CSCI 3104 Algorithms Name: Jeremy M. Hein
Problem Set 4
Fall 2020, CU Boulder ID: 810952267

Instructions

- This problem set is **open book**: you may refer to the lectured material found on Canvas and the recommended books to help you answer the questions.
- This problem set is an **individual effort**. You must arrive at your answers independently and write them up in your own words. Your solutions should reflect your understanding of the content.
- **Posting questions to message boards or tutoring services including, but not limited to, Chegg, StackExchange, etc., is STRICTLY PROHIBITED. Doing so is a violation of the Honor Code.**
- Your solutions must be submitted typed in LaTeX, **handwritten work is not accepted**. If you want to include a diagram then we do accept photos or scans of hand-drawn diagrams included with an appropriate `\includegraphics` command. It is your responsibility to ensure that the photos you obtain are in a format that pdflatex understands, such as JPEG.
- The template tex file has carefully placed comments (% symbols) to help you find where to insert your answers. There is also a **STUDENT DATA** section in which you should input your name and ID, this will remove the warnings in the footer about commands which have not been edited. You may have to add additional packages to the preamble if you use advanced LaTeX constructs.
- You must CITE any outside sources you use, including websites and other people with whom you have collaborated. You do not need to cite a CA, TA, or course instructor.
- Take care with time, we do not usually accept problem sets submitted late.
- Take care to upload the correct pdf with the correct images inserted in the correct places (if applicable).
- Check your pdf before upload.
- **Check your pdf before upload.**
- **CHECK YOUR PDF BEFORE UPLOAD.**

Quicklinks: 1 (2a) (2b) (2c) (3a) (3b) (3c) (3d) (4a) (4b) (4c)

Problem 1

Provide a one-sentence description of each of the components of a divide and conquer algorithm. Recall binary search and explain how the design of it conforms to the divide and conquer paradigm.

Each component of a divide and conquer algorithm can be described as follows:

1. Divide: break a problem instance into several smaller instances of the same problem
2. Conquer: if a smaller instance is trivial, solve it directly; otherwise, divide again
3. Combine: use the results of smaller instances together to make a solution to a larger instance

Binary search is an algorithm for finding an element in a sorted list, and is a good example of a divide and conquer algorithm. It follows the same steps as below:

1. Divide: first we find the middle element of our list. This divides our list into two sub-lists - one will all elements less than the middle element and one with elements all greater than the middle element.
2. Conquer: the conquer step is if the middle element is equal to the element we are looking for. If it is, we "solve" by returning the index of the element. If our element is less than the middle element then we divide again, by repeating our search on the half of the list that is less than the middle element. If our element is greater than the middle element, then we divide again by repeating our search on the half of the list that is greater than the middle element. This process is repeated on smaller and smaller lists until we find our element.
3. Combine: our combine step uses the results of the smaller lists to find the index of the element in our array. Once the index is found, it returns up the stack and returns the index, which is the solution to the larger instance.

Problem 2

For 2a, 2b, and 2c **you must** use the pseudocode for QUICKSORT and PARTITION on page 3 of Week3.pdf of the lecture notes and the array $A = [2, 6, 5, 7, 1, 9, 4]$. Suppose that we start by calling QUICKSORT$(A, 1, 7)$.

(2a) What is the value of the pivot in the call PARTITION$(A, 1, 7)$?

The pseudocode for PARTITION requires inputs A, s, e, where A is our array, s is the starting index, and e is the index of the pivot (the last element in the array in this implementation). The call PARTITION$(A, 1, 7)$ means the element $A[7]$ is our pivot, and $A[7] = 4$.

For 2a, 2b, and 2c **you must** use the pseudocode for QUICKSORT and PARTITION on page 3 of Week3.pdf of the lecture notes and the array $A = [2, 6, 5, 7, 1, 9, 4]$. Suppose that we start by calling QUICKSORT$(A, 1, 7)$.

(2b) What is the index of the pivot value (returned) at the end of that call to PARTITION?

I will show each step below, including the values in the array as well as the current place of each index:

1. Our first iteration in the for loop has index i and j at the start of our list, and pivot p equal to 4:

$$2_{ij} \quad 6 \quad 5 \quad 7 \quad 1 \quad 9 \quad 4_p$$

2. 2 is less than or equal to 4, so both i and j are incremented. Note i and j are the same element, so 2 remains where it is in this case:

$$2 \quad 6_{ij} \quad 5 \quad 7 \quad 1 \quad 9 \quad 4_p$$

3. 6 is not less than or equal to 4, so j is incremented and we continue looping until we find an element that is less than or equal to 4:

