CS 624: Analysis of Algorithms

Fall 2024 Assignment 1

Solutions

1. This question is based on Appendix C in CLRS, 4^th edition, question C.1-11 (page 1183). Argue that for any integers $n \geq 0, j \geq 0, k \geq 0$ and $j + k \leq n$:

   $\binom{n}{j+k} \leq \binom{n}{j} * \binom{n-j}{k}$

   Provide both an algebraic proof and an argument based on a method for choosing j + k items out of n. Give an example in which equality does not hold.

   Solution:

   (a) Algebraic solution:

   $$\binom{n}{j+k} = \frac{n!}{(j+k)!(n-j-k)!}$$
   $$\binom{n}{j} * \binom{n-j}{k} = \frac{n!}{j!(n-j)!} * \frac{(n-j)!}{k!(n-j-k)!} = \frac{n!}{j!k!(n-j-k)!}$$

   The difference between the two sides is that one the left we have $(j+k)!$ in the denominator and on the right we have $j!k!$:

   $$(j+k)! = 1 * 2 * 3.... * j * (j+1) * (j+2)... * (j+k)$$
   $$j!k! = 1 * 2 * 3.... * j * 1 * 2 * 3... * k$$
   $$\frac{(j+k)!}{j!k!} = \frac{(j+1) * (j+2) * ... * (j+k)}{1 * 2 * ... * k}$$

   The quotient is ≥ 1 (equality is when either j or k are zero). Therefore, $\binom{n}{j+k} \leq \binom{n}{j} * \binom{n-j}{k}$ because the difference between the two side is in the denominator, and on the left the denominator is equal to or larger than the right.

   (b) Both sides describe the process of selecting j+k items out of n, but on the left we pick all j+k items at the same time, and on the right we pick j items and then pick k items out of the remaining n-j. The right hand side, then, involves not only selecting j+k items but also partitioning them into a set of j and a set of k – and there are many more ways to do it (unless j or k are zero). For example – let $j = 4$ and $k = 1$. There are $\binom{n}{5}$ ways to select five items out of n and $\binom{n}{4} * \binom{n-4}{1} = \binom{n}{4} * (n-4)$ ways to select four and then one. Now, $\binom{n}{4} * (n-4) = \frac{n!}{4!(n-5)!} = 5 * \binom{n}{5}$. In other words, we have five times more options (which makes sense, considering that we have five ways to partition five elements into 4 and 1).

2. Decide whether each of the following statements is true or false, and prove that your conclusion is correct.

(a) $2^{n+1} = O(2^n)$.

Solution: True, because $2^{n+1} = 2 * 2^n$. The two functions are equal up to a constant factor.

(b) $f(n) = O\big(g(n)\big)$ implies $2^{f(n)} = O\big(2^{g(n)}\big)$.

Solution: False. We can use the previous case as a counter-example. $2^{n+1} = O(2^n)$, but $2^{2^{n+1}} = (2^{2^n})^2$. In other words, $2^{2^{n+1}}$ grows quadratically faster than (2^{2^n}) so it can never be bounded by it up to any constant factor.

3. Prove the correctness of the following algorithm for evaluating a polynomial

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

at a number x:

Algorithm 1 Horner(a,x)

```
p = a_n
for i = n − 1 to 0 do
    p := p · x + a_i
end for
return p
```

This algorithm, as you probably know, is called *Horner's method.* You can use induction on the loop invariant using initiation, maintenance and termination.

Solution: The loop invariant depends on the value of i: In the beginning of the j^{th} iteration where $i = n - j$, the value of the polynomial is $p = \sum_{k=1}^{j} a_{n+k-j} x^{k-1}$.

- **Initiation:** Before the start of the first loop, $i = n - 1$ and $j = 1$. The value of the polynomial is $p = \sum_{k=1}^{1} a_n x^0 = a_n$.
- **Maintenance:** Let us assume it is true when we start loop $j \geq 1$, where the value of i is n-j. By inductive hypothesis, in the beginning of the loop, $p = \sum_{k=1}^{j} a_{n+k-j} x^{k-1}$. The loop multiplies the entire sum by x and adds $a_i = a_{n-j}$, rendering it $p = \sum_{k=1}^{j} a_{n+k-j} x^k + a_{n-j} = \sum_{k=1}^{j+1} a_{n+k-(j+1)} x^{k-1}$ after the loop ends, so the condition holds in the beginning of the next loop if there is a next loop.
- **Termination:** After the end of the last loop, when i=0 and j=n, the polynomial is $p = \sum_{k=1}^{n+1} a_{k-1} x^{k-1} = \sum_{k=0}^{n} a_k x^k$ as needed.

