CSCD320 Homework1, Spring 2014, EasternWashington University. Cheney,Washington.

Name: Brandon Fowler **EWU ID:** 00639348 **Due:** 11:59pm, April 13, 2014 (Sunday)

Please follow these rules strictly:

1. Write your name and EWUID on **EVERY** page of your submission.

2. Verbal discussions with classmates are encouraged, but each student must independently write his/her own solutions, without referring to anybody else's solution.

3. The deadline is sharp. Late submissions will **NOT** be accepted (it is set on the Canvas system). Send in whatever you have by the deadline.

4. Submission must be computer typeset in the **PDF** format and sent to the Canvas system. I encourage you all to use the LATEX system for the typesetting, as what I am doing for this homework as well as the class slides. LATEX is a free software used by publishers for professional typesetting and are also used by nearly all the computer science and math professionals for paper writing.

5. Your submission PDF file must be named as:
firstname_lastname_EWUID_cscd320_hw1.pdf
(1) We use the underline '_' not the dash '-'.
(2) All letters are in the lower case including your name and the filename's extend.
(3) If you have middle name(s), you don't have to put them into the submission's filename.

6. Sharing any content of this homework and its keys in any way with anyone who is not in this class of this quarter is NOT permitted.

Note:

• Take every homework as an opportunity to train yourself to be a better professional computer scientist. That is, use professional language and write out what you have in your mind precisely and concisely. The textbook is a good example of excellent computer science professional writing.

• In this homework, we use log to denote $\log_2$ outside of any asymptotic notation, and n assumes to only take natural numbers.

Problem 1 (5 points)**.** *Based on your learning from the CSCD300 Data Structures course, describe your understanding of the connection and difference between the "data structures" and "algorithms". Say your opinions in your own language. Any reasonable opinion is welcome.*

To my knowledge, a data structure is a method of storing or separating data in a certain way, such as the use of an array or a tree. Whereas, an algorithm is a means by which to organize or manipulate data within a data structure. For example, using the insertion sort algorithm to organize data, within an array data structure. Data structures store data in an organized fashion. Whereas, algorithms use data to solve a problems. Algorithms can also be used to build or traverse data structures.

CSCD320 Homework1, Spring 2014, EasternWashington University. Cheney,Washington.

Name: Brandon Fowler **EWU ID:** 00639348 **Due:** 11:59pm, April 13, 2014 (Sunday)

Problem 2 (15 points)**.** *Prove:* $\frac{n^2}{logn} + (10^5)n\sqrt{n} = O\left(\frac{n^2}{logn}\right)$

$\frac{n^2}{logn} + (10^5)n\sqrt{n} \leq C\left(\frac{n^2}{logn}\right)$ → $1 + \frac{(10^5)n\sqrt{n}}{\left(\frac{n^2}{logn}\right)} \leq C$

→ $1 + \frac{logn(10^5)n\sqrt{n}}{n^2} \leq C$ → $\frac{logn(10^5)\sqrt{n}}{n} \leq C$

Let n0 = 2, → 1 + 70710.68 ≤ C → So choose a greater value for C, such as C = 80000

For n0 = 2, C = 80000, and any n ≥ n0, C will always be greater.

Therefore: $\frac{n^2}{logn} + (10^5)n\sqrt{n} = O\left(\frac{n^2}{logn}\right)$

Problem 3 (15 points)**.** *Prove:* $2(n + 100\sqrt{n})\log^2 n = o(n\sqrt{n})$

$2(n + 100\sqrt{n})\log^2 n \leq C(n\sqrt{n})$ → $\frac{2\log^2 n(n+100\sqrt{n})}{n\sqrt{n}} \leq C$ → $\frac{2\log^2 n(n+100\sqrt{n})(n\sqrt{n})}{n^3} \leq C$

→ $\frac{2\log^2 n\sqrt{n}(n+100\sqrt{n})}{n^2} \leq C$ → $\frac{2n\log^2 n\sqrt{n}+2\,00n\log^2 n}{n^2} \leq C$ → $\frac{2\log^2 n\sqrt{n}+2\,00\log^2 n}{n} \leq C$

No matter what value is chosen for C, we can always find a positive value n0, such that when n ≥ n0, the left side becomes less than the right, because the left side is going towards 0 when n goes towards infinity.

Problem 4 (15 points)**.** *Let* f(n) *and* g(n) *be asymptotically nonnegative increasing functions. Prove: (f(n) + g(n)) / 2 = Θ(max { f(n), g(n) }), using definition of Θ.*

$(c1)\max\{f(n), g(n)\} \leq \frac{1}{2}(f(n) + g(n)) \leq (c2)\max\{f(n), g(n)\}$

The middle piece will always have the same degree as the outer two pieces, and therefore differs by at most one constant factor, making the pieces asymptotically equal. Since this is the case, there does exist a c1, c2, and n0, such that with any n ≥ n0 the above inequality will always be satisfied.

Therefore: *(f(n) + g(n)) / 2 = Θ(max { f(n), g(n) })*

Problem 5 (15 points)**.** *Let* f(n) *be an asymptotically nonnegative increasing function. Is f(n) = o(f(n^2)) always true? Justify your answer.*

$f(n^2)$ will always be growing much faster than f(n). Since this is the case, for any constant C, there exists a positive constant n0, such that when n ≥ n0,
$0 \leq f(n) \leq C * f(n^2)$

Therefore: Yes, f(n) = o(f(n^2)) is always true.