$$2 \quad 6_i \quad 5_j \quad 7 \quad 1 \quad 9 \quad 4_p$$

$$2 \quad 6_i \quad 5 \quad 7_j \quad 1 \quad 9 \quad 4_p$$

$$2 \quad 6_i \quad 5 \quad 7 \quad 1_j \quad 9 \quad 4_p$$

4. Here 1 is less than or equal to 4, so the values at index i and j are swapped, and both indices are incremented:

$$2 \quad 1 \quad 5_i \quad 7 \quad 6 \quad 9_j \quad 4_p$$

5. On the last iteration we see that 9 is not less than or equal to 4, so j is incremented and we exit the loop. Lastly, $A[e]$ and $A[i]$ are swapped, and we return i. So our final array is below and in this case return 3, as $i = 3$:

$$2 \quad 1 \quad 4_{pi} \quad 7 \quad 6 \quad 9 \quad 5$$

For 2a, 2b, and 2c **you must** use the pseudocode for QUICKSORT and PARTITION on page 3 of Week3.pdf of the lecture notes and the array $A = [2, 6, 5, 7, 1, 9, 4]$. Suppose that we start by calling QUICKSORT$(A, 1, 7)$.

(2c) On the next recursive call to QUICKSORT, what subarray does PARTITION evaluate? Give the state of the whole of A at the start of this call *and* the indices specifying the subarray.

As shown in $2(b)$, the state of the array is as follows:

$$2 \quad 1 \quad 4 \quad 7 \quad 6 \quad 9 \quad 5$$

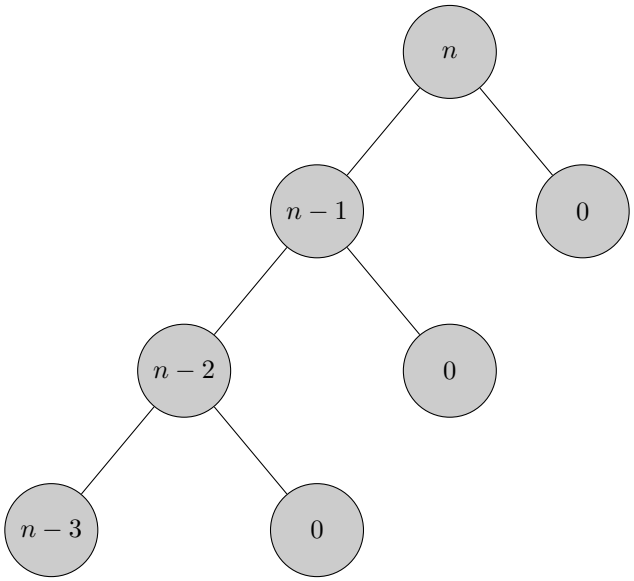
In this case, the value of q is equal to 3, and the next call to QUICKSORT is QUICKSORT$(A, s, q-1)$. So our call would be QUICKSORT$(A, 1, 2)$ and the subarray is $[2, 1]$.

Problem 3

Recall that the PARTITION algorithm encodes a fixed choice of pivot: $A[e]$ is always chosen. In this problem we ask you to consider alternative implementations of PARTITION that differ in how they choose the pivot.

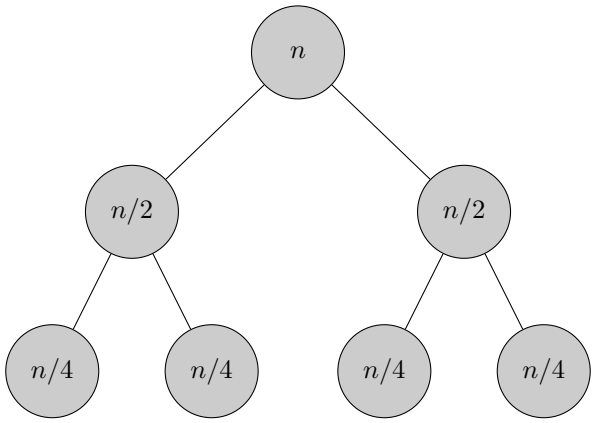
(3a) Given an array consisting of n distinct elements, what choice of pivot will result in the best partitioning, and which one will result in the worst partitioning? Justify your answers in terms of the shape of the recursion tree and the running time of QUICKSORT subject to these pivot choices. You do not need to give a full, formal proof of any statement about running time.

The worst partitioning would occur when we have a split in the form $(n-1, 0)$. In this case, we're only reducing the size of the sub-problems by one element. The first recursive call is on a list of size $n-1$, and our second recursive call is an empty list. The resulting shape is an unbalanced recursion tree that looks like this:



The resulting run time in this scenario is $\Theta(n^2)$, which we could prove (and have in lectures) by unrolling the recursion $T(n) \leq T(n\text{ - }1) + T(0) + (n) = T(n\text{ - }1) + \Theta(n)$.

