University of Houston

Homework 1

COSC 3320
Algorithms and Data Structures

Gopal Pandurangan

Solutions

This page intentionally left blank.

1 Class Questions

1.1 January 26

Question 1

Show that any degree d polynomial $p(n) = a_0 n^d + a_1 n^{d-1} + \cdots + a_{n-1} n + a_n$, with $a_0 > 0$, is $\mathcal{O}(n^d)$.

Solution.

$$\begin{aligned} \lim_{n\to\infty} \frac{a_0 n^d + a_1 n^{d-1} + \cdots + a_{n-1} n + a_n}{n^d} &= \lim_{n\to\infty} a_0 + \frac{a_1}{n} + \cdots + \frac{a_{n-1}}{n^{d-1}} + \frac{a_n}{n^d} \\ &= a_0 \end{aligned}$$

□

Question 2

Show that, for any $a > 1$, $n^b = \mathcal{O}(a^n)$.

Solution. First, this is clearly true if $b < 0$. Otherwise, we can apply the limit definition. Say b is an integer. Then

$$\begin{aligned} \lim_{n\to\infty} \frac{n^b}{a^n} &= \lim_{n\to\infty} \frac{bn^{b-1}}{a^n \ln a} \\ &= \lim_{n\to\infty} \frac{b(b-1)n^{b-2}}{a^n \ln^2 a} \\ &\vdots \\ &= \lim_{n\to\infty} \frac{b!}{a^n \ln^b a} \\ &= 0 \end{aligned}$$

If, on the other hand, b is not an integer, then $b = \lfloor b \rfloor + r$, and

$$\begin{aligned} \lim_{n\to\infty} \frac{n^b}{a^n} &= \lim_{n\to\infty} \frac{bn^{b-1}}{a^n \ln a} \\ &= \lim_{n\to\infty} \frac{b(b-1)n^{b-2}}{a^n \ln^2 a} \\ &\vdots \\ &= \lim_{n\to\infty} \frac{b(b-1)\ldots(1+r)(r)b^{r-1}}{a^n \ln^b a} \\ &= 0 \end{aligned}$$

Alternatively, we can approach this more elegantly by considering the series

$$S_n = \sum_{n=0}^{\infty} \frac{n^b}{a^n}$$

If this series converges, then

$$\lim_{n\to\infty} \frac{n^b}{a^n} = 0$$

Apply the ratio test on n^b/a^n

$$\begin{aligned}\lim_{n\to\infty} \frac{(n+1)^b/a^{n+1}}{n^b/a^n} &= \lim_{n\to\infty} \frac{a^n(n+1)^b}{a^{n+1}n^b} \\ &= \lim_{n\to\infty} \frac{1}{a}\left(1+\frac{1}{n}\right)^b \\ &= \frac{1}{a}\end{aligned}$$

The above limit is less than 1, since $a > 1$, hence the series converges by the ratio test. Thus,

$$\lim_{n\to\infty} \frac{n^b}{a^n} = 0$$

and $n^b = \mathcal{O}(a^n)$. □

Question 3

Show that $a^{\log_b n} = n^{\log_b a}$.

Solution. Let $x = \log_b a \log_b n$. Then

$$\begin{aligned} b^x &= b^{\log_b a \log_b n} \\ &= \left(b^{\log_b a}\right)^{\log_b n} \\ &= a^{\log_b n}\end{aligned}$$

Similarly,

$$\begin{aligned} b^x &= b^{\log_b a \log_b n} \\ &= b^{\log_b n \log_b a} \\ &= \left(b^{\log_b n}\right)^{\log_b a} \\ &= n^{\log_b a}\end{aligned}$$

□

1.2 January 28

Question 1

Prove by mathematical induction that the gcd algorithm given in class is correct. You may assume $\gcd(a, b) = \gcd(b, a \bmod b)$.

Solution. Carefully formulate the induction hypothesis: we wish to show that, for all n, the algorithm given in class (which we will refer to as the Euclidean Algorithm going forward) for determining $\gcd(a, b)$ for any $a \leq n$, $b \leq n$, is correct. Notice that this is equivalent to the statement "the Euclidean algorithm correctly determines $\gcd(a, b)$ for any non-negative integers a and b". From here, the proof is straightforward: the base case is trivial. Assume this is true for all $k < n$. Now, consider $\gcd(a, b)$ for $a \leq n$ and $b \leq n$. Without loss of generality, assume $a \geq b$ (since we can simply swap a and b if this is untrue).

