CMPS 6610 Problem Set 02

Name:_ Aaron Dumont_____

In this assignment we'll work on applying the methods we've learned to analyze recurrences, and also see their behavior in practice. As with previous assignments, some of of your answers will go in `main.py`. Please add your written answers to `answers.md` which you can convert to a PDF using `convert.sh`. Alternatively, you may scan and upload written answers to a file named `answers.pdf`.

1. Prove that $\log n! \in \Theta(n \log n)$.

We must show that $log(n!)$ is bounded above and below by a constant multiple of $nlogn$ for sufficiently large n.

$(n!) = n * (n-1) * (n-2) \ldots\ 2 * 1$

$log(n!) = log(n * (n-1) * (n-2) \ldots\ 2 * 1)$

$log(n!) = log(n) + log(n-1) + log(n-2) + \ldots + log2 + log1$

or rearranging: $log(n!) = log1 + log2 + \ldots + log(n-2) + log(n-1) + log(n)$

This can be expressed as a summation:

$$log(n!) = \sum_{i=1}^{n} log(i)$$

1. Proof of the Upper Bound: $log(n!) \in O(nlogn)$

We must prove $log(n!) \leq c * nlogn$ for some constant $c > 0$ and for all $n \geq n_0$

Looking at the summation above, each term $log(i)$ is $\leq$ the largest term which is $log(n)$

ie. $log(i) \leq log(n)$ for all i from 1 to n. Hence, we can establish an upper bound by replacing every term in the sum with $log(n)$:

$$\sum_{i=1}^{n} log(i) \leq \sum_{i=1}^{n} log(n)$$

Note: The summation below is the same as adding $log(n)$ to itself n times and hence $= nlogn$

$$\sum_{i=1}^{n} log(n) = n \cdot logn$$

Therefore we have: $log(n!) \leq nlogn$. This fits the definition of Big-O with $c = 1$ and $n_0 = 1$ which therefore proves $log(n!) \in O(nlogn)$.

2. Proof of the Lower Bound: $log(n!) \in \Omega(nlogn)$

We must prove $log(n!) \geq c * nlogn$ for some constant $c > 0$ and for all $n \geq n_0$.

$$log(n!) = \sum_{i=1}^{n} log(i) = log(1) + ... + log(\frac{n}{2}) + ... + log(n)$$

The first half of the terms (from $i = 1$ to $i = n/2$) is smaller than the second half. To find the lower bound, we can ignore the first half of the terms.

$$\sum_{i=1}^{n} log(i) \geq \sum_{i=n/2}^{n} log(i)$$

The smallest value on the right will be $log(\frac{n}{2})$.

$log(i) \geq log(\frac{n}{2})$ for all i from $\frac{n}{2}$ to n.

Moreover, the the number of terms in the sum from $i = \frac{n}{2}$ to n is $(n - \frac{n}{2}) + 1 = \frac{n}{2} + 1$ which is greater than $\frac{n}{2}$.

We can use the above now such that the smallest value is $log(\frac{n}{2})$ and there are at least $\frac{n}{2}$ terms:

$$v \geq \sum_{i=n/2}^{n} log(n) \geq n/2 * log(\frac{n}{2}) = \frac{n}{2}(log(n) - log(2)) = \frac{n}{2} \cdot log(n) - \frac{n}{2} \cdot log(2)$$

$log(n!) \geq c * nlogn$ for some constant $c > 0$ and for all $n \geq n_0$.

If we set $n \geq 4$ and take the log of both sides: $log(n) \geq log(4) = 2log(2)$

We divide both sides by 2: $\frac{1}{2}log(n) \geq log(2)$

We can now substitute back into $(\frac{n}{2}) \cdot log(n) - (\frac{n}{2}) \cdot log(2) \geq (\frac{n}{2})log(n) - (\frac{n}{2})((\frac{1}{2})log(n)) = (\frac{n}{4})log(n) = (\frac{1}{4})n \cdot log(n)$

Therefore we have shown that $log(n!) \geq (\frac{1}{4})n \cdot log(n)$ with $c = \frac{1}{4}$ and $n_0 = 4$.