4. Prove that if $f = O(g)$ and $g = O(h)$ then $f = O(h)$.

Solution: We go to the basics. If $f(n) = O(g(n))$ then there are constants c, n_0 such that $f(n) \leq c * g(n)$ for all $n \geq n_0$. Similarly, if $g(n) = O(h)$ then there are constants d, n_1 (don't use the same constants are before!) such that $g(n) \leq d * h(n)$ for all $n \geq n_1$. Substituting the second equation into the first gives us $f(n) \leq c * d * h(n)$ for all $n \geq \max(n_0, n_1)$. So, by definition, $f(n) = O(h(n))$.

5. Give asymptotic tight bounds for T(n) for each of the recurrences. Justify your answers.

(a) $T(n) = 2T(n/2) + n^3$
This is case 3 of the Master theorem. a=2 and b=2, so $n^{\log_b a} = n$ and $f(n) = n^3$. Obviously, $n = O(n^3 - \epsilon)$. We also have to find constants $0 < c < 1$ and n_0 such that $2(n/2)^3 = \frac{1}{4}n^3 \leq c * n^3$. Obviously, any $c \geq \frac{1}{4}$ satisfies this for all positive n, therefore $T(n) = O(n^3)$.

(b) $T(n) = T(8n/11) + n$
a=1, b=$\frac{11}{8}$, so $n^{\log_{\frac{11}{8}} 1} = 1$. sp this is case 3 of the master theorem if we can find constants $0 < c < 1$ and n_0 such that $8n/11 \leq cn$. Any $c \geq \frac{8}{11}$ will do, so

(c) $T(n) = 16T(n/4) + n^2$
Here a=16, b=4 and $n^{\log_4 16} = n^2$. This is case 2 of the Master theorem and $T(n) = O(n^2 \log n)$.

(d) $T(n) = 7T(n/2) + n^2 \log n$
Here a=7, b=2 and $n^{\log_2 7} \approx n^{2.81}$ and $f(n) = n^2 \log n$. This is case 1 of the Master theorem and $T(n) \approx O(n^{2.81})$. Notice that the log is not a problem here, since $n^2 \log n = O(n^{2.81-\epsilon})$ for any $\epsilon < 0.81$.

(e) $T(n) = 2T(n/4) + \sqrt{n}$
Here a=2, b=4 and $n^{\log_2 2} = \sqrt{n}$. This is case 2 of the Master theorem and $T(n) = O(\sqrt{n} \log n)$.

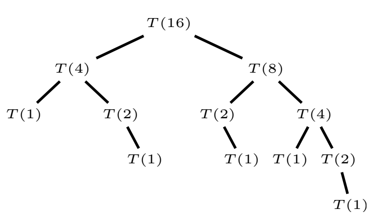
6. Problem 4.2 in Lecture notes 1 (page 7).

$$T(n) = \sum_{j=2}^{n}(a + (j-1)c) = \sum_{j=2}^{n}(a) + c\sum_{j=2}^{n}(j-1) = a(n-1) + c\sum_{j=1}^{n-1} j = a(n-1) + c\frac{n(n-1)}{2}.$$

Rearranging the terms we get $an - a + n^2\frac{c}{2} - n\frac{c}{2}$. So, if we set $A = \frac{c}{2}$, $B = a - \frac{c}{2}$ and $C = -a$ we get a term of the form $T(n) = An^2 + Bn + C$. Since $c > 0$, then $A > 0$ as well, so it's a quadratic term.

7. Problem 4.1 in Lecture notes 2 (page 13).

See drawing. I can be quite flexible with any handling of terminal nodes where $T(n/4) < 1$.