If $a < n$, we are done by our induction hypothesis. Otherwise, $a = n$. Now, if $b = n$, we are again done (since then $\gcd(a, b) = n$). Otherwise, the Euclidean algorithm updates $\gcd(a, b) \leftarrow \gcd(b, a \bmod b)$. Since $b < n$, correctness follows from the induction hypothesis. □

1.3 February 4

Question 1

Solve $T(n) = 3T(n/2) + n^2$.

Solution. Write $a = 3$, $b = 2$, and $f(n) = n^2$. Then $af(n/b) = 3(n/2)^2 = 3n^2/4$, hence $c = 3/4 < 1$. Thus, $T(n) = \Theta(f(n)) = \Theta(n^2)$. $\square$

Question 2

Show the correctness of `MergeSort` using Induction. You may assume the `Merge` subroutine is correct.

Solution. The base case is trivial. Consider `MergeSort` on an array of length n. `MergeSort` is then recursively called on subarrays of length $n/2$, which are then sorted by our induction hypothesis. Then, by the correctness of the `Merge` procedure, the final array must be sorted. $\square$

Question 3

Argue that QuickSort cannot take more than $\binom{n}{2} = n(n-1)/2$ comparisons.

Solution. In the worst case, the pivot is chosen so that the partitions are of size $n-1$ and 1, hence there are $n-1$ comparisons. If this happens at each step, this totals

$$(n-1) + (n-2) + \cdots + 2 + 1 = \binom{n}{2}$$

comparisons. $\square$

2 Textbook Exercises

Exercise 4.1

Prove the asymptotic bound for the following recurrences by using induction. Assume that base cases of all the recurrences are constants i.e., $T(n) = \Theta(1)$, for $n < c$ where c is some constant.

(a) $T(n) \le 2T(n/2) + n^2$. Then, $T(n) = \mathcal{O}\big(n^2 \log n\big)$.

Solution. The base case is given. Suppose $T(k) \le ck^2 \log k$ for all $k < n$. Then

$$\begin{aligned}
T(n) &\le 2T(n/2) + n^2 \\
&\le 2c\left(\frac{n}{2}\right)^2 \log \frac{n}{2} + n^2 \text{ by the induction hypothesis} \\
&= \frac{c}{2}n^2(\log n - \log 2) + n^2 \\
&= \frac{c}{2}n^2(\log n - 1) + n^2 \\
&= \frac{c}{2}n^2 \log n - \frac{c}{2}n^2 + n^2 \\
&= \frac{c}{2}n^2 \log n + \left(1 - \frac{c}{2}\right)n^2 \\
&\le \frac{c}{2}n^2 \log n \text{ for } c \ge 2 \\
&\le cn^2 \log n
\end{aligned}$$

$\square$

Exercise 4.3(6)

Solve the following recurrences. Give the answer in terms of *Big-Theta* notation. Solve up to constant factors, i.e., your answer must give the correct function for $T(n)$, up to constant factors. You can assume constant base cases, i.e., $T(1) = T(0) = c$, where c is a positive constant. You can ignore floors and ceilings. You can use the DC Recurrence Theorem if it applies.

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

Solution. Write $a = 4$, $b = 2$, and $f(n) = n^3$. Then $af(n/b) = 4(n/2)^3 = n^2/2$, hence $c = 1/2 < 1$. Thus, $T(n) = \Theta(f(n)) = \Theta(n^3)$. □

Exercise 4.4

You are given an array consisting of n numbers. A popular element is an element that occurs (strictly) **more than** $n/2$ times in the array. Give an algorithm that finds the popular element in the array if it exists, otherwise it should output "NO". Your algorithm should take no more than $2n$ comparisons. (As usual, we only count the comparisons between array elements.) Give pseudocode, argue its correctness, and show that your algorithm indeed takes no more than $2n$ comparisons. (Hint: Use a decrease and conquer strategy, similar to the celebrity problem.)

Solution. Let A denote our array and define Popular-Candidate(A), so that, if a popular element of A exists, it returns it, as follows

Algorithm Determine if a popular element candidate exists

```
def Popular-Candidate(A):
    if |A| = 1:
        return A[0]
    else if |A| = 2 :
        if A[0] = A[1]:
            return A[0]
        else:
            return Null
    else:
        if two distinct elements exist in A:
            remove them
        else:
            return A[0]
```

The key intuition here is to observe that, if p is a popular element, then it is still popular when two *distinct* elements of A are removed. The proof is straightforward: if neither removed element is p, there are now $n - 2$ elements remaining, but $n/2 > (n - 2)/2$ occurrences of p. If exactly one element is p, then there are $(n - 1)/2 > (n - 2)/2$ occurrences of p, and it is still popular. Since the elements are distinct, they cannot *both* be p. Correctness follows directly from this: algorithm Popular-Candidate will return the popular element if it exists. If it does not exist, we must determine so, as in the following pseudocode eckboyermoore

