Collin Bauer

CSCI 310 - Advanced Algorithms

Assignment 1

Please forgive the makeshift notation. This is the best I can do with current tools. I may switch to LaTeX to avoid this in the future.

2.1

9.

For each of the following pairs of functions, indicate whether the first function of each of the following pairs has a lower, same, or higher order of growth (to within a constant multiple) than the second function.

- a) Same order of growth:

  > $\lim_{n \to \infty} n(n + 1) / 2000n^2$
  > $\approx \lim_{n \to \infty} n^2/2000n^2 = 1/2000 = \text{Constant}$

- b) $100n^2$ grows slower than $100n^2$ $0.01n^3$:

  > $\lim_{n \to \infty} 100n^2 / 0.01n^3$
  > $= \lim_{n \to \infty} 100 / 0.01n = 0$

- c) Same order of growth:

  > $\lim_{n \to \infty} \log_2(n) / \ln(n)$
  > $\lim_{n \to \infty} (\ln(n)/ (\ln(n) * \ln(2)) = 1/\ln(2) = \text{Constant}$

- d) I don't like this notation *at all*. But, if I'm interpreting them correctly...
  $\log_2^2(n)$ grows faster than $\log_2(n^2)$:

  > $\lim_{n \to \infty} \log_2^2(n) / \log_2(n^2)$
  > $= \lim_{n \to \infty} \log_2(n)^2 / 2\log_2(n)$
  > $= \lim_{n \to \infty} \log_2(n) / 2 = \infty$

- e) Same order of growth:

  > $\lim_{n \to \infty} 2^{n - 1} / 2^{n}</sup = \lim_{n \to \infty} 2^{n}/ 2^{n + 1} = 1/2 = \text{Constant}$

- f) Skipping for now...

2.2

6.

- a)
- b)

2.3

1.

Compute the following sums.

- b) 2046
- c) $SUM_{[i = 3 \rightarrow n + 1]}(1)$
  $= 1 + 1 + \ldots + 1$
  $= n - 2$
- d) $SUM_{[i = 3 \rightarrow n + 1]}(i)$
  $= 3 + 4 + 5 + \ldots + n + (n + 1)$
  $= n^2 + n - 2 = (n + 2)(n - 1)$
- e) $SUM_{[i = 0 \rightarrow n - 1]}(i + 1)$
  $= 2 + 3 + 4 + \ldots + (n - 2) + (n - 1)$
  $= n^2 - n - 2 = (n - 2)(n + 1)$
- g) $SUM_{[i = 1 \rightarrow n]}(SUM_{[j = 1 \rightarrow n]}(ij))$
  $= SUM_{[i = 1 \rightarrow n]}(i + 2i + 3i + \ldots + ni)$
  $= SUM_{[i = 1 \rightarrow n]}(in^2)$
  $= n^2 + 2n^2 + 3n^2 + \ldots + n^3$
  $= n^4$
- h) $SUM_{[i = 1 \rightarrow n]}(1/i (i+1))$ <= *Ambiguous notation. I'm going to assume the (i+1) is suppsoed to be in the numerator.*
  $= SUM_{[i = 1 \rightarrow n]}((i + 1)/i)$
  $= SUM_{[i = 1 \rightarrow n]}(i/i + 1/i)$
  $= SUM_{[i = 1 \rightarrow n]}(1 + 1/i)$
  $= (1 + 1) + (1 + 1/2) + (1 + 1/3) + \ldots + (1 + 1/n)$
  $= n + \ln(n)$

5.

- a) This algorithm computes the the the difference of the maximum and minimum values, or the *range*, of a data set.
- b) Its basic operation is comparing values to the maxval and minval variables.
- c) The basic opperation happens about 2 times the number of values in the set, or 2(n - 1).
- d) The efficiency class is O(n).

- e) The algorithm is already efficient.

6.

- a) The algorithm checks a matric and returns whether it is symmetric.
- b) Its basic operation is comparing A[i,j] to A[j,i].
- c) The basic operation happens n^2 times
- d) The efficiency class is $O(n^2)$
- e) The algorithm is already efficient.

2.4

1.

- c) x(1) = 0, x(2) = 0 + 5 = 5, x(3) = 5 + 5 = 10
  x(n) = 5n - 5
- e)

3.

- a) $S(n) = S(n - 1) + n^3$
  $S(1) = 1$
  ...
  $S(2) = 1 + 2^3$, $S(3) = (1 + 2^3) + 3^3$ $S(n) = n^4 - n^3$
- b) The efficiency of the more straightforward algorithm is O(n), since the basic operation of adding cubes happens n times. The recursive algorithm, by comparison, still has the same basic operation and complexity of O(n), but its efficiency in practice may not be as good due to the small tax recursion has on computers.

9.

- a) This algorithm computes the smallest element in an array.
- b) R(1) = A[0]
  R(n) = R(n - 1)

10.