In lab 1c, I use the following 3 test cases as my benchmarks:

Simpsh:

**./simpsh --rdonly a --pipe --trunc --wronly c --wronly d --command 0 2 4 cat —command\
1 3 4 sort —wait**

**./simpsh --profile --rdonly a --pipe --pipe --creat --trunc --wronly c --creat --append --wronly d\
--command 0 2 6 sort --command 1 4 6 cat b - --command 3 5 6 tr a-z A-Z —wait**

**./simpsh --profile --rdonly a --pipe --pipe --pipe --creat --trunc --wronly c --creat --append --wronly d\
--profile --command 0 2 8 sort --profile --command 1 4 8 cat - a --profile --command 3 6 8 tr '[:space:]' 'z' --profile --command 5 7 8 uniq - c --wait**

Bash/Dash:

cat < a | sort > c 2>d

sort < a | cat b - | tr a-z A-Z > c 2>>d

sort a | cat - a | tr '[:space:]' 'z' | uniq - c > c 2>>d

For tests above, the file ‘a’ to be read in has a size of 27009 bytes. The following table reveals the system time of my simplest method and standard method running in dash and bash shell environment:

Simpsh Benchmark Tests Time Measurement			
	Benchmark 1 (s)	Benchmark 2 (s)	Benchmark 3 (s)
Real Time	0.004	0.012	0.014
User time	0	0.004	0.005
Sys Time	0.0016	0.003	0.004

Bash Benchmark Tests Time Measurement			
	Benchmark 1 (s)	Benchmark 2 (s)	Benchmark 3 (s)
Real Time	0.009	0.007	0.011
User time	0.004	0.004	0.003
Sys Time	0.002	0.004	0.006

Dash Benchmark Tests Time Measurement			
	Benchmark 1 (s)	Benchmark 2 (s)	Benchmark 3 (s)
Real Time	0.008	0.007	0.07
User time	0.003	0.005	0.005
Sys Time	0.003	0.003	0.005

As you can see, Simplesh has the best system time.

To further compare the time, I put —profile option before every subcommand and I gets the following result:

—profile option Benchmark result (subcommand only)						
	Benchmark 1 (s)		Benchmark 2 (s)		Benchmark 3 (s)	
	child	parent	child	parent	child	parent
User time	0.0033	0.0033	0.005	0.005	0.0041	0.0041
Sys Time	0.0019	0.0019	0.0008	0.0008	0.003	0.003

The child and parent has the same user and system time because for those three simpsh commands, the parent will always wait for all child to finish executing.

As we can see from the chart above, the result does not really match with the previous table. This is mainly because we are only performing —profile option on subcommands. Another reason behind this is that there is certain degree of fluctuation in time values as each time we measure, the value changes slightly.

However, the ‘time’ method does not seem to give high precision values, to get a more clear view on the simpsh method, I used the ‘strace -c’ method to get a more accurate system time usage printout:

TestBench 1:

% time	seconds	usecs/call	calls	errors	syscall
57.57	0.000783	392	2		wait4
17.50	0.000238	48	5		open
9.19	0.000125	9	14		write
6.54	0.000089	45	2		clone
2.94	0.000040	5	8		mmap
1.40	0.000019	5	4		mprotect
1.32	0.000018	2	8		getrusage
0.81	0.000011	2	6		close
0.59	0.000008	8	1		munmap
0.51	0.000007	2	4		brk
0.37	0.000005	5	1	1	access
0.37	0.000005	5	1		pipe
0.29	0.000004	1	3		fstat
0.29	0.000004	4	1		execve
0.15	0.000002	2	1		read
0.15	0.000002	2	1		arch_prctl
100.00	0.001360		62	1	total

TestBench 2:

% time	seconds	usecs/call	calls	errors	syscall
81.98	0.002648	883	3		wait4
6.16	0.000199	40	5		open
4.46	0.000144	48	3		clone
3.62	0.000117	5	23		write
0.80	0.000026	3	8		mmap
0.71	0.000023	2	14		getrusage
0.56	0.000018	5	4		mprotect
0.37	0.000012	2	8		close
0.31	0.000010	5	2		pipe
0.25	0.000008	8	1		munmap
0.22	0.000007	2	4		brk
0.15	0.000005	2	3		fstat
0.15	0.000005	5	1	1	access
0.15	0.000005	5	1		execve
0.06	0.000002	2	1		read
0.03	0.000001	1	1		arch_prctl
100.00	0.003230		82	1	total

TestBench 3:

% time	seconds	usecs/call	calls	errors	syscall
79.99	0.004936	1234	4		wait4
10.39	0.000641	128	5		open
3.50	0.000216	7	30		write
3.11	0.000192	48	4		clone
0.62	0.000038	2	18		getrusage
0.53	0.000033	4	8		mmap
0.39	0.000024	6	4		mprotect
0.39	0.000024	24	1		execve
0.31	0.000019	2	10		close
0.24	0.000015	5	3		pipe
0.15	0.000009	9	1		munmap
0.11	0.000007	2	4		brk
0.11	0.000007	7	1	1	access
0.10	0.000006	2	3		fstat
0.03	0.000002	2	1		read
0.03	0.000002	2	1		arch_prctl
100.00	0.006171		98	1	total

Now the time value has a higher precision and we can see that the value does match quite well with previous values. Also, with the strace -c method, we are also able to see the time value taken by each sys call in the program and the number of evoked system calls.