Algorithm Return a popular element, if it exists, else return `Null`

```
def Popular-Element(A):
    n ← |A|
    candidate ← Popular-Candidate(A)
    count ← 0
    for each ele ∈ A:
        if ele = candidate:
            count ← count + 1
    if count > n/2:
        return candidate
    else:
        return Null
```

A note: Popular-Candidate should be implemented to take only n comparisons. This can be done in a variety of ways (the implementation details of which will vary from language to language), but the basic idea is to have one index i at the first element and a second index j at the second. Increment j until the elements differ, then remove the elements. □

Exercise 5.1

Determine the total number of comparisons that each of the following algorithms takes on $S = [8, 2, 6, 7, 5, 1, 4, 3]$.

- SimpleSort
- MergeSort
- QuickSort

Show the steps of the algorithm when calculating the number of comparisons.

Solution. SimpleSort

1. $[8, 2, 6, 7, 5, 1, 4, 3] \to [2, 8, 6, 7, 5, 1, 4, 3]$
2. $[2, 8, 6, 7, 5, 1, 4, 3]$
3. $[2, 8, 6, 7, 5, 1, 4, 3]$
4. $[2, 8, 6, 7, 5, 1, 4, 3]$
5. $[2, 8, 6, 7, 5, 1, 4, 3] \to [1, 8, 6, 7, 5, 2, 4, 3]$
6. $[1, 8, 6, 7, 5, 2, 4, 3]$
7. $[1, 8, 6, 7, 5, 2, 4, 3]$
8. $[1, 8, 6, 7, 5, 2, 4, 3] \to [1, 6, 8, 7, 5, 2, 4, 3]$
9. $[1, 6, 8, 7, 5, 2, 4, 3]$
10. $[1, 6, 8, 7, 5, 2, 4, 3] \to [1, 5, 8, 7, 6, 2, 4, 3]$
11. $[1, 5, 8, 7, 6, 2, 4, 3] \to [1, 2, 8, 7, 6, 5, 4, 3]$
12. $[1, 2, 8, 7, 6, 5, 4, 3]$
13. $[1, 2, 8, 7, 6, 5, 4, 3]$
14. $[1, 2, 8, 7, 6, 5, 4, 3] \to [1, 2, 7, 8, 6, 5, 4, 3]$
15. $[1, 2, 7, 8, 6, 5, 4, 3] \to [1, 2, 6, 8, 7, 5, 4, 3]$
16. $[1, 2, 6, 8, 7, 5, 4, 3] \to [1, 2, 5, 8, 7, 6, 4, 3]$
17. $[1, 2, 5, 8, 7, 6, 4, 3] \to [1, 2, 4, 8, 7, 6, 5, 3]$
18. $[1, 2, 4, 8, 7, 6, 5, 3] \to [1, 2, 3, 8, 7, 6, 5, 4]$
19. $[1, 2, 3, 8, 7, 6, 5, 4] \to [1, 2, 3, 7, 8, 6, 5, 4]$
20. $[1, 2, 3, 7, 8, 6, 5, 4] \to [1, 2, 3, 6, 8, 7, 5, 4]$
21. $[1, 2, 3, 6, 8, 7, 5, 4] \to [1, 2, 3, 5, 8, 7, 6, 4]$
22. $[1, 2, 3, 5, 8, 7, 6, 4] \to [1, 2, 3, 4, 8, 7, 6, 5]$
23. $[1, 2, 3, 4, 8, 7, 6, 5] \to [1, 2, 3, 4, 7, 8, 6, 5]$
24. $[1, 2, 3, 4, 7, 8, 6, 5] \to [1, 2, 3, 4, 6, 8, 7, 5]$
25. $[1, 2, 3, 4, 6, 8, 7, 5] \to [1, 2, 3, 4, 5, 8, 7, 6]$
26. $[1, 2, 3, 4, 5, 8, 7, 6] \to [1, 2, 3, 4, 5, 7, 8, 6]$
27. $[1, 2, 3, 4, 5, 7, 8, 6] \to [1, 2, 3, 4, 5, 6, 8, 7]$
28. $[1, 2, 3, 4, 5, 6, 8, 7] \to [1, 2, 3, 4, 5, 6, 7, 8]$