8. The Split3-Sort algorithm is defined as follows:

Algorithm 2 Split3-Sort(A,p,r)

if $(A[p] > A[r])$ **then**
  Swap A[p] with A[r]
end if
if $(p + 1 < r)$ **then**
  $k = \lfloor (r - p + 1)/3 \rfloor$ // Round down
  $Split3 - sort(A, p, r - k)$ // First two thirds
  $Split3 - sort(A, p + k, r)$ // Last two thirds
  $Split3 - sort(A, p, r - k)$ // First two thirds again
end if

(a) Prove that the call to Split3-Sort(A, 1, n) correctly sorts the array $A[1..n]$ (**Hint:** I found the best way is to use induction, but be careful with the base case - notice line 4. What is the minimum difference between p and r?)

Solution:

- The boundary condition in line 4 happens is if $p + 1 \geq r$, so if the array is size 2 or less. Therefore, the base case is an array of size 2 (empty arrays or arrays of size 1 are trivially sorted). In this case the if condition in lines 1–2 takes care of the sorting by swapping the two entries if they are out of order.
- The inductive hypothesis: for an array size k such that $2 \leq k < n$, $Split3 - sort(A, p, r)$ sorts the array.
- Now the tricky part. After the first call in line 6, the first two-thirds are sorted within one another by inductive hypothesis (let's call them third-1 and third-2, resp.). The second call of line 7 uses third-2 and third-3. Since third-1 and third-2 sorted after line 6, we are sure that third-1 include the smallest $\frac{1}{3}$ of these two thirds. We don't know how they are with respect to third-3, but one thing we know for sure: They can't possibly be the largest $\frac{1}{3}$ of the entire array (why?). Therefore, the largest $\frac{1}{3}$ of the entire array are somewhere within third-2 and third-3. After the second call in line 7, which sorts third-2 and third-3 by inductive hypothesis, we are now certain that the largest $\frac{1}{3}$ of the entire array are indeed now in third-3, in sorted order. Therefore, third-1 and third-2 contain the remaining smallest $\frac{2}{3}$. The call in line 8 sorts them as well, by inductive hypothesis.

(b) Write the recurrence formula for Split3-Sort and give the asymptotic bound on the run time (Θ notation). **Solution:** $T(n) = 3T(\frac{2}{3}n) + c$ – we make three recursive calls to $\frac{2}{3}$ of the input and the rest – boundary conditions + if statement and swap, is a constant. Using the Master theorem, $a = 3$, $b = \frac{3}{2}$ and $f(n) = c$ (constant). $n^{\log_b a} \approx 2.71$. This is case 3 of the Master theorem (why?), so $T(n) = O(n^{2.71})$.

(c) Compare the run time from (b) to the run time of HeapSort, MergeSort and QuickSort. Is it better? Worse? Same?

Solution: Obviously, based on the discussion in class, it is worse.

9. Let $\{f_n : n = 0, 1, \dots\}$ be the Fibonacci sequence (where by convention $f_0 = 0$ and $f_1 = 1$).

(a) This question is based on material from lecture notes 2. Show that $\sum\limits_{n=1}^{\infty} \frac{nf_n}{2^{n-1}} = 20$. Do this by using a generating function as shown in the last section of the Lecture 2 notes, and differentiating. **Hint:** The derivative of $\frac{x}{1-x-x^2}$ is $\frac{1+x^2}{(1-x-x^2)^2}$.

Solution: We showed that the sum of the Fibonacci generating function, $F(X) = \sum\limits_{n=0}^{\infty} f_n x^n = \frac{x}{1-x-x^2}$. Differentiating both sides using the hint above gives us:

$\frac{1+x^2}{(1-x-x^2)^2} = \sum\limits_{n=1}^{\infty} nf_n x^{n-1}$. Substituting $x = \frac{1}{2}$ gives us $\sum\limits_{n=1}^{\infty} \frac{nf_n}{2^{n-1}} = 20$ as needed.

(b) Show why (in the same way as you proved the first part of this problem) you might think that $\sum\limits_{n=1}^{\infty} nf_n = 2$. Then show why this could not possibly be true (it doesn't have to be a long answer, but it has to be convincing).

Solution: We could follow the previous steps and use $x = 1$ instead of $\frac{1}{2}$. This would give us $\sum\limits_{n=1}^{\infty} nf_n = 2$. However, the entire derivation relies on the fact that a series of the form $\sum_{n=0}^{\infty} x^n$ converges into a constant. This is only true if $x < 1$. When $x = 1$ this series goes to infinity, just like $\sum_{n=1}^{\infty} n f_n$.