CSCD320 Homework1, Eastern Washington University, Spokane, Washington.

Name: Matthew Adams EWU ID: Madams41

Please follow these rules strictly:

1. Verbal discussions with classmates are encouraged, but each student must independently write his/her own solutions, without referring to anybody else's solution.
2. The deadline is sharp. Late submissions will NOT 3be accepted (it is set on the Canvas system). Send in whatever you have by the deadline.
3. 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 lecture slides. LATEX is a free software used by publishers for professional typesetting and are used by almost all computer science and math professionals for paper writing. But of course you are also allowed to use other software for editing (for ex., Microsoft Office Word) but save and submit the file as a PDF.
4. 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.
5. 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: In this homework, as well as in those of the rest of the quarter, *n* assumes 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.*

Based on what we have gone over in class, Data Structures taught the implementation of the data storage methods. Algorithms is about looking at these different methods and understanding their time complexity and how to make them more efficient.

Problem 2 (10 points). *Show:* $5000n^2 + n\log n = O(n^2)$.

To show $5000n^2 + n\log(n) = O(n^2)$, I need to show that $5000n^2 + n\log n \leq c \cdot n^2$ is true for all $n \geq n_0$. $5000n^2$ is the dominating term so the nlogn can be ignored, meaning I need to find $5000n^2 \leq c \bullet n^2$. If I am understanding this correctly c would be 5000 or more as 5000 is the constant on the left side of the equation, and n_0 would be 1 or more, if c is 5000 or more the equation on the right side would be growing faster at any n.

Problem 3 (10 points). *Show:* $5000n^2 + n\log n = o(n^3)$.

To show $5000n^2 + n\log n = o(n^3)$, I need to prove $5000n^2 + n\log n < c \bullet n^3$. $5000n^2$ is the dominating term so the nlogn can be ignored. That means I need to find a c and a n_0 that proves $5000n^2 < c \cdot n^3$. Therefore, just like in the previous c would be 5000. Having c be 5000 would make it so that any $n > 1$ would make the equation true

Problem 4 (10 points). *Show:* $5000n^2 + n\log n = \Omega(n^2)$.

To show this equation I need to prove $5000n^2 + n\log n \geq c \cdot n^2$ for every $n > n_0$. Just like with the previous equations, $5000n^2$ is the dominating term so the n log n can be ignored. That means I need to find a c that proves $5000n^2 \geq c \bullet n^2$. Since the equations are almost identical if c was 5000, they would be always equal. Therefore, any $c \leq 5000$ would make this true for every positive $n > 0$.

Problem 5 (10 points). *Show:* $5000n^2 + n\log n = \omega(n)$.

To show this function I would need to find a c and an n_0 that proves $5000n^2 + n\log n > c \cdot n$. Just like the previous equations, $5000n^2$ is the dominating term in this equation. This means that the nlogn can be ignored. That brings our equation is $5000n^2 > c \bullet n$. That said the left side of the equation is exponential. You would have to set $c > 5000$ and n_0 as 1

Problem 6 (15 points). *Let $f(n)$ be a nonnegative increasing function. Is $f(n) = \Theta(f(2n))$ always true? If yes, prove it; otherwise, give a counter example. Hint: try many different possible functions for f.*

To prove that $f(n) = \Theta(f(2n))$ is always true we need to look at functions where it might be false. One function that comes to mind is something that might be exponential, something like f(n)=2n. For this function f(2n) would be $f(2n)=2^{2n}$. This means that when we are not able to find constant c1, c2 & $n \geq n_0$ that makes our definition of $c_1 \cdot 2^{2n} \leq 2n \leq c_2 \cdot 2^{2n}$ true

Problem 7 (25 points). *The idea of* Merge Sort *that we have discussed in the class is to split the input sequence of size n into two subsequences of each sized n/2, recursively sort each subsequence, and merge the two sorted subsequences into a sorted version of the original sequence. Now, someone proposed the following new idea: why don't we split the input sequence into more subsequences of smaller size, so that the algorithm can reach the exit condition (which is when the sequence size becomes 1) more quickly and thus make the algorithm faster? Your job:*

1. *Change the algorithm and give the pseudocode for this proposed new* Merge Sort*, where the input sequence is divided into 4 subsequences of each sized n/4.*

   The sudo code would look something like this,

   ```
   Merge_Sort(A, p, r)
   {
   If(p<r)
     q = floor(p+r)/2
     q2 = floor(p+r)/4
     q3 = 3* floor(p+r)/3
   ```

```
    Merge_Sort(A, p, q)
    Merge_Sort(A, q+1, q2)
    Merge_Sort(A, q2+1, q3)
    Merge_Sort(A, q3+1, r)
    Merge(A, p, q, q2, q3, r)
}
```

2. *Analyze the time complexity of this new algorithm and present the result using the Θ notation.*

   The notes say the original mergsort has a time of $\log_2$. This new one would have a time of $\log_4$. In Θ notation it would be $T(n) = cn \log_4 n = \Theta(n\log n)$.

