CS 325: Assignment 1

Jared Wasinger

October 3, 2016

1. Intersection of $8n^2$ and $64nlog_2n \approx 43.5593$. Insertion sort will beat merge sort at $n >= 44$

2. Table:)

3. **Base Case**(n=2):
   $2lg2 = 2$
   Inductive Step:
   $T(2^{k+1}) = 2^{k+1}lg(2^{k+1})$
   $T(2^{k+1}) = 2^k * 2 * lg(2^k * 2)$
   $T(2^{k+1}) = 2^k * 2 * lg(2^k) * 2lg2$
   $T(2^k * 2) = 2^k * 2 * lg(2^k)$
   $T(2^k) = 2^k * lg(2^k)$
   $T(2^{k+1})$ implies $T(2^k)$

4. Answers:

   (a) $\lim_{x\to\infty} f(x)/g(x) = \lim_{x\to\infty} n^{0.75}/n^{0.5} = \lim_{x\to\infty} n^{0.25} = \infty$
   $f(n) = \Omega(g(n))$

   (b) $\lim_{x\to\infty} f(x)/g(x) = \lim_{x\to\infty} \frac{n}{log^2 n} = \infty$ (L.H. doesn't simplify result)
   $f(n) = \Omega(g(n))$

   (c) $\lim_{x\to\infty} f(x)/g(x) = \lim_{x\to\infty} \frac{log(n)}{log_2 n} = log(2)$ $f(n) = \Theta(g(n))$

   (d) $\lim_{x\to\infty} f(x)/g(x) = \lim_{x\to\infty} \frac{e^n}{2^n} = \infty$ $f(n) = \Omega(g(n))$

   (e) $\lim_{x\to\infty} f(x)/g(x) = \lim_{x\to\infty} \frac{e^n}{2^n} = \infty$ $f(n) = \Omega(g(n))$

   (f) $\lim_{x\to\infty} f(x)/g(x) = \lim_{x\to\infty} \frac{2^n}{2^{n-1}} = \lim_{x\to\infty} 2^{n-(n-1)} = 2$ $f(n) = \Theta(g(n))$
   $f(n) = \Omega(g(n))$

5. Algorithm:

   (a) Split array into pairs of consecutive values

   (b) Sort pair elements into local minima maxima (2 arrays): n/2 comparisons

(c) Compare all local minima (associatively) to find global minimum: n/2 comparisons

(d) Compare all local maxima (associatively) to find global maximum: n/2 comparisons

Worst case performance: 1.5n comparisons

Example: A=[9,3,5,10,1,7,12], n=7

(a) (9,3), (5,10), (1,7), (12)

(b) Local Minima: [3,5,1,12], Local Maxima: [9, 10, 7, 12] = 3 comparisons

(c) Global Maximum: 12 = 3 comparisons

(d) Global Minimum: 1 = 3 comparisons

(e) Total number of comparisons: 9. 9/7 = 1.28n comparisons

6. Proofs:

7. Results:

(a)

n (recursive)	**time (recursive)**	**n (iterative)**	**time (iterative)**
10	0.01s	20000	0.01s
15	0.01s	30000	0.01s
20	0.01s	40000	0.02s
30	0.26s	50000	0.03s
35	2.89s	60000	0.04s

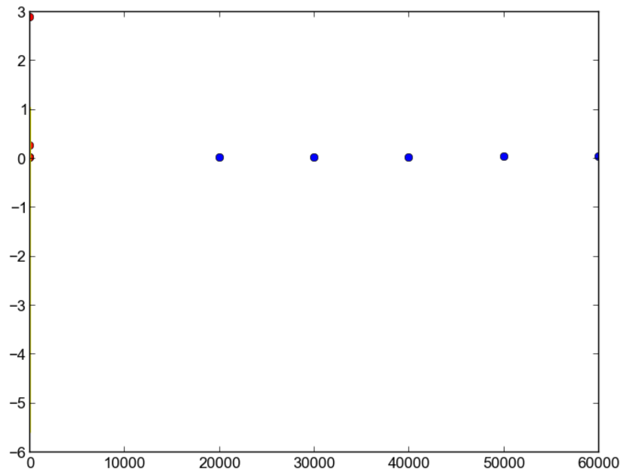
(b) **Matplotlib graphing source code**:

```
#! /bin/python

import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)

n_recur = [10, 15, 20, 30, 35]
time_recur = [0.01, 0.01, 0.01, 0.26, 2.89]

n_iter = [20000,30000,40000,50000,60000]
time_iter = [0.01, 0.01, 0.02, 0.03, 0.04]

fit_recur = np.polyfit(n_recur, time_recur, deg=3)
fit_iter = np.polyfit(n_iter, time_iter, deg=3)

plt.plot(n_recur, time_recur, 'ro')
plt.plot(n_iter, time_iter, 'bo')

plt.plot(fit_iter, color='green')
plt.plot(fit_recur, color='yellow')

plt.show()
```

Fibonacci profiling source code:

```
#! /bin/python

import numpy as np
import matplotlib.pyplot as plt
import time
import pdb

def fib_iter(n):
  fib = 0
  a = 1
  t = 0
  for k in range(1, n):
    t = fib + a
    a = fib
    fib = t
  return fib

def fib_recur(n):
 if n == 0:
   return 0
 elif n == 1:
   return 1
 else:
   return fib_recur(n-1) + fib_recur(n-2);

def compare_fib():
  n_vals = [5000, 10000, 20000, 30000, 40000, 50000]
  output = []

  for n in n_vals:
    start_time = time.clock()
    fib_iter(n)
    delta_time_iter = time.clock() - start_time

    start_time = time.clock()
    fib_recur(0)
    delta_time_recur = time.clock() - start_time
```

```
        output.append({'n':n, 'time_iter': delta_time_iter, 'time_recur': de
    return output

output = compare_fib()
```



(c)

Polynomial equations of fit
Polynomial equations are the best fit for these functions because they can be made to approximate the data set to an arbitrary precision.
Recursive fit: $1.056 \times 10^{-3}x^3 - 6.003 \times 10^-2x^2 + 1.048x - 5.584$
Iterative fit: $-8.333 \times 10^{-16}x^3 + 1.143 \times 10^-10x^2 - 4.060 \times 10^-6x - 5.20 \times 10^-2$
Note: These curves of fit do not match the data set. According to error output I received when running the line-of-fit calculations, this appears to be a numerical error, likely due to the fact that data points are so close to 0.

(d) The difference in running times is due to the immense (relative) overhead caused by the recursive implementation of the Fibonacci calculations. The additional computation cost imposed by recursive function calls (call stack manipulation) is very restrictive.