. .

2. Derive asymptotic upper bounds for each recurrence below, using a method of your choice.

- $T(n) = 2T(n/6) + 1$

$C_(root) = 1$

$C_(level1) = 2 \cdot 1 = 2$ [2 subproblems at cost of 1 each]

Therefore, this is leaf dominated and cost is upper bounded by the number of leaves at unit cost 1.

#leaves = c

Therefore, $T(n) = O(n^{(log_6 2)})$

. .

- $T(n) = 6T(n/4) + n$ $C_(root) = n$

$C_(level1) = 6 \cdot f(\frac{n}{4}) = 6 \cdot n/4 = \frac{3}{2}n$

Therefore this is leaf dominated and cost is upper bounded by the number of leaves.

#leaves $= a^{log_b(n))} = n^{(log_b(a))} = n^{(log_4 6)}$

Therefore, $T(n) = O(n^{log_4 6})$

. .

- $T(n) = 7T(n/7) + n$

$C_(root) = n$

$C_(level1) = 7(\frac{n}{7}) = n$

Cost is same at each level, therefore this is balanced.

Depth $= log_b n = log_7 n$. Average cost per level is n. Total cost = depth * average cost per level $= log_7 n$ * n. Therefore, $T(n) = O$(nlog_7n)$

. .

- $T(n) = 9T(n/4) + n^2$

$C_(root) = n^2$

$C_(level1) = 9(\frac{n}{4})^2 = \frac{9}{16}n^2$ [cost decreased by factor of 9/16]

Therefore, this is root dominated. $T(n) = O(n^2)$

. .

- $T(n) = 4T(n/2) + n^3$

$C(root) = n^3$

$C(level1) = 4 \cdot (\frac{n}{2}^3) = \frac{4}{8}n^3 = \frac{1}{2}n^3$ [cost decreased by a factor of 1/2]

Therefore, this is root dominated. $T(n) = O(n^3)$

. .

- $T(n) = 49T(n/25) + n^{3/2} \log n$

$C_(root) = n^{3/2} \log n$ $C_(level1) = 49 \cdot (\frac{n}{25}^{3/2} \cdot log(\frac{n}{25})) = \frac{49}{125} \cdot n^{3/2} \cdot (logn - log25))$
[This is less geometrically less than the root]

Therefore this is root-dominated. **$T(n) = O(n^{3/2} \log n)$

. .

- $T(n) = T(n-1) + 2$

This is not a "divde and conquer" recurrence, so the tree is a single long branch.

$C_(root) = 2$

$C_(level1) = 2$

This is a balanced recurrence.

The problem size decreases by 1 each step. $n, n-1, n-2, ..., 1$. Therefore the total number of calls (bricks) is n.

The cost per level is 2. Total cost $= 2n + C$ where C is the constant cost of each base case.

Therefore, $T(n) = O(n)$

. .

- $T(n) = T(n-1) + n^c$, with $c \geq 1$

$C_(root) = n^c$

$C_(level1) = (n-1)^c$

The work at each level is quite consistent, on the order of $O(n^c)$.

The number of levels or depth: The problem goes from n to $n-1 \ldots$ 1 so there are n levels.

So the total Cost, C, $= O(n^c \cdot n^1) = O(n^{c+1})$

. .

- $T(n) = T(\sqrt{n}) + 1$

$C_(root) = 1

$C_(level1) = 1

Therefore this recurrence is balanced with a work of 1 at each level.

The problem size goes from n to $n^{\frac{1}{2}}$ to $n^{\frac{1}{4}} \ldots$ So that at any general level k, the problem size is $n^{\frac{1}{2^k}}$. The recursion stops when the problem size is reduced to a base case, for example 2. We need to find the depth where this occurs.

$n^{\frac{1}{2^k}} = 2$. We can then take log_2 of both sides.

