Assignment 2 - Written Component

● Graded

Student

Tyler NGUYEN

Total Points

8 / 8 pts

Question 1

Question 1 — **2** / 2 pts

1.1 **Question 1a)** — **0.6** / 0.6 pts

✔ **– 0 pts** Correct

– 0.3 pts Missing/Wrong final answer.

– 0.2 pts Missing/Wrong calculation/table.

– 0.2 pts Cost per node is wrong.

– 0.6 pts Both the process and final answer are wrong.

– 0.6 pts No submission / Almost no work!

1.2 **Question 1b)** — **0.7** / 0.7 pts

✔ **– 0 pts** Correct

– 0.35 pts Missing/Wrong final answer.

– 0.25 pts Missing/Wrong calculation/table.

– 0.25 pts Cost per node is wrong.

– 0.7 pts Both the process and final answer are wrong.

– 0.7 pts No submission / Almost no work!

1.3 **Question 1c)** — **0.7** / 0.7 pts

✔ **– 0 pts** Correct

– 0.35 pts Missing/Wrong final answer.

– 0.25 pts Missing/Wrong calculation/table.

– 0.25 pts The height calculation is wrong.

– 0.7 pts Both the process and final answer are wrong.

– 0.7 pts No submission / Almost no work!

Question 2

Question 2

2 / 2 pts

✔ **– 0 pts** Correct

– 1 pt Good thinking path, but the wrong methodology was used.

– 0.5 pts The correct way, but miscalculated

– 0.5 pts Missing/Wrong comparison between three algorithms.

– 0.5 pts Missing/Wrong comparison to Strassen's algorithm.

– 1 pt Wrong justification.

– 2 pts No submission.

Question 3

Question 3

4 / 4 pts

3.1 **Question 3a)**

3 / 3 pts

✔ **– 0 pts** Correct

– 3 pts Wrong Answer

– 1 pt Partially correct. The given answer does not fully satisify question specification.

– 0.25 pts n/2 is not an integer which cannot be used as an index (as n is assumed to be odd in question).

– 1 pt Insufficient Justification on the complexity of the algorithm. You should show that the complexity is exactly O(logn)

– 0.5 pts Partially corret. The main idea is correct, but there is(are) trivial mistake(s).

– 1.5 pts Incomplete answer. The algorithm (or justification) is not complete.

– 1 pt No justification/expaination on the correctness of your algorithms

3.2 **Question 3b)**

1 / 1 pt

✔ **– 0 pts** Correct

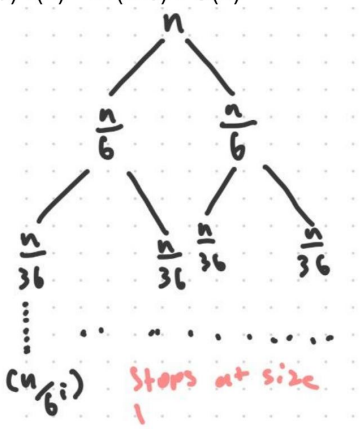
– 1 pt Wrong Answer

– 0.25 pts Partially correct.

– 0.5 pts Incomplete answer. You should show all the steps for computing the given example inputs with your algorithms.

Questions assigned to the following page: 1.1 and 1.2

CPSC 413 Assignment 2
Author: Tyler Nguyen
UCID: 30158563
Question 1:
a) T(n) = 2T(n/6) + O(n)



Level/layer	Number of nodes	Cost/node	Total cost
0	1	n	n
1	2	n/6	2(n/6)
2	4	n/36	4(n/36)
i	2^i	n/6^i	2^i(n/6^i)

Layers(h): $\frac{n}{6^h}$ = 1 => n = 6^h = $\log_6 n$ = layers.