The best partitioning would occur when we have a split in the form $(\lfloor \frac{n-1}{2} \rfloor, \lceil \frac{n-1}{2} \rceil)$. In this case, we are (more or less) splitting our sub-problems equally and both of the recursive calls are on lists of size $\frac{n-1}{2}$. The resulting shape is a balanced recursion tree that looks like this:



Note that since $\frac{n-1}{2} < \frac{n}{2}$, it is valid to look at the recursive relation $T(n) \leq 2T(n/2) + \Theta(n)$ as an upper bound. This relation is depicted in the recursion tree above. This recursive relation results in a run time of $\Theta(nlog(n))$, which can be proved using unrolling, the tree method, or the master method.

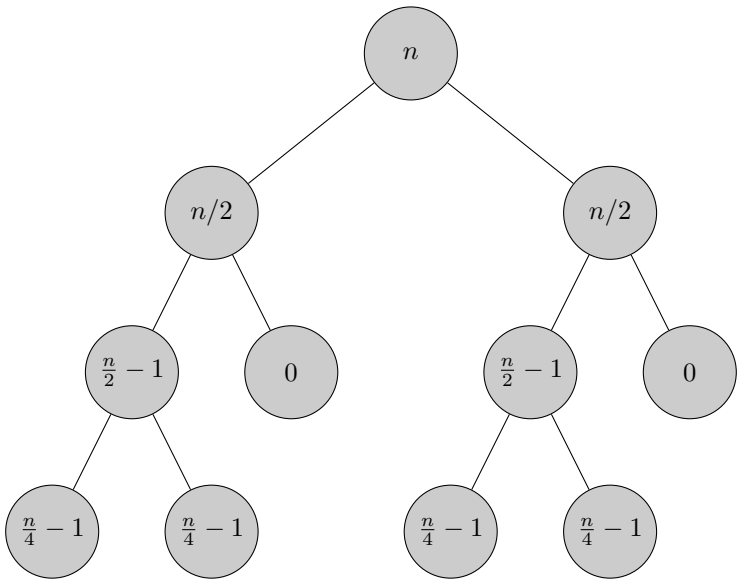
Recall that the PARTITION algorithm encodes a fixed choice of pivot: $A[e]$ is always chosen. In this problem we ask you to consider alternative implementations of PARTITION that differ in how they choose the pivot.

(3b) Suppose that we modify PARTITION(A, s, e) so that it chooses the median element of $A[s..e]$ in calls that occur in nodes of even depth of the recursion tree of a call QUICKSORT$(A, 1, \text{len}(A))$, and it chooses the maximum element of $A[s..e]$ in calls that occur in nodes of odd depth of this recursion tree.

Assume that the running time of this modified PARTITION is still $\Theta(n)$ on any subarray of length n. You may assume that the root of a recursion tree starts at level 0 (which is an even number), its children are at level 1, etc.

Write down a recurrence relation for the running time of this version of QUICKSORT given an array n distinct elements and solve it asymptotically, i.e. give your answer as $\Theta(f(n))$ for some function f. Show your work.

I'll first start by using a recursion tree to represent the pattern. Note that in the case of an even node, we'll have a split in the form $(\lfloor \frac{n-1}{2} \rfloor, \lceil \frac{n-1}{2} \rceil)$. Similar to $3(a)$, I will represent this as $\frac{n}{2}$ since $\frac{n-1}{2} < \frac{n}{2}$ and it is valid to look at this as an upper bound.



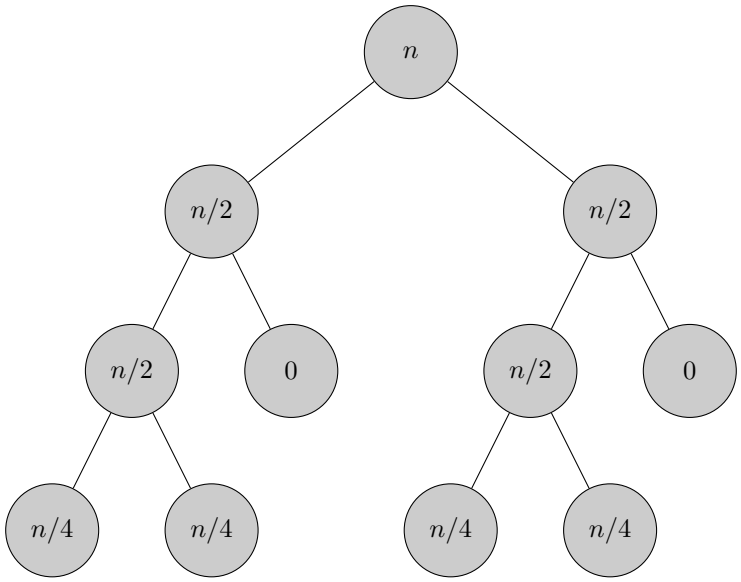
There are a few things to notice here. The first is that the children from the even nodes are splitting sub-problems (roughly) in half, while the children from the odd nodes are only reducing the sub-problems by 1. To put this simply, the even nodes are doing much more "dividing and conquering" in relation to the odd nodes. Referencing Lecture 6 from Professor Davies, he states "In fact, getting a "good" split a constant fraction of the time is enough to

obtain $\Theta(nlog(n))$ running time." He goes on to say that proving this is too detailed for our course.

