Branden Vennes
CS 324
Homework #3
2/6/2019

Problem 1

Consider the (hypothetical) Jumping Juniper Problem. The following things are know about this problem:

- There is an algorithm that solves the problem in $\Theta(n^{2.5})$ time.
- There is another algorithm that solves the problem in $\Theta(n^2(\log(n))^3)$ time.
- There is a third algorithm that solves the problem in $\Theta(n^{1.5})$ in 99% of the cases, and in $\Theta(n^4)$ the other 1% of the cases.
- It has been proven mathematically that the problem cannot be solved in less than $\Theta(n\log(n))$ time.

What are the worst-case upper and lower bounds for the Jumping Juniper Problem? Justify your answer.

The worst-case upper bound for the problem is $\Theta(n^{2.5})$ because when n is greater than 2, $n^{2.5} < n^2(\log(n))^3$.

The worst-case lower bound for the problem is $\Theta(n\log(n))$ because it has been proven an algorithm for Jumping Juniper Problem can not be faster than $\Theta(n\log(n))$.

Problem 2

Use Strassen's algorithm to compute the matrix product
[1 3] [6 8]
[7 5] [4 2]
Show your work.

```
A = 1     E = 6
B = 3     F = 8
C = 7     G = 4
D = 5     H = 2

P1 = A(F-H) = 1*(8-2) = 6
P2 = (A+B)H = (1+3)*2 = 8
P3 = (C+D)E = (7+5)*6 = 72
P4 = D(G-E) = 5*(4-6) = -10
P5 = (A+D)(E+H) = (1+5)*(6+2) = 48
P6 = (B-D)(G+H) = (3-5)*(4+2) = -12
P7 = (A-C)(E+F) = (1-7)*(6+8) = -84

R = P5 + P4 – P2 + P6 = 48 + -10 – 8 + -12 = 18
S = P1 + P2 = 6 + 8 = 14
T = P3 + P4 = 72 + -10 = 62
U = P5 + P1 - P3 – P7 = 48 + 6 – 72 - -84 = 66

= [ 18 14 ]
  [ 62 66 ]
```

<u>Problem 3</u>
How would you modify Strassen's algorithm to multiply n x n matrices in which n is not an exact power of 2? Show that the resulting algorithm runs in time Θ($n^{\lg 7}$).

If n is not a power of two, we pad the missing values with rows and columns of zeroes until n is a power of two.

So if n = 3, we add an additional row and column containing zeroes. If n = 6, we add two rows and columns of zeroes so that n = 8.

We would then just perform Strassen's algorithm on the new padded matrix.

The resulting algorithm will still be $n^{\lg 7}$ because the number of rows/ columns required to add will be less than n.

Problem 4

What is the smallest possible depth of a leaf in a decision tree for a comparison sort?

The smallest possible depth of a leaf in a decision tree for comparison sort will be
n-1
where n is the number of elements in the list.

This case will occur when the list is already sorted. When this occurs, n-1 comparisons must be made in order to verify that the list is in sorted order.

<u>Problem 5</u>
Consider a 21-element array, a (1-based indexing), where all elements contain the value 1. The following code is then executed:

```
    parallel for i = 1 to 20
        a[i] = a[i] + a[i+1]
```

What is the largest value that a[1] can possibly have after the completion of the loop?

a[1] = 21 would be the largest possible value of a[1].

This would occur if the loop was executed in backwards order, meaning the statements were executed in the following order:

```
a[20] = a[20] + a[21] // a[20] = 2
a[19] = a[19] + a[20] // a[19] = 3
…
a[1] = a[1] + a[2] // a[1] = 21
```

What is the smallest value?

A[1] = 2 would be the smallest possible value of a[1].

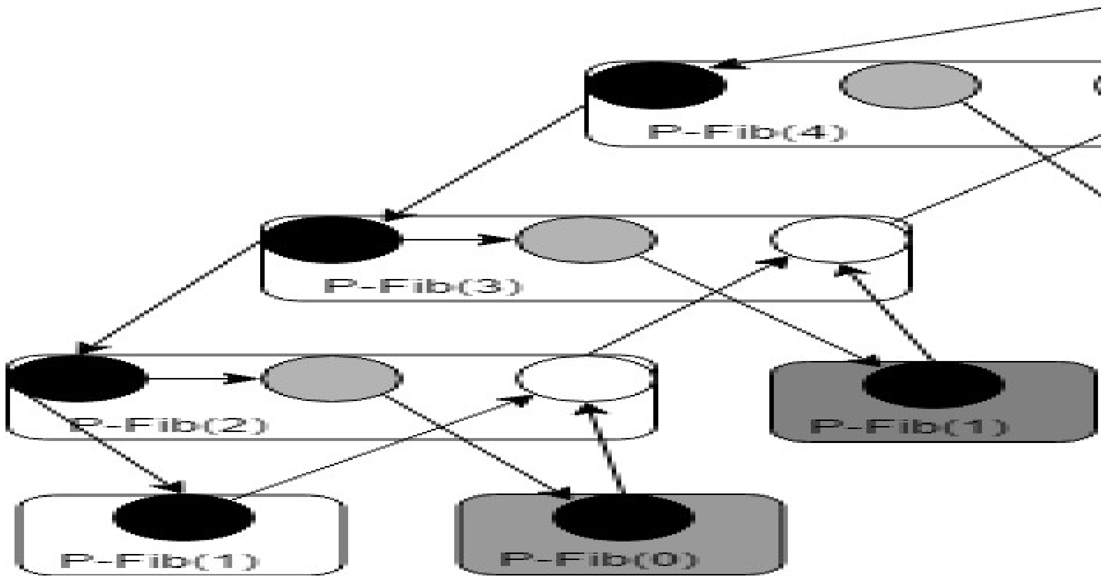
This would occur if a[1] = a[1] + a[2] was the first statement to be executed.

What is the largest value that a[20] can have after the completion of the loop? What is the smallest value? Explain your reasoning in all cases.

The largest and smallest value a[20] could have is 2. This is because a[21] and a[20] will always hold the value 1 before the value of a[20] is overwritten. Therefore, a[20] will always end up being equal to 2.

<u>Problem 6</u>
Draw the computation dag that results from executing P-FIB(5). Assuming that each strand in the computation takes unit time, what are the work, span, and parallelism of the computation? Show how to schedule the dag on 3 processors using greedy scheduling by labeling each strand with the time step in which it is executed.

Compute actual integers when you analyze the work and span, using the parameter given (5). Do not analyze asymptotically, or as a function of n. Consider the black and white circles to be the strands that required one unit of time to execute.



Work = 29

Span = 10

Parallelism = 29 / 10 = 2.9

<u>Problem 7</u>
Consider the following multi-threaded pseudo-code for transposing an n x n matrix A in place:

```
P-TRANSPOSE(A)
1    n = A.rows
2    parallel for j = 2 to n
3         parallel for i = 1 to j - 1
4              exchange $a_{ij}$ with $a_{ji}$
```

Analyze the work, span, and parallelism of this algorithm.

The total amount of work for this algorithm will be 3*(n-1)! because the iterator will run (n-1)! times and each exchange will take 3 steps (save a_{ij} to temp, copy a_{ji} to a_{ij}, copy from temp to a_{ji}).

The span of the algorithm is 3. This is because (n-1)! threads will be created and each thread will run 3 steps.

T_1 = 3*(n-1)!
T_∞ = 3

Slackness = $T_1/(P*T_\infty)$ = 3*(n-1)!/(3*P) = (n-1)!/P

The speedup of the algorithm is linear.