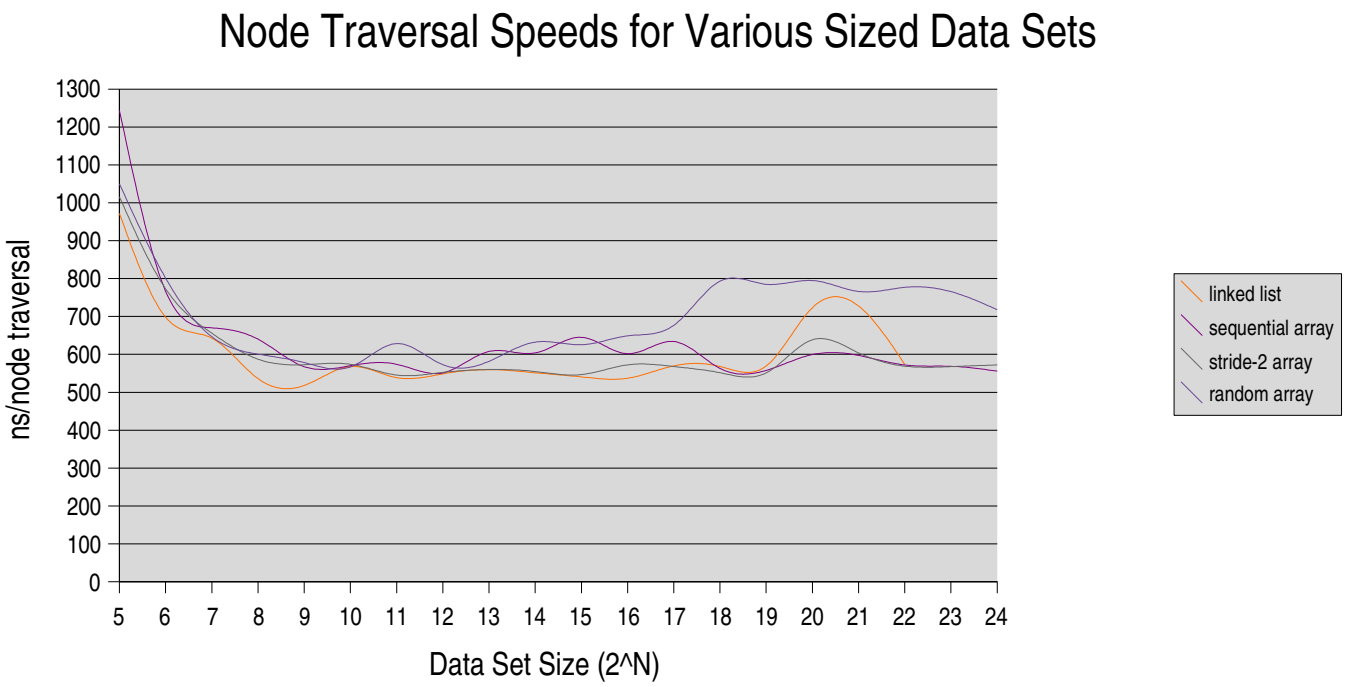
Chris Kelly
CS8803 – Fall 2006
Homework Assignment 1



This graph may be a bit different than other students because I chose to implement the assignment using perl instead of a compiled language. I thought that this might have some effect on caching or optimizations, but with basic verbal comparisons with several of my classmates, my performance seems to be much more consistent without regards to caching. It is difficult to point out any place in the graph where caching may have had an effect, and the only statistically significant slowdown as data set size increased was around 2^17 when the random array access time jumped up. This is most likely due to the cost of rapid creation and teardown of the temporary mappings to higher memory required for such large data sets (Linux HIGHMEM).

These numbers were run on a 2Ghz AMD Athlon 2600 with 1.5GB of DDR SDRAM. This allowed for the array to be scaled to 2^24, but prevented my linked list implementation from scaling any higher than 2^22. The initial high traversal time is most likely due to entering and leaving the array traversal method, but it essentially levels out once the data set hits 2^6 items.

