CSCD320 Homework1, Winter 2012, Eastern Washington University. Cheney, Washington.

Name: Eric Fode **EWU ID:005301214**

Solution for Problem 1 The diffrence between data structures and algorithms is the difference between form and function. Data structures gives your data form, it determines how you store your data. Algorithms determine how you work with your data. They are related concepts in that all algorithms operate on some data structure even it is as simple as a list.

Solution for Problem 2

Proof. Show that $3n^2 + n\sqrt{n} = O(n^2)$

$$\lim_{n\to\infty} \frac{3n^2 + n\sqrt{n}}{n^2} = 3$$

so

$$3n^2 = O(n^2)$$

$\square$

Solution for Problem 3

Proof. Show that $2(n + 100\sqrt{n})(log(n))^2 = o(\frac{n\sqrt{n}}{log(n)})$

$$\lim_{n\to\infty} \frac{2(n + 100\sqrt{n})(log(n))^2}{\frac{n\sqrt{n}}{log(n)}} = 0$$

so

$$2(n + 100\sqrt{n})(log(n))^2 = o(\frac{n\sqrt{n}}{log(n)})$$

$\square$

CSCD320 Homework1, Winter 2012, Eastern Washington University. Cheney, Washington.

Name: Eric Fode **EWU ID:005301214**

Solution for Problem 4

Proof. it will always be true that $f(n) \leq g(n)$ or that $g(n) \leq f(n)$ so we can say that

$$2(\max\{f(n), g(n)\}) \geq f(n) + g(n)$$

givin that defenition of θ implies that

$$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

if

$$\lim_{n\to\infty} \frac{2\max\{f(n), g(n)\}}{\max\{f(n), g(n)\}} = 2$$

then

$$\lim_{n\to\infty} \frac{f(n) + g(n)}{\max\{f(n), g(n)\}} \leq 2$$

because

$$\frac{2\max\{f(n), g(n)\}}{\max\{f(n), g(n)\}} \geq \frac{f(n) + g(n)}{\max\{f(n), g(n)\}}$$

a

$$\lim_{n\to\infty} \frac{f(n) + g(n)}{\max\{f(n), g(n)\}} > 0$$

because

$$\lim_{n\to\infty} \frac{f(n) + g(n)}{f(n)} > 0 \text{ and } \lim_{n\to\ \infty} \frac{f(n) + g(n)}{g(n)} > 0$$

so by definition $f(n) + g(n) = \theta(\max\{f(n), g(n)\})$

□

Solution for Problem 5

Proof. is $f(n) = \omega(f(\sqrt{n}))$ always true?
$f(n) = \omega(f(\sqrt{n}))$ is shown by

$$\lim_{n\to\infty} \frac{n}{\sqrt{n}} = \infty$$

so as long as $n_0 > 0$ it is true $f(n) = \omega(f(\sqrt{n}))$

□

CSCD320 Homework1, Winter 2012, Eastern Washington University. Cheney, Washington.

Name: Eric Fode **EWU ID:005301214**

Solution for Problem 6

1. Pseudocode:

max($a, lower, upper$)

Input: A Section of an array a with bounds at $lower$ and $upper$

Result: The largest number in a will be returned

begin
- **if** $upper \neq lower$ **then**
  - $mid \longleftarrow ((upper - lower)/2) + lower$;
  - **return** $\max\{max(a, lower, mid), max(a, mid + 1, upper)\}$;
- **else**
  - **return** $lower$;

2. θ **analysis using tree** The recurance for that discribes this algroithm is

$$T(N) = 2T(N/2) + 1$$

There will be $log(n)$ layers to the tree
The sum of each layer will be 2^d where d is the depth of the layer// So the bottom in terms of n will be

$$\begin{aligned} 2^{log_2 n} &= \\ n^{(log_2)} &= \\ &= n \end{aligned}$$

the summation is

$$2^0 + 2^1 + 2^2 + ... + n$$

$$\sum_{i=0}^{log2(n)} 2^i = 2n - 1$$

so O is

$$O(n)$$

3. Asymptotic Comparison
Asymptotically the algoritims are the same

4. Reality Comparison
In practice the d-c algorithm will be half as fast as the simple method because it is doing twice as many comparisons

CSCD320 Homework1, Winter 2012, Eastern Washington University. Cheney, Washington.

Name: Eric Fode **EWU ID:005301214**

Solution for Problem 7

1. **Towers of Hanoi** The problem is divided smaller steps that consist of moving the nth ring to the middle peg then to the far right peg, n is incresed from the smallest ring to the largest until all of the rings have been moved

2. **Multiplication of numbers** For a givin x and y, x and y is broken into two parts x_h and x_l where the high and low parts are each half of the number represented as an array then the algorithim recureses with the paramaters a x array and a y (four times total) when the arrays reach a size of one they are multiplyed to together then combied with bit shifts.

3. **Finding nth greatest number** This algorithim spilts the array in to three parts,a greater, less then, and equal to then some number (in the array) then counts the number of items in each part and then if the count of items in the less then array is less then then n (the place of the number a sorted version of the array) then it recureses and runs the algorithim again on the less then array, if it is greater then the less then count it adds the count of equals items to the count if n is now less then the count then it recurses on to the equals array with n = n - count of items in the less then array . If none of these are true then it recurses onto the greater then array with n = n- the count of all items not in the greater then array.