$log_2 n^{\frac{1}{2^k}} = log_2 2$

$\frac{1}{2^k} log_2(n) = 1$

$2^k = log_2(n)$. We need to solve for k so take log_2 of both sides.

$k = log_2 log_2(n)$

Now total work is the sum of the work at all levels. Cost at each level is 1 and number of levels = k.

Therefore $T(n) = T(\sqrt{n}) + 1 = O(1 \cdot log_2 log_2(n)) = O(log_2 log_2(n))$

. .

3. Suppose that for a given task you are choosing between the following three algorithms:

   - Algorithm $\mathcal{A}$ solves problems by dividing them into two subproblems of one fifth of the input size, recursively solving each subproblem, and then combining the solutions in quadratic time.
   - Algorithm $\mathcal{B}$ solves problems of size n by recursively one subproblems of size $n-1$ and then combining the solutions in logarithmic time.
   - Algorithm $\mathcal{C}$ solves problems of size n by dividing them into a subproblems of size $n/3$ and a subproblem of size $2n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^{1.1})$ time.

   What is the work and span of these algorithms? For the span, just assume that it is the same as the work to combine solutions (i.e. the non-recursive quantity). Which algorithm would you choose? Why?

Algorithm $\mathcal{A}$:

$W_A(n) = 2W_A(n/5) + n^2$

$C_(root) = n^2$

$C_(level1) = 2(n/5)^2 = \frac{2}{25} \cdot n^2$

Therefore this is root dominated. $W_A(n) = O(n^2)$

Span: The recursive calls can be run in parallel, so we can account for the longest one plus the time to combine:

$S_A(n) = S_A(n/5) + O(n^2)$

$C_(root) = n^2$

$C_(level1) = (n/5)^2 = \frac{1}{25} \cdot n^2$

Therefore this is root dominated. $S_A(n) = O(n^2)$

. .

Algorithm $\mathcal{B}$:

$W_B(n) = W_B(n-1) + logn$

$C_(root) = logn$

$C_(level1) = log(n-1)$

The work is not geometrically changing and this is therfore a balanced recurrence.

There are n levels (n to $n-1 \ldots 1$) with a cost at each level on the order of $logn$.

Therefore total work is: $W_B(n) = O(nlogn)$

Span: For the recursive calls, there is only one subproblem ($S_B(n-1)$) so nothing to run in parallel. So the work and span should be the same.

$S_B(n) = S_B(n-1) + logn$

$S_B(n) = O(nlogn)$

. .

Algorithm $\mathcal{C}$:

$W_C(n) = W_C(n/3) + W_C(2n/3) + n^{1.1}$

$C_(root) = n^{1.1}$

$C_(level1) = (n/3)^{1.1} + (2n/3)^{1.1}$

It appears that the work is geometrically decreasing (level one would be 0.897 the size of the root for example).

Therefore the work is asymptotically equivalent to the work of the root: $W_C(n) = O(n^{1.1})$.

Span: The two subproblems will run in parallel but the longest path will be determined by the $(2n/3)$ subproblem.

Hence: $S_C(n) = S_c(2n/3) + n^{1.1}$. This is also root dominated so $S_c(n) = O(n^{1.1})$

Summary of Algorithms: | Algorithm | Work Complexity | Span Complexity | | :— | :— | :— | | $\mathcal{A}$ | $\Theta(n^2)$ | $\Theta(n^2)$ | | $\mathcal{B}$ | $\Theta(n \log n)$ | $\Theta(n \log n)$ | | $\mathcal{C}$ | $\Theta(n^{1.1})$ | $\Theta(n^{1.1})$ |

I would choose alogrithm $\mathcal{B}$ as its work and span are both log-linear functions and will be asymptotically dominated by the polynomial work and span functions of algorithms $\mathcal{A}$ and $\mathcal{C}$. Work is more important than span overall, but from both work and span perspectives, algorithm $\mathcal{B}$ is superior.

. .

