SE 3F03 — Assignment 3

Ned Nedialkov

26 February 2015

Due date: 10 March

Problem 1 (10 points) You are given the `shellsort` function

```
void shellsort(int *A, int n)
{
  int gap, i, j, temp;
  for (gap = n/2; gap > 0; gap /=2)
    for (i = gap; i < n; i++)
      for (j = i - gap; j >= 0 && A[j] > A[j + gap]; j -= gap)
        {
          temp = A[j];
          A[j] = A[j + gap];
          A[j + gap] = temp;
        }
}
```

This code implements the shell sort algorithm; see e.g. `http://en.wikipedia.org/wiki/Shell_sort`.

a. (6 points) Write a NASM assembly version of it.

b. (4 points) Document the assembly code well by inserting appropriate comments.

Problem 2 (9 points) Write a main program that, for $n = 10,000,000$, $n = 20,000,000$, $\ldots$, $n = 100,000,000$,

a. generates an array of n random integers using the C function `rand()`

b. sorts them with the NASM version of `shellsort` and outputs the CPU time that it takes to sort.

   To measure the time, you can use the C `clock()` function.

c. after the timing is done, call the function

```
void checkSorting(int *A, int n)
{
  /* check if the sorting is correct */
  int i;
  for (i = 0; i < n-1; i++)
    if (A[i] > A[i+1])
      {
        printf("␣Error␣in␣sorting\n");
        exit(-1);
      }
}
```

Repeat 1–3 with the C version of `shellsort` and quick sort from the standard C library. See `man qsort`. When compiling, use the `-O2` option.

Problem 3 (9 points) In one plot, plot the CPU time versus n for the above three timings.

a. (2 points) Explain why `qsort` is so much faster.

b. (2 points) Can you explain why your assembly version is slower/faster than the one generated from the C code by the compiler.

c. (5 points) Shellsort runs in $O(n^2)$. Can you find the constants in the Big-O notation. That is, determine from your timing results α and β in αn^β.

Submit

- Hard copy of your programs
- All your programs to SVN under directory `A3`.

  When make is typed, an executable with name `a3` must be created. When `./a3` is executed, it should output on the screen in the form

```
  10000000     3.58e+00
  20000000     8.65e+00
  30000000     1.32e+01
  40000000     2.16e+01
  50000000     2.49e+01
  60000000     3.16e+01
  70000000     3.74e+01
  80000000     5.43e+01
  90000000     4.83e+01
 100000000     5.71e+01
```