MergeSort

1. $[8][2] \to [2,8]$
2. $[6][7] \to [6,7]$
3. $[2,8][6,7] \to [2]$
4. $[2,8][6,7] \to [2,6]$
5. $[2,8][6,7] \to [2,6,7,8]$
6. $[5][1] \to [1,5]$
7. $[4][3] \to [3,4]$
8. $[1,5][3,4] \to [1]$
9. $[1,5][3,4] \to [1,3]$
10. $[1,5][3,4] \to [1,3,4,5]$
11. $[2,6,7,8][1,3,4,5] \to [1]$
12. $[2,6,7,8][1,3,4,5] \to [1,2]$
13. $[2,6,7,8][1,3,4,5] \to [1,2,3]$
14. $[2,6,7,8][1,3,4,5] \to [1,2,3,4]$
15. $[2,6,7,8][1,3,4,5] \to [1,2,3,4,5,6,7,8]$

QuickSort

1. $[8,2,6,7,5,1,4,3] \to [2],[8],[]$
2. $[8,2,6,7,5,1,4,3] \to [2,6],[8],[]$
3. $[8,2,6,7,5,1,4,3] \to [2,6,7],[8],[]$
4. $[8,2,6,7,5,1,4,3] \to [2,6,7,5],[8],[]$
5. $[8,2,6,7,5,1,4,3] \to [2,6,7,5,1],[8],[]$
6. $[8,2,6,7,5,1,4,3] \to [2,6,7,5,1,4],[8],[]$
7. $[8,2,6,7,5,1,4,3] \to [2,6,7,5,1,4,3],[8],[]$
8. $[2,6,7,5,1,4,3] \to [],[2],[6]$
9. $[2,6,7,5,1,4,3] \to [],[2],[6,7]$
10. $[2,6,7,5,1,4,3] \to [],[2],[6,7,5]$
11. $[2,6,7,5,1,4,3] \to [1],[2],[6,7,5]$
12. $[2,6,7,5,1,4,3] \to [1],[2],[6,7,5,4]$
13. $[2,6,7,5,1,4,3] \to [1],[2],[6,7,5,4,3]$
14. $[6,7,5,4,3] \to [],[6],[7]$
15. $[6,7,5,4,3] \to [5],[6],[7]$
16. $[6,7,5,4,3] \to [5,4],[6],[7]$
17. $[6,7,5,4,3] \to [5,4,3],[6],[7]$
18. $[5,4,3] \to [4][5][]$
19. $[5,4,3] \to [4,3][5][]$
20. $[4,3] \to [3][4][]$

□

Exercise 5.9

Given three SORTED (in ascending order) arrays $A[1..n], B[1..n]$, and $C[1..n]$, each containing n numbers, give an $\mathcal{O}(\log n)$-time algorithm (again, counting the number of comparisons) to find the n^{th} smallest number of all $3n$ elements in arrays A, B, and C.

Solution. Let $i = \lfloor n/3 \rfloor$ and define $a = A[i]$, $b = B[i]$, $c = C[i]$. Without loss of generality, say $a = \min(a,b,c)$. Now, there are at most $3\lfloor n/3 \rfloor < n$ elements in A, B, and C less than a, hence we can remove these $\lfloor n/3 \rfloor$ smallest elements from A.

$$A = [\underbrace{a_0, a_1, \ldots, a_{i-1}}_{\text{remove these}}, a_i, \ldots, a_{n-1}]$$

In the same vein, set $i' = 2\lfloor n/3 \rfloor$ and let $b' = B[i']$. Since b is sorted and $a < b$, $a < b < b'$, hence there are at least $\lfloor n/3 \rfloor + 2\lfloor n/3 \rfloor$ elements less than b', and we can remove all elements to the right of b'

$$B = [b_0, b_1, \ldots, b_{i'-1}, \underbrace{b_{i'}, \ldots, b_{n-1}}_{\text{remove these}}]$$

And similarly for C

$$C = [c_0, c_1, \ldots, c_{i'-1}, \underbrace{c_{i'}, \ldots, c_{n-1}}_{\text{remove these}}]$$

Leaving us with 3 arrays

$$\begin{aligned}
A' &= [a_i, a_{i+1}, \ldots, a_{n-1}] \\
B' &= [b_0, b_1, \ldots, b_{i'-1}] \\
C' &= [c_0, c_1, \ldots, c_{i'-1}]
\end{aligned}$$

each of length $2\lfloor n/3 \rfloor$. This removal requires a constant number of comparisons, and we will perform this $\mathcal{O}(\log n)$ times (since we remove approximately a third of the elements at each step), for a total $\mathcal{O}(\log n)$ comparisons.

$\square$