T(n) = $\sum_{i=0}^{\log_6 n} 2^i \left(\frac{n}{6^i}\right)$ = n $\sum_{i=0}^{\log_6 n} \frac{2^i}{6^i}$ = n (c) = n (θ(1)) this is possible since we are dealing with asymptotic notation as mentioned in lecture. Therefore, this recurrence gives us O(n).

b) T(n) = T(n-1) + O(n)

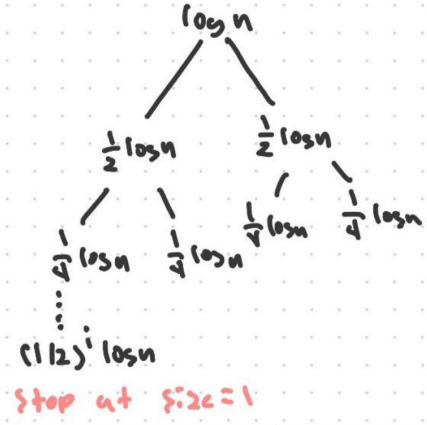
Level/layer	Number of nodes	Cost/node	Total cost
0	1	n	n
1	1	(n-1)	(n-1)
2	1	(n-2)	(n-2)

Questions assigned to the following page: 1.2, 2, and 1.3

i	1	(n-i)	(n-i)

T(n) = n + (n-1) + (n-2) + … + 1 = $\frac{n(n+1)}{2}$ = $\frac{n^2+n}{2}$ = n^2, this is possible since we are dealing with asymptotic notation mentioned before and in lecture. Therefore, this recurrence gives us O(n^2).

c) T(n) = 2T($\sqrt{n}$) + O(logn)



Level/layer	Number of nodes	Cost/node	Total cost
0	1	logn	logn
1	2	(1/2)logn	2((1/2)logn)
2	4	(1/4)logn	4((1/4)logn)
i	2^i	(1/2)^i logn	2^i((1/2)^i logn)

Layers(h): $n^{(\frac{1}{2})^h}$ = 1 => loglogn

T(n) = $\sum_{i=0}^{loglogn} 2^i * ((\frac{1}{2})^i \log n)$ = $\sum_{i=0}^{loglogn} (2^i(\frac{1}{2})^i \log n)$ = $\sum_{i=0}^{loglogn} \log n$ = log n $\sum_{i=0}^{loglogn} 1$ = log n(loglogn + 1) = logn(loglogn) + logn = logn(loglogn), this is possible since we are dealing with asymptotic notation as mentioned. Therefore, this recurrence gives us O(logn(loglogn)).

Question 2:

As stated Strassens algorithm will perform the matrix multiplication in $O(n^{\log_2 7})$ multiplications where this approximately evaluates to $O(n^{2.8074})$. In order to compare Dylan, Etienne and Francescas method to Stassens, we need to figure out the time complexity for their algorithms in regard to the same base as $n^{\log_2 7}$. This means we need to find the exponent "m" in the formula: $n^m = number\ of\ multiplications\ required\ (nomr)$, where n is the size of the matrices that is specific to each friends algorithm. We want it in this form since as mentioned during lecture, Strassens algorithm reduced the complexity of finger matrix multiplication from $O(n^3)$ to $O(n^{\log_2 7})$, where the exponents show how the number of multiplications scales with the size of the matrix size n. This also allows for direct comparison with Strassen since they take the same form of $O(n^m)$

Dylan:

As stated, Dylan found a way to multiply two 68x68 matrices using 132464 multiplications. Therefore, let n = 68 and nomr = 132464. So we get $68^m = 132464$, now solving for m we get $m = \frac{\log_2 132464}{\log_2 68} \approx 2.7951$so we get $O(n^{2.7951})$ for dylans algorithm when in comes to matrices of size 68.

Questions assigned to the following page: 3.1 and 2

Etienne:
As stated, Etienne found a way to multiply two 70x70 matrices using 143640 multiplications. Therefore, let n = 70 and nomr = 143640. So we get $70^m = 143640$, now solving for m we get $m = \frac{\log_2 143640}{\log_2 70} \approx 2.7951$so we get $O(n^{2.7951})$ for Etiennes algorithm when in comes to matrices of size 70.