4. Suppose that for a given task you are choosing between the following three algorithms:

   - Algorithm $\mathcal{A}$ solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
   - Algorithm $\mathcal{B}$ solves problems of size n by recursively solving two subproblems of size $n-1$ and then combining the solutions in constant time.

- Algorithm $\mathcal{C}$ solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What is the work and span of these algorithms? For the span, just assume that it is the same as the work to combine solutions (i.e., the non-recursive quantity). Which algorithm would you choose? Why?

Algorithm $\mathcal{A}$:

$W_A(n) = 5W_A(n/2) + n$

$C_(root) = n$

$C_(level1) = 5 \cdot (n/2) = 5/2n$

Therefore, this is a leaf-dominated recurrence. Number of leaves is $n^{log_2 5}$.

Therefore, $W_A(n) = O(n^{log_2 5})$

Span: Take length of longest subproblem = n/5. $S_A(n) = S_A(n/2) + n

$C_(root) = n$

$C_(level1) = 1 \cdot (n/2) = n/2$

Therefore, this is root dominated and $S_A(n) = O(n)$

. .

Algorithm $\mathcal{B}$:

$W_B(n) = 2W_B(n-1) + 1

$Cost_(root) = 1$

$Cost_(level1) = 2c$

Therefore, this recurrence is leaf dominated. There are n levels. The number of leaves per level is 2^i. So we need to sum the work from all levels (from k=0 to the last level of k=n-1).

Therefore, $\sum_{k=0}^{n-1} 2^k$. We can use or geometric series bounds. $\alpha = 2$, $\alpha > 1$.

$\sum_{k=0}^{n} \alpha^k = \frac{\alpha}{\alpha-1} \cdot \alpha^n$. Therefore, $= 2 \cdot 2^{n-1} = 2^n$.

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

Span: S_B(n) = S_B(n-1) + 1

$Cost_(root) = 1$

$Cost_(level1) = 1$

There are n levels with constant cost per level.

Therefore, $^*S_B(n) = O(n)$

. .

Algorithm $\mathcal{C}$:

$W_C(n) = 9W_C(n/3) + c(n^2)$

$Cost_(root) = c(n^2)$

$Cost_(level1) = 9 \cdot c(n/3)^2 = c \cdot \frac{9}{9}n^2 = c(n^2)$

Therefore, this is balanced. Work per level is $c(n^2)$. Number of levels is $(log_3 n)$.

$W_C(n) = O(n^2 \cdot logn)$

Span: $S_C(n) = S_C(n/3) + c(n^2)$

$Cost_(root) = c(n^2)$

$Cost_(level1) = c(n/3)^2 = c \cdot \frac{1}{9}n^2$. The cost is geometrically decreasing, therefore this recurrence is root dominated.

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

Here is a summary table:

Algorithm	Work Complexity	Span Complexity
$\mathcal{A}$	$\Theta(n^{\log_2 5})$	$\Theta(n)$
$\mathcal{B}$	$\Theta(2^n)$	$\Theta(n)$
$\mathcal{C}$	$\Theta(n^2 \log n)$	$\Theta(n^2)$

Work is generally more important than span. We can rule out algorithm B as it does exponential work. Now we have algorithms with polynomial complexity. The work between A and C is similar although for very large n, the work of C is lower (A has $\sim n^{2.32}$ versus C which has $n^2 \cdot logn$). Algorithm A has much better span (more parallelism) so assuming infinite processors A may be the better choice. If work is the primary consideration, and we do not have infinite processors, C would be the better choice.

. .

5. In Module 2 we discussed two algorithms for integer multiplication. The first algorithm was simply a recapitulation of the "grade school" algorithm for integer multiplication, while the second was the Karatsaba-Ofman algorithm. For this problem, you will use the stub functions in `main.py` to implement these two algorithms for integer multiplication. Once you've correctly implemented them, test the empirical running times across a variety of inputs to test whether your code scales in the manner predicted by our analyses of the asymptotic work.

