TCSS 343 - Week 2

Jake McKenzie

August 4, 2018

Divide and Conquer

"If anyone on the verge of action should judge himself according to the outcome, he would never begin."   Søren Kierkegaard

"A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools."

. . .

Douglas Adams

"With your career don't expect anything revolutionary. If you're used to learning it gets easier to learn. Assume walking in that you can learn whatever it is that you need to learn to do the job. If you go to college you should be learning how to learn."

. . .

Paul Sidenblad (Satalite Computer Engineer)

On the previous worksheet I had you write code for binary search. I want you to become more and more comfortable with trees and $\log n$ time. Work is just a function of "How many problems we have" and "the amount of work for each problem". Recursions trees is an extremely powerful way of illustrating the work of recurrence problems.
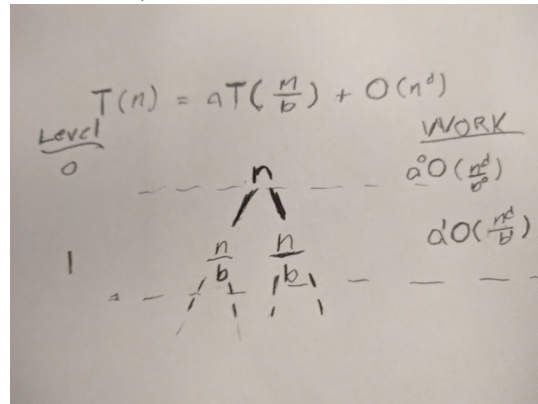
This will appear to be a digression but they are actually tightly connected ideas. Now remember the geometric series?

$$a + ar + ar^2 + ar^3 + \cdots + ar^{n-1} = a\frac{1 - r^n}{1 - r}$$

Good, yeah I know who could have forgotten it!? It's one of my favourite series and it should be your best friend right about now.The geomtric series is a series which progresses by some multiplicative factor.

0. The first thing I will have you do is find $O(a\frac{1-r^n}{1-r})$ for two cases. When $r > 1$ and $r < 1$ where a is some function of n.

1. Now that you've solved for those worst-case runtimes keep them for later because they're going to be useful! Now hold onto your butts because we're going to be having a lot of fun today. Complete this tree given the following recurrence for two more levels(The a need not be the same as the previous a...sorry a habit of notation):

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$$

WORK

Level 0

n

$a^0 O(n^d)$

$\frac{n}{b}$ $\frac{n}{b}$

$a^1 O\left(\frac{n^d}{b}\right)$

1

$\frac{n}{b}$ $\frac{n}{b}$

2. Take your previous result, at what level does the tree terminate?

3. Please, please try your darndest to express the total work as a summation. Remember: Work is just a function of "How many problems we have" and "the amount of work for each problem".

4. Solve for the three worst cases of the summation you found, when $r < 1$, $r = 1$ and $r > 1$.

"The purpose of computing is insight, not numbers" - Richard Hamming

5. Alright, alright I know...this isn't a math class this is an algorithms course! But if you solved those problems you now have a wonderful grab bag of results that you can use for now and forever! Now let's do some code.

Richard Hamming is one of my idols. He was a computer engineer who did a lot of really cool stuff and today we're going to calculate the distance that is named after him!

The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Given two integers x and y, can you calculate this distance for me? Well...if not for me at least for Richard's cat.

Note:
$0 \leqslant x,y < 2^{31}$
Example:

Input: $x = 1$, $y = 4$

Output: 2

Explanation:

```
1    (0 0 0 1)
4    (0 1 0 0)
        ↑   ↑
```

The above arrows point to positions where the corresponding bits are different(next page).

Write for me (or Richard's cat :3) some code to compute the hamming distance between two integers.

6. I'm going to ask you a lot of pop corn questions for now on but that doesn't mean I want you to treat them lightly. This is to test **YOUR** understanding not the **INTERNET**. Please do not google the answers. I know you can. I could when I was in this class but please, please don't. Think hard about these now and I'll post the solutions later.
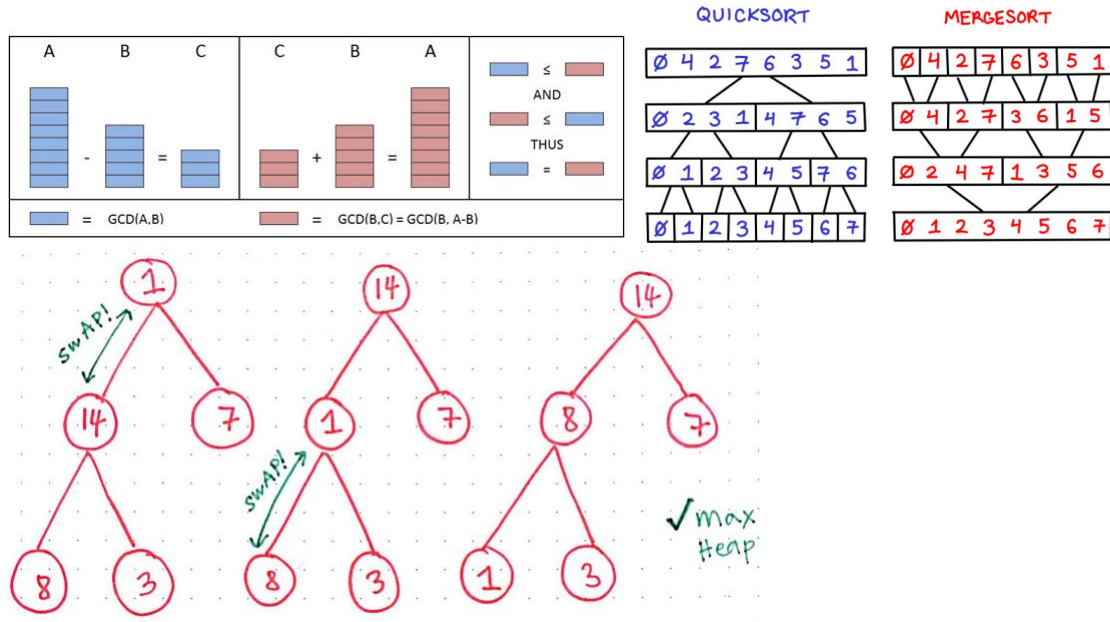
Which of the following algorithms is **NOT** a divide and conquer algorithm by nature?