CSCD320 Homework1, Spring 2014, EasternWashington University. Cheney,Washington.

Name: Brandon Fowler **EWU ID:** 00639348 **Due:** 11:59pm, April 13, 2014 (Sunday)

Problem6 (20)**.** *Suppose you are given an array* A *with* n *entries, with each entry holding a distinct number. You are told the sequence of values* A[1],A[2], . . .,A[n] *is* **unimodal***: For some index* p *between* 1 *and* n*, the values in the array entries increase up to position* p *in* A *and then decrease the remainder of the way until position* n*. That is,* A[1] < A[2] < ... < A[p] *and* A[p] > A[p + 1] > ... > A[n]*, for some index* p " {1, 2, . . . , n}*.* **Your task:** *find the "peak entry"* p *without having to read the entire array—in fact, by reading as few entries of* A *as possible. Show how to find the entry* p *by reading* O(log n) *entries of* A*. Describe your algorithmic idea, show its pseudocode, and explain why your algorithm's time complexity is* O(log n)*, where* n *is the input array size.*

To begin, I would divide the array in half by choosing the index in the middle. I would then check the index to the right of the middle; if the number stored there is higher than the number stored at my middle index, then I would move on from the first half of the array, and continue only with the second half. If the number stored to the right is smaller, then I would check the number stored to the left, and if it was larger, then I would ignore the second half of the array and work only with the first half. I would then split the array again in the half that I continued with, and perform the same process. I would continue in this manner, splitting the array into smaller pieces at the middle index, until the number stored in the index chosen, was bigger than both the number stored to the right and the number stored to the left of it. This index would be the location of the "peak entry".

Sudo code:

```
int array = unimodel array of numbers stored in indexes 0 to n;
int result = findPeak(array, floor(n/2));


public int findPeak(int array, int m)
{
    int result;
    if(array[m] < array[m+1])
        result = findPeak(array, floor(m+(m/2)));
    elseif(array[m] > array[m-1])
        result = findPeak(array, floor(m/2));
    else
        result = array[m];
    return result;
}
```

Since I am effectively cutting the array in half over and over recursively until I find the desired index, then I am just returning the value stored there; when n is the input size of the array and k is the number of times we split the array, I can derive from this information the formula: $\frac{n}{2^k} = 1$, where solving for k gives time complexity. To simplify: $\frac{n}{2^k} = 1 \rightarrow 2^k = n \rightarrow k = logn$. So the time complexity of this algorithm is O(log n).

CSCD320 Homework1, Spring 2014, EasternWashington University. Cheney,Washington.

Name: Brandon Fowler **EWU ID:** 00639348 **Due:** 11:59pm, April 13, 2014 (Sunday)

Problem 7 (15 points total; 5 points for each algorithm.). *Search and learn three existing algorithms that use the divide-conquer strategy (not those covered in the class and/or the textbook, such as merge sort, max subarray finding algorithm, and Strassen's matrix multiplication algorithm). For each algorithm, in your own language, concisely and clearly describe:*

1. the problem statement

2. the algorithmic idea in the solution (don't just copy the code to me)

3. the time complexity

4. the condition, on which the worst-case running time appears.

5. the source of your finding. For example, the url of the webpages, the title and page of a book, the title/author/year of an article, etc.

6. Note: if you just copy and paste with minor change in wording but without understanding and writing in your own language, you will get zero for this problem.

1. Quick Sort is an example of a divide and conquer algorithm that is used to solve the problem of sorting a list of comparable elements. It works by choosing a starting point in the list(usually the first element or the middle) to act as a pivot point. Items in the list will then be shifted depending on whether they are higher in value than the pivot or lower in value than the pivot, and placed on separate sides accordingly. The algorithm will then move on to each side recursively sorting smaller and smaller pieces by choosing a pivot, then shifting the values around it, until the entire list is sorted. The time complexity of this algorithm is O(n), and the worst case running time occurs when either the highest or lowest values are consistently chosen as the pivot points for each sub piece of the list. My source for this algorithm is: https://www.cs.auckland.ac.nz/~jmor159/PLDS210/qsort.html

2. Binary Search is another divide and conquer algorithm used to solve the problem of finding a certain specified comparable item from a list of other comparable items. It works by dividing the list into smaller and smaller pieces until it finds the item that is specified. Binary Search assumes that the list is already sorted, and when the array is divided it will check the value stored at the middle index for a match to the specified item. If it doesn't find a match it simply has to compare the data directly on each side of the split to choose which piece of the list to continue searching. For example, if the data one index higher in the list is greater than the specified item, then the algorithm will continue searching in the lower half of the array by splitting it again, and performing the same process until the specified item is found. The time complexity of this algorithm is O(log n), and the worst case running time occurs when the algorithm has to split the list until only one item(the specified item) is remaining. My research source for this algorithm is:
http://homepages.ius.edu/RWISMAN/C455/html/notes/Chapter2/DivConq.htm

3. Karatsuba's Algorithm, is a divide and conquer algorithm used to solve the problem of multiplying two multi digit numbers. It works by breaking the two numbers into smaller and smaller pieces until only small steps with single digit multiplication are left. After the single digit multiplication, the pieces are recombined via addition to attain the final solution to the original multiplication problem. The time complexity of this algorithm is $\Theta(n^{log3})$. The time complexity is consistent, so there isn't a worst case running time. My source for this algorithm is: http://en.wikipedia.org/wiki/Karatsuba_algorithm