To ensure proper traversals, I first ran the traversal printing values in each node for small data sets and then adding up the values in each node for larger data sets and comparing that to the expected sum. To get accurate timings I ran each test for each data set size 10 times and used the mean of the 10 values for the value in the graph. If the outliers were further than 25% from the mean, I re-ran the set of 10 tests again to get consistent numbers.

The asymptotic complexity of each algorithm is O(1).

The algorithmic model is accurate because the running time of the algorithms increases linerarily regardless of the size of the data set. (The per-data item running time remains relatively constant)

```
#!/usr/bin/perl -w
use strict;
use Time::HiRes qw(usleep ualarm gettimeofday tv_interval);
use Getopt::Std;
my %opts;
our @array;

sub print_results {
	my $elapsed = $_[0];
	my $size = $_[1];
	print "Total Runtime: ". $elapsed*1000 ." milliseconds\n";
	print "Runtime per node: ". ($elapsed/$size)*1000000000 ." nanoseconds\n";
}

sub traverse_array {
	our @array;
	my $start = [gettimeofday];
	for (my $i = 0; $array[$i] != -1; $i = $array[$i]) {
		;
	}
	my $elapsed = tv_interval($start);
	print_results($elapsed, $#array);
}

# linked list implementation ideas from http://perldoc.perl.org/perlfaq4.html
sub ll_append {
	my($list, $value) = @_;
	my $node = { "V" => $value };
	if ($list) {
		$node->{ "L" } = $list->{ "L" };
		$list->{ "L" } = $node;
	} else {
		$_[0] = $node;
	}
	return $node;
}

sub ll_runsize {
	my $size = $_[0];
	my ($head, $tail);
	$tail = ll_append($head, 1);
	for my $value ( 2 .. $size ) {
		$tail = ll_append($tail, $value);
	}
	my $start = [gettimeofday];
	for (my $node = $head; $node; $node = $node->{ "L" }) {
		;
	}
	my $elapsed = tv_interval($start);
	print_results($elapsed, $size);
}

sub arrayseq {
	my $size = $_[0];
	our @array = (1 .. $size);
	$array[$size] = -1;
```

```
}

sub arraystride {
	my $size = $_[0];
	our @array = ();
	for (my $i = 0; $i < $size; $i+=2) {
		$array[$i] = $i+2;
	}
	for (my $i = 1; $i < $size; $i+=2) {
		$array[$i] = $i+2;
	}
	#important: this assumes an even number of items. it wont work for odd!
	$array[$size] = 1;
	$array[$size-1] = -1;
}

sub arrayrandom {
	my $size = $_[0];
	my @targets = (1 .. $size);
	my @randomtargets = ();
	for ( @targets ) {
		my $r = rand(@randomtargets+1);
		push(@randomtargets,$randomtargets[$r]);
		$randomtargets[$r] = $_;
	}
	our @array = ();
	my $last = 0;
	while ($#randomtargets >= 0) {
		my $value = pop(@randomtargets);
		$array[$last] = $value;
		$last = $value;
	}
	$array[$last] = -1;
}

getopt('st', \%opts);
if (defined $opts{'s'} && defined $opts{'t'}) {
	print("Number of nodes: $opts{'s'}\n");
	print("Type of traversal: $opts{'t'}\n");
	if ($opts{'t'} eq 'linkedlist') {
		ll_runsize(2 ** $opts{'s'});
	} elsif ($opts{'t'} eq 'arrayseq') {
		arrayseq(2 ** $opts{'s'});
		traverse_array;
	} elsif ($opts{'t'} eq 'arraystride') {
		arraystride(2 ** $opts{'s'});
		traverse_array;
	} elsif ($opts{'t'} eq 'arrayrandom') {
		arrayrandom(2 ** $opts{'s'});
		traverse_array;
	} else {
		print("invalid traversal type");
	}
} else {
	print("Must specifiy -s size -t [linkedlist|arrayseq|arraystride|arrayrandom] !\n");
}
```