Francesca:
As stated, Francesca found a way to multiply two 72x72 matrices using 155424 multiplications. Therefore, let n = 72 and nomr = 155424. So we get $72^m = 155424$, now solving for m we get $m = \frac{\log_2 155424}{\log_2 72} \approx 2.7951$ so we get $O(n^{2.7951})$ for Francesca algorithm when in comes to matrices of size 72.

From these results we can compare them to Strassens, where his exponent is 2.8074 and the friends had an exponent of 2.7951. Since Dylan, Etienne and Francesa methods all resulted in an exponent of roughly around 2.7951 which is less than Strassens 2.8074 we can say that their methods are more efficient than Strassens algorithms. This is because the lower the exponent displays a slower growth rate as the matrices increase. Therefore, the friends have the fastest matrix multiplication algorithm specifically for matrices of size 68, 70 and 72.

Question 3:
a) Algorithm:

1. Calculate the median of each array A and array B, let median A = mA and median B = mB
2. now if mA = mB, then mA or mB is the over all median
3. if mA < mB then the median is not in the first half of array A or the second half of array B, which then we can remove the first half of array A and the second half of array B and consider the second half of array A and the first half of array B
4. if mA > mB then the median is not in the second half of array A or the first half of array B, which then we can remove the second half of array A and the first half of array B and consider the first half of array A and the second half of array B
5. recurse on the new halves of the array
6. then when the arrays are reduced to two elements each, the median of the combined arrays is the second smallest of these four elements that can be found by comparing the maximum of the lower ends and the minimum of the upper ends of array A and B

This algorithm is correct since at each step it will remove half of the elements from consideration because from the comparisons it will know that the median cannot possibly be in the removed halves. Since the median is the middle element of in a sorted array. This means half of the elements are less than the median and half are greater than the median. When we have two arrays A and B that are sorted, their medians will divide each array into two halves with the same property. Once comparing medians, if mA < mB the algorithm will know that all the elements in the first half of array A are less than mA as well as less than mB. This means that all elements in the second half of array B are greater than mB therefore bigger than mA. This also means that the elements in the first half of array A and the second half of array B can never be the median of the combined arrays. This means we can remove them safely. If mA > mB then the opposite is true where we can remove the second half of A and the first half of B. By doing this recursively the algorithm finds the median in O(log n) time as desired since the problem size will be halved at each step. Where at the first

Questions assigned to the following page: 3.1 and 3.2

step the problem is reduced in two arrays of size n/2. After the second step the size becomes n/4. This division or halving continues until it reaches the base case which is when the arrays are small enough to find the median by simple comparisons which in this case is when the arrays both hold 2 elements.

b) step through for arrays A = [1,1,3,4,8] and B = [1,2,5,6,9]

- the initial median A = A[5/2] = A[2] = 3 and median B = B[5/2] = B[2] = 5 due to 0-based indexing and rounding down
- we then compare the medians mA = 3 < mB = 5
- since mA < mB we remove the first half of A and the second half of B which results in the new arrays to look like A' = [3,4,8] which is the second half of A and B' = [1,2,5] which is the first half of B
- now we must recurse on the new halves
- so we find the new medians, median A' = A'[3/2] = A'[1] = 4 and median B' = B'[3/2] = B'[1] = 2
- then we compare the new medians mA' = 4 > mB' = 2
- since mA' > mB' we can remove the second half of A' and the first half of B' which results in the new arrays to look like A'' = [3,4] which is the first half of A' and B'' = [2,5] which is the second half of B'
- now since we have 2 elements in each of the arrays, the median of the combined array is the second smallest element of these four elements.
- so we take the maximum of the lower ends which is the max of A''[0], B''[0] = max of 3,2 which is 3
- and we take the minimum of the upper ends which is the min of A''[1], B''[1] = min of 4,5 which is 4
- then the median is the average of these two numbers 3 and 4 since they would be the middle two numbers if we were to sort the final four elements
- the median = (3+4)/2 = 3.5 but since we are looking for a median in a sorted array of odd total length the median should be an integer, so in this case it would be the lower of the two which is 3 as stated in the algorithm where we are taking the second smallest element of the four.
- therefore, the median of the combined array is 3