

Session 3

Corbet Elkins

October 23, 2026

Purdue University

Questions over lecture content?

See title

Karatsuba's Algorithm

Recall last week I mentioned that we can multiply two integers in $O(n \log n \log \log n)$.

Today I will go over Karatsuba's algorithm which gives an $O(n^{1.585})$ runtime.

Goal: given x, y whose length in binary is a power of 2, return xy .

Karatsuba's Algorithm

Let $x = (x_1, x_2)$, $y = (y_1, y_2)$ where $x_1, y_1 \in \{0, 1\}^{n/2}$ are the $n/2$ most significant bits of x, y respectively.

That is $x = x_1 2^{n/2} + x_2$, $y = y_1 2^{n/2} + y_2$. Hence:

$$xy = x_1 y_1 2^n + (x_1 y_2 + x_2 y_1) 2^{n/2} + x_2 y_2$$

However, we can rewrite the middle term:

$$x_1 y_2 + x_2 y_1 = (x_1 + x_2)(y_1 + y_2) - x_1 y_1 - x_2 y_2$$

Why is this useful?

Karatsuba's Algorithm

Let $x = (x_1, x_2)$, $y = (y_1, y_2)$ where $x_1, y_1 \in \{0, 1\}^{n/2}$ are the $n/2$ most significant bits of x, y respectively.

That is $x = x_1 2^{n/2} + x_2$, $y = y_1 2^{n/2} + y_2$. Hence:

$$xy = x_1 y_1 2^n + (x_1 y_2 + x_2 y_1) 2^{n/2} + x_2 y_2$$

However, we can rewrite the middle term:

$$x_1 y_2 + x_2 y_1 = (x_1 + x_2)(y_1 + y_2) - x_1 y_1 - x_2 y_2$$

Why is this useful? We have turned 4 multiplications into only 3!

Karatsuba's Algorithm

```
def multiply(x[1..n],y[1..n]):  
    If  $n = 0$  return 0  
    If  $n = 1$  return  $x[1] * y[1]$   
    Define  $x_1, x_2, y_1, y_2$  as before  
    Let  $x_3[1..n/2 + 1] = x_1 + x_2$   
    Let  $y_3[1..n/2 + 1] = y_1 + y_2$   
    Let  $Z_i = \text{multiply}(x_i, y_i)$  for  $i \in \{1, 2, 3\}$   
    return  $2^n * Z_1 + 2^{n/2} * (Z_3 - Z_1 - Z_2) + Z_2$ 
```

What's the runtime for a call not including recursive calls?

What's the runtime for a call not including recursive calls? $O(n)$.

What's the runtime for a call not including recursive calls? $O(n)$.

How many recursive calls of what size?

What's the runtime for a call not including recursive calls? $O(n)$.

How many recursive calls of what size? 3 calls of size $n/2$.

$$T(n) = 3T(n/2) + O(n)$$

What's the runtime for a call not including recursive calls? $O(n)$.

How many recursive calls of what size? 3 calls of size $n/2$.

$$T(n) = 3T(n/2) + O(n)$$

How much work is done on each level?

Karatsuba Runtime

What's the runtime for a call not including recursive calls? $O(n)$.

How many recursive calls of what size? 3 calls of size $n/2$.

$$T(n) = 3T(n/2) + O(n)$$

How much work is done on each level? $(3/2)^i n$ for level i

Karatsuba Runtime

What's the runtime for a call not including recursive calls? $O(n)$.

How many recursive calls of what size? 3 calls of size $n/2$.

$$T(n) = 3T(n/2) + O(n)$$

How much work is done on each level? $(3/2)^i n$ for level i

How many levels?

Karatsuba Runtime

What's the runtime for a call not including recursive calls? $O(n)$.

How many recursive calls of what size? 3 calls of size $n/2$.

$$T(n) = 3T(n/2) + O(n)$$

How much work is done on each level? $(3/2)^i n$ for level i

How many levels? $\log(n)$

Karatsuba Runtime

$$\begin{aligned} T(n) &\approx \sum_{i=0}^{\log(n)} (3/2)^i n \\ &\approx n \sum_{i=0}^{\log(n)} (3/2)^i \end{aligned}$$

$$\begin{aligned}T(n) &\approx \sum_{i=0}^{\log(n)} (3/2)^i n \\&\approx n \sum_{i=0}^{\log(n)} (3/2)^i \\&\approx n(3/2)^{\log(n)}\end{aligned}$$

$$\begin{aligned}T(n) &\approx \sum_{i=0}^{\log(n)} (3/2)^i n \\&\approx n \sum_{i=0}^{\log(n)} (3/2)^i \\&\approx n(3/2)^{\log(n)} \\&\approx n(3/2)^{\log_{3/2}(n) / \log_{3/2}(2)}\end{aligned}$$

$$\begin{aligned}T(n) &\approx \sum_{i=0}^{\log(n)} (3/2)^i n \\&\approx n \sum_{i=0}^{\log(n)} (3/2)^i \\&\approx n(3/2)^{\log(n)} \\&\approx n(3/2)^{\log_{3/2}(n) / \log_{3/2}(2)} \\&\approx nn^{1/\log_{3/2}(2)}\end{aligned}$$