Please see main.py for code implementation. Here is the output: `PS C:\Users\adumo\GitHub\problem-set-02-adumont2> python main.py`

```
All tests passed!
Timing for n = 10...
Timing for n = 100...
Timing for n = 1000...
Timing for n = 10000...
Timing for n = 100000...
Timing for n = 1000000...
Timing for n = 10000000...
Timing for n = 100000000...
Timing for n = 1000000000...
```

```
Comparison of Multiplication Algorithm Runtimes (in ms)
|          n |   Quadratic O(n^2) |   Subquadratic O(n^1.585) |
|------------|--------------------|---------------------------|
|         10 |              0.037 |                     0.007 |
|        100 |              0.037 |                     0.013 |
|       1000 |              0.092 |                     0.035 |
|      10000 |              0.093 |                     0.049 |
|     100000 |              0.148 |                     0.056 |
|    1000000 |            inf     |                     0.087 |
|   10000000 |            inf     |                     0.087 |
|  100000000 |            inf     |                     0.129 |
| 1000000000 |            inf     |                     0.156 |
```

One can see that both algorithms work. However, the Karatsuba-Ofman (subquadtratic_multiply) algorithm is vastly more efficient and scalable for large numbers.

. .

6. A "white hat" conducts hacking activities for the common good, while a "black hat" hacker does so for nefarious reasons. Let's consider a computer security class with n students who are all either white hat or black hat hackers. You're the instructor, and you don't know who is a white hat or a black hat, but all of the student do.

Your goal is to identify the white hats and you're allowed to ask a pair of students about one another. White hats will always tell the truth, but you cannot trust a black hat's response. For a pair of students A and B then there are several possible outcomes:

A says	B says	Conclusion
B is a white hat	A is a white hat	both are good, or both are bad
B is a white hat	A is a black hat	at least one is bad
B is a black hat	A is a white hat	at least one is bad

A says	B says	Conclusion
B is a black hat	A is a black hat	at least one is bad

6a. Show that if more than $n/2$ students are black hats, you cannot determine which students are white hats based on a pairwise test. Note that you must assume the black hats are conspiring to fool you.

Goal: Show that if more than $n/2$ students are black hats, the instructor cannot reliably identify the white hats.

Let w be the number of white hats and b be the number of black hats, where $w + b = n$. The condition stipulates that $b > n/2$, which also implies that $b > w$.

To prove it is impossible to identify the white hats, we can demonstrate that there are at least two possible configurations of students that can produce the exact same interview results, making them indistinguishable.

Scenario 1: The Real World. There is a group W contining w white hats. There is a group B containing b black hats.

Scenario 2: The Conspiracy. All n students are black hats. Since $b > w$, they can form a subgroup of w black hats (we will call $B_(conspire)$) to perfectly mimic the behavior of the real white hats. The other $b - w$ black hats are in the other group (we will call $B_(other)$).

Here is how the black hats in Scenario 2 can conspire to produce answers that are identical to those from Scenario 1: 1. If you interview 2 students from $B_(conspire)$: They will lie and say each other is a "white hat". This is indistinguishable from interviewing two real white hats from group W.

2. If you interview a student from $B_(conspire)$ and a student from $B_(other)$: The student from $B_(conspire)$ will say the other is a "black hat" (which is true in this case). This is indistinguishable from a real white hat indentifying a real black hat.

3. If you interview two students from $B_(other)$: They can say whatever they want, just as real black hats could in Scenario 1.

The group of conspirators ($B_(conspire)$) can perfectly mimic the behavior and interview responses of a true group of white hats (W), so there is not set of questions you can ask to tell Scenario 1 and 2 apart. Since you cannot distinguish between w white hats and w black hats that are just pretending, **you therefore cannot guarantee the identification of white hats when black hats are in the majority.**

. .

6b. Consider the problem of finding a single white hat, assuming strictly more than $n/2$ of the students are white hats. Show that $n/2$ pairwise interviews is enough to reduce the problem size by a constant fraction.