3. *Is this new algorithm asymptotically faster than the one in the textbook ? Justify your answer.*

   This new algorithm is not faster. In this notation we ignore constants. $n\log_2 n$ is an ignored constant, same as $n\log_4 n$. This would cause the algorithm to not be noticeably faster.

4. *Can this algorithm be faster in practice than the one in the textbook ? Justify your answer.*

   I don't think this algorithm would be faster in practice. You would be doing more math to find the four parts to split the array at and you would be doing a lot more copying and sorting when you eventually have to merge them all back together.

5. *Let's think of the extreme case. We split the input sequence into n subsequences of each sized just* 1*. Can this algorithm be asymptotically faster than the one in the textbook ? Justify your answer.*

   I don't think this algorithm would be asymptotically faster than the one in the textbook. We would still have the constant of logn, It would just be in base one instead of 2. That would not cause a major shift in the time complexity of the algorithm like a change from n^1 to n^2 would

6. *Do you get any insight why the textbook* Merge Sort *only splits the input sequence into two halves ?*

   I think the textbook only splits the input sequence into rwo halves because there is no noticeable difference in splitting it into more pieces asymptotically, but it would make more operations in practice and is not worth it.

Problem 8 (15 points total; 5 points for each algorithm.). *Search and learn three existing algorithms that use the divide-conquer strategy. For each algorithm, in your own language, concisely and clearly describe:*

Binary search

1. *the problem statement*

   The goal of a binary search is to find a specific element in a sorted array.

2. *the algorithmic idea in the solution (don't just copy the code or the text on the webpage to me)*

   The binary search approach is to have a key and look in the middle of the array. If the key value is bigger than the middle of the array then checks the right half, if it is smaller then check the left half

3. *the time complexity*

   The time complexity of binary search is O(logn)

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

   The condition, on which the worst-case is present is when the wanted item is in the very beginning or end of the array because it will take the maximum number of comparisons of 2^{n-1}

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.*

   https://www.geeksforgeeks.org/binary-search/

   https://www.geeksforgeeks.org/complexity-analysis-of-binary-search/

Quick sort

1. *the problem statement*

   The goal of quick sort is to sort an array

2. *the algorithmic idea in the solution (don't just copy the code or the text on the webpage to me)*

   The idea of quick sort is to pick a pivot element and split the array into things less than and greater than the pivot. That is then done recursively on each side until the array is sorted.

3. *the time complexity*

   The time complexity of quick sort is $T(N) = \log_2 N * (N + 1)$

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

   The worst case condition appears when the array keeps getting divided into two parts becoming $O(n^2)$

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.*

   https://www.geeksforgeeks.org/quick-sort/

   https://www.geeksforgeeks.org/time-and-space-complexity-analysis-of-quick-sort/

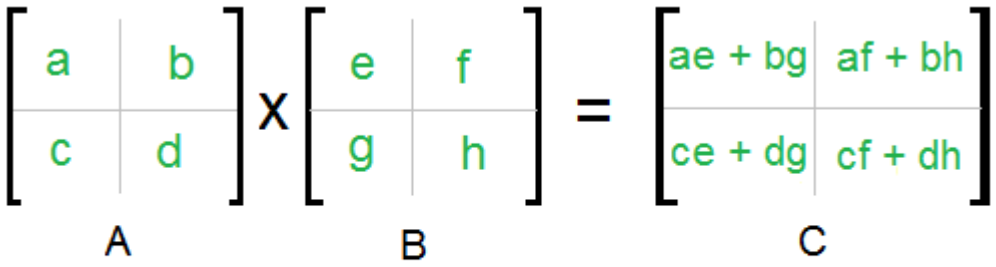
Strassen's Matrix Multiplication

1. *the problem statement*

The goal is to multiply two matrices.

2. *the algorithmic idea in the solution (don't just copy the code or the text on the webpage to me)*

   The Idea of the algorithm is to split each matrix into four sub matrices, then multiply the values in the subarray by the values in the sub array then add the different values. I think the picture from the website explains it better than words.



A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2

3. *the time complexity*

   From what I could find the time complexity ends up being $O(n^{log7})$

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

   From what I Can find the time complexity goes up the more recursive calls you have.

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.*

https://www.geeksforgeeks.org/strassens-matrix-multiplication/

Note: If you just copy and paste or with small modification without your own understanding, you will get zero for this problem.