Karatsuba Runtime

$$\begin{aligned}T(n) &\approx \sum_{i=0}^{\log(n)} (3/2)^i n \\&\approx n \sum_{i=0}^{\log(n)} (3/2)^i \\&\approx n(3/2)^{\log(n)} \\&\approx n(3/2)^{\log_{3/2}(n) / \log_{3/2}(2)} \\&\approx nn^{1/\log_{3/2}(2)} \\&\approx n^{1.585}\end{aligned}$$

Problems

Problem

Suppose you have a stack of n pancakes of distinct radii. You wish to stack the pancakes such that they are arranged smallest to largest. You have a spatula and for any $k \leq n$ you can reverse the order of the top k pancakes. Give an algorithm which can order the pancakes in $O(n)$ flips.¹

Problem (Savitch's Theorem)

Given a directed graph $G = (V, E)$, and two points $s, t \in V$, give an $O(\log^2(|V|))$ space algorithm to determine if there exists a path from s to t , the runtime is not important. (Hint: consider an algorithm that takes in s, t and a number k and returns true if there is an $s - t$ path of length at most k).

¹Fun Fact: Bill Gates wrote a paper on this problem with Prof. Psomas' advisor

Savitch's Theorem

- Recall that given a (directed) graph $G = (V, E)$, with $|V| = n$, $|E| = m$, and $s, t \in V$, we can determine if there exists an $s \rightarrow t$ path in $O(n + m)$ time.

Savitch's Theorem

- Recall that given a (directed) graph $G = (V, E)$, with $|V| = n$, $|E| = m$, and $s, t \in V$, we can determine if there exists an $s \rightarrow t$ path in $O(n + m)$ time.
- What if our graph is massive, and we have limited hardware, now we instead want to focus on space bounded computation.

Savitch's Theorem

- Recall that given a (directed) graph $G = (V, E)$, with $|V| = n$, $|E| = m$, and $s, t \in V$, we can determine if there exists an $s \rightarrow t$ path in $O(n + m)$ time.
- What if our graph is massive, and we have limited hardware, now we instead want to focus on space bounded computation.
- Any guesses on how much space we need to find an $s \rightarrow t$ path?

Savitch's Theorem

- Recall that given a (directed) graph $G = (V, E)$, with $|V| = n$, $|E| = m$, and $s, t \in V$, we can determine if there exists an $s \rightarrow t$ path in $O(n + m)$ time.
- What if our graph is massive, and we have limited hardware, now we instead want to focus on space bounded computation.
- Any guesses on how much space we need to find an $s \rightarrow t$ path?
- We will give a recursive algorithm for STCONN in $O(\log^2(n))$ time

Savitch's Theorem

- We will build a recursive algorithm

Savitch's Theorem

- We will build a recursive algorithm
- What should we recurse on?
- Remember the runtime doesn't need to be efficient.

Savitch's Theorem

- We will build a recursive algorithm
- What should we recurse on?
- Remember the runtime doesn't need to be efficient.
- We will divide on the length of the path.

STCONN algorithm

```
def k_edge_path(s,t,k):  
    /* Test whether a path of length at most  
       k exists from s to t */  
    if k = 0:  
        return s = t  
    if k = 1:  
        return s = t || (s,t) in edges  
    for u in V:  
        if k_edge_path(s,u,  $\lfloor k/2 \rfloor$ ) &&  
            k_edge_path (u,t,  $\lceil k/2 \rceil$ ):  
            return True  
    return False
```

- `k_edge_path(s,t,n)` solves the original problem, since any path must be of length at most n .

- $k_edge_path(s, t, n)$ solves the original problem, since any path must be of length at most n .
- Need $O(\log(n))$ space to store s, t, k .

Space Analysis

- $k_edge_path(s, t, n)$ solves the original problem, since any path must be of length at most n .
- Need $O(\log(n))$ space to store s, t, k .
- We only need to store $O(\log n)$ recursive frames at a time, each requiring $O(\log n)$.

- $k_edge_path(s, t, n)$ solves the original problem, since any path must be of length at most n .
- Need $O(\log(n))$ space to store s, t, k .
- We only need to store $O(\log n)$ recursive frames at a time, each requiring $O(\log n)$.
- Thus the space complexity is $O(\log^2 n)$.

Proof of Correctness

Proof.

We will prove this by induction on k . The base case $k = 0, 1$ is trivial. Suppose then the result holds for all $k' < k$. If there is an $s \rightarrow t$ path of length at most k then there exists a u such that there is an $s \rightarrow u$ path of length at most $\lfloor k/2 \rfloor$ and $u \rightarrow t$ path of length at most $\lceil k/2 \rceil$. By the inductive hypothesis our recursive calls are correct. Conversely, if there is no $s \rightarrow t$ path of length at most k , then there cannot exist a u which satisfies our requirements. □