We are given this table which is helpful:

A says	B says	Conclusion
B is a white hat	A is a white hat	both are good, or both are bad
B is a white hat	A is a black hat	at least one is bad
B is a black hat	A is a white hat	at least one is bad
B is a black hat	A is a black hat	at least one is bad

Approach: We can use a "pairwise elimination" strategy as follows:

1. We can pair the students arbitrarily into $(n/2)$ pairs and leave one unpaired if n is an odd number of students.
2. For each pair (A, B), ask A about B and B about A with the results as shown in the table above.
3. If both say the other is a white hat, keep one (will say A) as a candidate. In this scenario both are good or both or bad as per the table.
4. In all other cases, discard both A and B. This ensures at least one bad student is discarded as shown in the table.
5. Keep the leftover student (if any) as a candidate.

Justification: This process uses $(n/2)$ pairwise interviews and reduces the number of candidates to at most $(n/2)$. We have successfully reduced the problem size by a constant fraction. Also, the new group of candidates is guaranteed to contain at least one white hat. If the new group had zero white hats, it would imply that every original white hat was paired with a black hat - which is impossible given the constraints of the problem where white hats are the majority (strictly $> (n/2)$ or $w > b$). Therefore, we are left with a smaller, valid subproblem with each iteration.

6c. Using the above show that all white hats can be identified using $\Theta(n)$ pairwise interviews.

The overall strategy is to first find one guaranteed white hat in linear time, and then use that person to identify everyone else also in linear time (white hats will always tell the truth).

Part 1: Identify the one guaranteed white hat. We will recursively use the reduce algorithm outlined in 6b. We will start with the initial group of n students, S_0. We will then create a new group, S_1 with the algorithm with size $\sim (n/2)$. We will create a new group, S_2, by applying the algorithm on S_1 with size of $S_2 \sim (n/4)$. We continue this process until only one candidate is left, C. The final candidate will be a white hat (proof above in 6b).

Interviews = $[n/2] + [n/4] + [n/8] + \ldots + 1$. This is a geometric series whose sum will always be less than n. So finding our candidate costs $O(n)$ interviews.

We can also look at this from the perspective of a recurrence and use the brick method. At each step, we perform $(n/2)$ interviews and then make one recursive call on a problem of a most size $(n/2)$. We can write this recurrence as:

$I(n) = I(n/2) + c \cdot n$

This is root dominated where work is decreasing geometrically by a factor of 2, so $O(n)$.

Part 1b: Verify the candidate. While we proved C must be a white hat above, we can formally verify that C is a white hat to be certain. We can take the candidate C and ask every other student in the original group of n about them. We can count how many students claim "C is a white hat". If the count is greater than $n/2$, C is certified to be a white hat. If C is a white hat, all w white hats (where $w > n/2$) will truthfully say C is a white hat, guaranteeing the count is $> n/2$. If C were a black hat, the w white hats would truthfully say C is a black hat. Only the b black hats could possibly lie but they are a minority. This verification step costs $n - 1$ interviews. Combining both steps to find one certified white hat is $O(n) + O(n)$ which is still $O(n)$.

Part 2 - Find all other white hats. Now that we have one certified white hat, $W_(cert)$, we can use this white hat as an oracle. For each of the other $n - 1$ students we can ask the one verified white hat to identify them and they will always be truthful. This stage costs exactly $n - 1$ interviews which is $O(n)$.

Total complexity:

Upper bound: The total number of interviews is the sum of the costs from Part 1 and 2 above which is $O(n) + O(n)$ which is still $O(n)$.

Lower bound: To be certain of every student's identity, each student must be included in at least one interview (either as the interviewer or interviewee). Since each interview involves 2 students, one must perform at least $n/2$ interviews to gather information on everyone. Therefore, this problem must have a lower bound of $\Omega(n)$.

Since the algorithm runs in $O(n)$ time, and the problem is lower bounded by $\Omega(n)$, the number of pairwise interviews required is $\theta(n)$.