a) Euclidean algorithm to compute the GCD

b) Heap Sort

c) Merge Sort

d) Quick Sort

```
int main()
{
    int x, y, m, n;
    scanf ("%d %d", &x, &y);
    /* x > 0 and y > 0 */
    m = x; n = y;
    while (m != n)
    {
        if(m>n)
            m = m - n;
        else
            n = n - m;
    }
    printf("%d", n);
}
```

7. Consider the following c program above, what does it compute?

a) The greatest common divisor of x and y

b) $x + y$ using repeated subtraction

c) $x \mod y$

d) The least common multiple of x and y

8. Maximum Subarray Sum problem is to find the subarray with maximum sum. For example, given an array {4, -10, 7, 9, -8, -5, 2, 0, -2, 8}, the maximum subarray sum is 16. The naive solution for this problem is to calculate sum of all subarrays starting with every element and return the maximum of all. We can solve this using Divide and Conquer, what will be the worst case time complexity using Divide and Conquer.
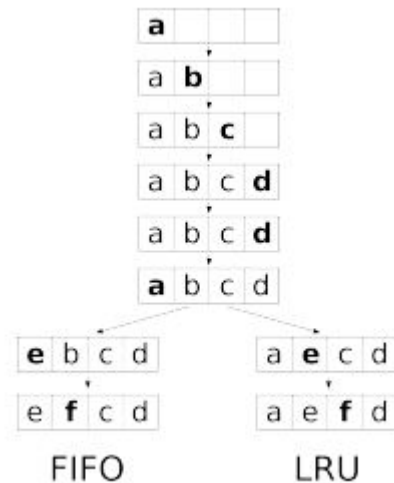
a) $O(n)$

b) $O(n \log n)$

c) $O(\log n)$

d) $O(n^2)$

9. Given two integers x and n, say we wrote a function to compute x^n using divide and conquer. What is the recurrence relation and worst case runtime of our function? Please also attempt to write some code that accomplishes this.

a) $T(n) = T(n/2) + c$; $T(n) \in O(n \log n)$

b) $T(n) = T(n/2) + n$; $T(n) \in O(n \log n)$

c) $T(n) = 2T(n/2) + c$; $T(n) \in O(\log n)$

d) $T(n) = T(n/2) + c$; $T(n) \in O(\log n)$

FIFO LRU

A. In undergraduate algorithms we typically worry about worst case analysis, but problems can be hard when it doesn't matter. In a computer there is a small speedy memory (the "cache) and a big sluggish memory (typically a "harddisk"). Imagine there are two competing programs, one uses "Least Recently Used" algorithm and another using "First In First Out" that periodically issue read and write requests to data stored on "pages in the big sluggish memory. If the requested page is also in the cache, then we can happily say it can be accessed directly! If not :(we get what is known as a "page fault a.k.a. "cache miss the requested page needs to be brought into the cache, and this requires evicting an incumbent page. The key algorithmic question is then: **which page should we evicted**? Which algorithm do you think has the worst case runtime and which do you think works better in practice? Why? Is there an intuitive answer to this reality?

a) Least Recently Used

b) First in First Out

B. I have this list S in sorted order where $S_{ij} = (s_i, s_{i+1}, \ldots, s_j) \; \forall \; 1 \leqslant i < j \leqslant n$. Take my word for it that the probability that randomized quicksort compares each element s_i and s_j at least once is given by this equation where $\Pr[s_i$ is compared to $s_j] \leqslant \frac{1}{j-i-1} + \frac{1}{j-i-1}$. I found this by using

Find the **WORST CASE RUNTIME** of the expected value E[X] if

$$E[X] \leqslant \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{j-i-1} + \frac{1}{j-i-1}$$

C. Any comparison based sorting algorithm must make at least f(n) comparisons. Show that is f(n) $\in \Omega(n \log n)$. Hint: How many possible ways are there are there of choosing, without replacement n things from n where order matters.

a) $\log n!$

b) $n^2 \log n$

c) $\log n^n$

d) n