Collecting Timing Data

CS 165, Project in Algorithms and Data Structures
UC Irvine
Spring 2020

Presented by Rob Gevorkyan

The time command

- All the major OSes can provide detailed performance data of program runs
  - Linux: /usr/bin/time ...
  - Windows 10: Using Windows Subsystem for Linux, use the same executable as for Linux
  - macOS: With homebrew installed:
    - brew install gnu-time
    - gtime …

Using the time command

Default formatting on Ubuntu 18.04 (may differ on other OSes)

```
/usr/bin/time shell_sort1 32768

1.01user 0.00system 0:01.01elapsed 99%CPU (0avgtext+0avgdata
1192maxresident)k
0inputs+0outputs (0major+54minor)pagefaults 0swaps

# the output above may give us more information than we need.
# it is also not in a format that is easy to parse.
```

Using the time command

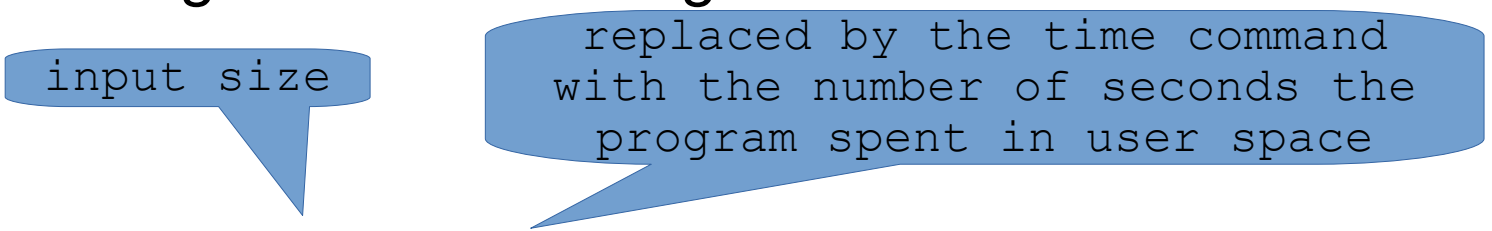
Custom formatting

```
# change output just to contain the number of
# seconds the program spent in user space.
# you may want to look at some of the other metrics.
# read the documentation to see what's available
/usr/bin/time --format "%U" ./project1 shell_sort1 -n 32768

1.01
```

Saving the timing data

- If you use the suggested benchmark driver format, you can write a bash script that loops over all of the function names and your desired array sizes.
- Each call to time might look something like this:




```
/usr/bin/time –format “1048576, %U”
shell_sort1 1048576 -o
shell_sort1_timings.csv --append
```

Timings file format

- We suggest putting the timings for each function in a separate file.
- The file should be a csv (comma-separated value) file with a header line indicating what each column represents (we will use this later).
- Example:

shell_sort1.csv

```
size, time
1024, 0.89
2048, 2.3
4096, 5.18
...
```

we use powers of 2 since later we will plot on log scale

for loops in bash

- bash supports for loops in a few different ways shown below:

```
for fn in shell_sort, merge_sort,
insertion_sort
do
   echo $fn
done

#prints the following lines:
#shell_sort
#merge_sort
#insertion_sort
```

```
for j in {10..14}
do
   echo $((2**j))$
done

#prints the following lines:
#1024
#2048
#4096
#8192
```

nesting for loops in bash

- We can combine the forms just shown to get something that loops over all of the functions to benchmark with all of the sizes.

  Here is a snippet showing the basic structure:

```
for fn in shell_sort, merge_sort, insertion_sort
do
    for ((i = 10; i <= 20; i++)); do
        # replace the following line with a call to ‘time’
        echo "$fn $((2**i))"
    done
done
```