#### 3. Timing: Part 1 (20 Points):

Compile and run the program without any extra optimizations, but with *profiling* for timing:

gcc -c -pg -O0 main.c

gcc -c -pg -O0 mergeSort.c

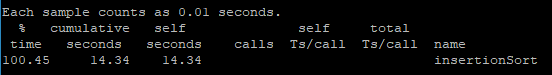
gcc -c -pg -O0 insertionSort.c

gcc main.o mergeSort.o insertionSort.o -pg -O0 -o assign1-0

*Run the program twice* timing it both times, and answer the following:

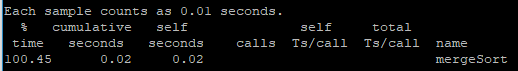
1. *How many****self seconds****did insertionSort() take?*

My answer : 14.34



1. *How many****self seconds****did mergeSort() take?*

My answer : 0.02



4. **Timing: Part 2 (20 Points):**

Compile and run the program *with* optimization, but with *profiling* for timing:

gcc -c -pg -O2 main.c

gcc -c -pg -O2 mergeSort.c

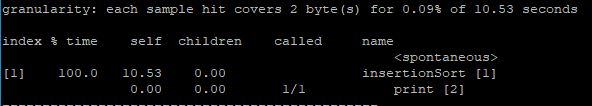
gcc -c -pg -O2 insertionSort.c

gcc main.o mergeSort.o insertionSort.o -pg -O2 -o assign1-2

*Run the program twice* timing it both times, and answer the following:

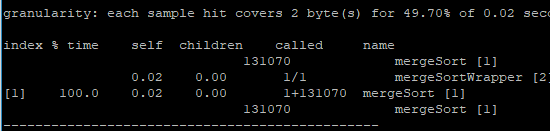
1. *How many****self seconds****did insertionSort() take?*

My answer : 10.53



1. *How many****self seconds****did mergeSort() take?*

My answer : 0.02



5. **Human vs. Compiler Optimization (10 Points):**

Which is faster:

* A bad algorithm and data-structure optimized with -O2
* A good algorithm and data-structure optimized with -O0

My answer: A good algorithm and data-structure optimized with -O0

#### 6. Parts of an executable (Points 20):

Please find the following inside of assign1-0 by using objdump.

* If it *can* be found then *both*
  1. Give the objdump command, and
  2. Show the objdump result
* If it *cannot* be found then tell why not. Where in the memory of the runtime process is it?

Look for:

1. The string constant in main()
2. Global integer numNumbers in main.c
3. The code for freeList()
4. The pointer argument nodePtr in freeList()

|  |  |  |
| --- | --- | --- |
| **Question** | **Command** | **Result** |
| (A) | objdump -d -j .rodata assign1-0 |  |
| (B) | objdump -t -j .rodata assign1-0 |  |
| (C) | objdump -d -j .text assign1-0 |  |
| (D) | NA | nodePtr is a local variable, so we can only find it in memory om stack |

#### 7. Compiler optimizations (Points 10):

Look at the assembly code of assign1-0 and assign1-2. Find and show at least **2** optimizations that the compiler did in either assign1-2 or assign1-0.

|  |  |  |
| --- | --- | --- |
|  | assign1-0 | assign1-2 |
| Optimization 1 : insertionSort function:  the assembly code with O2 is much shorter than the one with none optimization. |  |  |
| Optimization 2:mergeSort function:  the assembly code with O2 is much shorter than the one with none optimization. |  |  |