So my final answer is $\Theta(nlog(n))$, as it was stated in lecture. To put this in the form of a recursive relationship, we could write:

$$T(n) = \left(\frac{1}{2}\right) 2T(n/2) + \left(\frac{1}{2}\right) T(n-1) + \Theta(n)$$

Where we get a "good split" $(n/2)$ half the time and a "bad split" $(n-1)$ the other half the time. In order to solve, we can further bound our run time by making the assumption that our "bad splits" don't reduce the problem at all. This is valid as they are only reducing the problem size by 1 anyways. In this scenario, our tree would look like this:



Each level of the tree is doing n work (so without our assumption, it must be $\leq n$ work each level). So then the question is how deep is the tree? Well, if we were splitting the problem in half on every recursive call we would have sub-problem size of $i = \frac{n}{2^i}$, and we would hit $n = 1$ when $\frac{n}{2^i} = 1$, or $i = \lceil \log_2(n) \rceil$ levels. Since this is only occurring half the time, then we would have twice as many levels or $2log_2(n)$ levels. And since each level is $\leq n$, work, we can conclude that $T(n) \leq O(nlog(n))$.

Recall that the PARTITION algorithm encodes a fixed choice of pivot: $A[e]$ is always chosen. In this problem we ask you to consider alternative implementations of PARTITION that differ in how they choose the pivot.

(3c) Compare the running time of the modified version of QUICKSORT given by your answer to (3b) with the running time of the usual QUICKSORT (as stated in Week3.pdf) under the assumption that the median element is the pivot every time that we did in class.

If the two running times are Θ of each other then compare the implicit constant hidden in the Θ notation with a short non-rigorous explanation.

If the two running times are not Θ of each other, identify the one that is asymptotically larger and explain why with a short, non-rigorous explanation.

The two run times are Θ of each other. I think the implicit constant hidden in the Θ is referencing the rate at which the sub-problem size is being reduced. The best-case of quicksort is reducing the size of A by a factor of 2. The modified version of quicksort in 3(b) is doing the same thing, but it's only doing it on every other level. But it's still reducing by a factor of 2 so that's why we see the same run time.

Recall that the PARTITION algorithm encodes a fixed choice of pivot: $A[e]$ is always chosen. In this problem we ask you to consider alternative implementations of PARTITION that differ in how they choose the pivot.

(3d) Suppose now that we modify PARTITION so that the algorithm chooses the maximum as the pivot for three levels of recursion, then the median as the pivot at the next level, and this repeats so the maximum is the pivot for the next three levels, then the median for one level, etc.

What is your estimate of the asymptotic running time? We expect you to provide an explanation of your answer, but (unlike the previous subproblems) we do not require a formal proof and we do not require you to state or solve a specific recurrence relation for this subproblem.

It would still be $\Theta(nlog(n))$, as we are still getting "good splits" a constant fraction of the time.

Problem 4

Consider a chaining hash table A with b buckets that holds data from a fixed, finite universe U.

(4a) State the simple uniform hashing assumption and give one reason why this is a useful assumption and one reason why it is a poor assumption. Use full sentences.

Consider a chaining hash table A with b buckets that holds data from a fixed, finite universe U.

(4b) Recall the definition of worst-case analysis, and consider starting with A empty and inserting n elements into A under the assumption that $|U| \leq bn$. **Do not assume the simple uniform hashing assumption for this subproblem.**

(i) What is the worst case for the number of hash collisions? Give an exact answer and justify it.

(ii) Calculate the worst-case amortized cost of these n insertions into A, and give your answer as $\Theta(f(n))$ for a suitable function f. Justify your answer.

Consider a chaining hash table A with b buckets that holds data from a fixed, finite universe U.

(4c) Consider the task of designing a data structure to hold the percentage score as an integer in the range 0..100 for every student in the course *DTDJ4215: Algorithms* that has 256 students. No course at *CU Cpvmefs*, where you work, can ever have more than 256 enrolled students.

The requirements are that reasonably efficient implementations of insertion and printing the scores in increasing order are important, and to have low memory overhead. Your colleague suggests that a dynamically-sized open addressing hash table with a highly sophisticated *disfrobunation* probing strategy will be a good implementation.

Your are knowledgeable enough to know of a better option. Choose an alternative data structure write a (polite) email to your colleague that explains your choice of data structure and why it fits the requirements better.
