Pre-lecture brain teaser

Merge Sort splits into 2 (roughly) equal sized arrays. Can we do better by splitting into more than 2 arrays? Say k arrays of size n/k each?

ECE-374-B: Lecture 10 - Divide and Conquer Algorithms

Instructor: Nickvash Kani

University of Illinois at Urbana-Champaign

Pre-lecture brain teaser

Merge Sort splits into 2 (roughly) equal sized arrays. Can we do better by splitting into more than 2 arrays? Say k arrays of size n/k each?

Quick Sort

Quick Sort

Quick Sort [Hoare]

1. Pick a pivot element from array
2. Split array into 3 subarrays: those smaller than pivot, those larger than pivot, and the pivot itself.
3. Recursively sort the subarrays, and concatenate them.

Quick Sort [Hoare]

1. Pick a pivot element from array
2. Split array into 3 subarrays: those smaller than pivot, those larger than pivot, and the pivot itself.
3. Recursively sort the subarrays, and concatenate them.

Quick Sort

Quick Sort [Hoare]

1. Pick a pivot element from array
2. Split array into 3 subarrays: those smaller than pivot, those larger than pivot, and the pivot itself. Linear scan of array does it. Time is $O(n)$
3. Recursively sort the subarrays, and concatenate them.

Quick Sort

Quick Sort [Hoare]

1. Pick a pivot element from array
2. Split array into 3 subarrays: those smaller than pivot, those larger than pivot, and the pivot itself. Linear scan of array does it. Time is $O(n)$
3. Recursively sort the subarrays, and concatenate them.

Quick Sort: Example

- example array: 16, 12, 14, 20, 5, 3, 18, 19, 1
- worst case array: 3, 7, 5, 1, 2, 4, 6, 8

See visualizer:

https://www.hackerearth.com/practice/algorithms/sorting/quick-sort/visualize/

Time Analysis

- Let k be the rank of the chosen pivot. Then, $T(n) = T(k-1) + T(n-k) + O(n)$

Time Analysis

- Let k be the rank of the chosen pivot. Then, $T(n) = T(k-1) + T(n-k) + O(n)$
- If $k = \lceil n/2 \rceil$ then $T(n) = T(\lceil n/2 \rceil - 1) + T(\lfloor n/2 \rfloor) + O(n) \leq 2T(n/2) + O(n)$.
  Then, $T(n) = O(n \log n)$.

Time Analysis

- Let k be the rank of the chosen pivot. Then, $T(n) = T(k-1) + T(n-k) + O(n)$
- If $k = \lceil n/2 \rceil$ then $T(n) = T(\lceil n/2 \rceil - 1) + T(\lfloor n/2 \rfloor) + O(n) \leq 2T(n/2) + O(n)$. