Design and Analysis of Algorithms

Notes by Arjun Maneesh Agarwal

based on course by Siddharth Pritam

Texbooks to be used are CLRS and Algorithms by Jeff Ericson.

Table of Contents

Proof of correctness.

Claim 0.1. $m \mid n$, $\gcd(m, n) = m$.

Trivial.

Claim 0.2. $\gcd(m, n) = \gcd(n \bmod m, m)$

Assume $\gcd(m, n) = a \Rightarrow m = ap, n = aq, p \nmid q$.

Thus, $\gcd(m, n) = \gcd(ap, aq) = \gcd(a(q \bmod p), ap)$ ■

Idea : How Fast?

Let's try to see how fast doesn $n + m$ decrease. Let the next level be m' and n'.

Using the modulo condition, we can get $3(m' + n') = m' + n' + 2(m' + n') < p + p + 2q = 2(p + q)$.

This implies the algorithm is lower bounded by $\log_{\frac{3}{2}}(m' + n')$

DFA to NFA

```
def delta(a, S, nfaDelta):
ans = []
for s in S:
    k = nfaDelta(a,s)
    if k not in ans:
        push k ans
```

DFA to NFA

```
return ans
def func(alphabet, nfaDelta, S, nfaFinal):
    initial = S
    dfaDelta = []
    final = []
    unVisited = Queue [S]
    checked = []
    while unVisited is not empty:
    state = pop unVisited
    for s in state:
    if state in nfaFinal:
        push s final
        break
        for a in alphabet:
            k = delta(a, S)
            DFA[state][a] = k
            if k in checked: +continue
            else:
                push k unVisited
    return(alphabet, checked, initial, dfaDelta, final)
```

Exercise

- n vs $n \log n$
- $n^{\log^2(n)}$ vs $2^{\log^2(n)}$
- $n^{\log \log n}$ vs $2^{\log n \log \log n}$

Solution

We get:

- $n = O(n \log n)$
- $n^{(\log(n))^2} = O\left(2^{\log^2(n)}\right)$ (by switching to same base)
- $n^{\log \log n} = \Theta\left(2^{\log n \log \log n}\right)$ (same trick!)

Exercise

If $f(n) = O(g_1(n))$ and $f(n) = O(g_2(n))$ is $f(n) = O(g_1(n) + g_2(n))$

Solution

Trivial enough really!

1. Recuerence relation!

1.1. Substitution Method

Given a recuerence, we guess the solution and then prove its correctness by induction.

Example

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Solution

Just induction on $T(n) = cn\log(n)$

Example

$$T(n) = T\left(\left\lceil\frac{n}{2}\right\rceil\right) + T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + 1$$

Solution

Taking $T(n) = cn$ doesn't work as we get $T(n) = cn + 1$ which doesn't work.

Let $T(n) = cn - d$ and we will deal with it later.

$$\begin{aligned} T(n) &= c\left\lceil\frac{n}{2}\right\rceil + c\left\lfloor\frac{n}{2}\right\rfloor - 2d + 1 \\ &= cn - 2d + 1 \\ \text{should} &= cn - d \end{aligned}$$

This works out for $d = 1$ and hence, we 'guess' $cn - 1$.

Example

$$T(n) = 2T(\sqrt{n}) + \log(n)$$

Solution

Take $n = 2^m$

$$T(2^m) = 2\left(T\left(2^{\frac{m}{2}}\right)\right) + 2$$
$$S(m) = 2S\left(\frac{m}{2}\right) + m$$

This gives $S(m) = O(m \log(m))$

Thus, $T(n) = O(\log(n) \log(\log(m)))$

Exercise

$$T(n) = T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + \Theta(n)$$

Solution. We can take the worst case scenario and say the recursion tree has depth $\log_{\frac{10}{9}} n$. Every level of recursion has total work $\Theta(n)$.

The time speant at leaves is 2^d where d is the depth. Thus, $2^{\log_{\frac{10}{9}} n}$

Thus, we take total time

$$cn \log_{\frac{10}{9}}(n) + 2^{\log_{\frac{10}{9}} n}$$
$$= O(n \log n) + n^{\frac{1}{\log\left(\frac{10}{9}\right)}}$$
$$= O(n^7)$$

Thus, we have a time complexity of $O(n^7)$.[1] ■

1.2. Master Method

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

Making the recursion tree, we have $\log_b(n)$ depth. The work at some depth k will be $a^k\left(\frac{n}{b^k}\right)^d$. We will have $a^{\log_{b(n)}}$ leaves.

Thus, the total work will be

$$T(n) = n^d\left(\underbrace{1 + \frac{a}{b^d} + \frac{a^2}{b^{2d}} + \dots}_{\log_b(n)}\right) + a^{\log_b n}$$

Note,

TODO. *Copy from CS-161 slides!*

[1] This is an extreme upper bound. The actual number comes much closer to $n \log(n)$ as the longest and shortest branches differ by a lot!

Theorem(Master's Theorem) 1.1.

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^d) \Rightarrow T(n) = \begin{cases} O(n^{\log_b(a)}) & \text{if } \frac{a}{b^d} > 1 \\ O(n^d) & \text{if } \frac{a}{b^d} < 1 \\ O(n^d \log(n)) & \text{if } \frac{a}{b^d} = 1 \end{cases}$$

[2]

2. Divide And Conquer Algorithms

Idea

1. Break the input into smaller, disjoint parts.
2. Recurse on the parts.
3. Combine the solutions.

Step 1 and step 3 are normally where the work happens. For example in merge sort, we do almost no work in spliting and linear work in combining. On the other hand in quick sort, we do linear work in splitting and no work in combining.

2.1. Convex Hull

Given n points $p \in \mathbb{R}^2$ where $p = (x, y)$, the convex hull is the smallest polygon such that a line between any two points in inside or on the boundry of the hull.

We can strech a rubber band over the points (which are nails) and let it go.

A dumb algorithm is just checking every combination. This is roughly $O(n!)$.

Convex Hull in cubic time

For all $p, q \in P \subseteq \mathbb{R}^2$
  Check if $\text{sign}(p, q, r)$ is same for all $r \in P \setminus \{p, q\}$
    Include p, q in solution

We have already gone from $n!$ to $O(n^3)$.

Another option is to pick the lowest point and then sort by angles and pick the lowest points, one by one. This is called a Jarvis Search.

Although, we have included this in the divide and conquer section for a reason!

We will divide the points in two parts and then try to recombine.

[2]There is absolutly nothing this theorem masters. It was name orignating from CLRS and other than working for some 'common algorithms' it is far inferior to just making the tree yourself.

Divide and Conqour Convex Hull combining

```
i = 1
j = 1
while (y(i, j+1) > y(i, j) or y(i-1, j) > y(i, j))
    if y(i, j+1) > y(i, j): move right pointer ↷
    j = j + 1 mod q {q := # in the right hand side}
else
    i = i-1 mod (p) {p := # in the left hand side}
return (a_i, b_j) as upper tangent
```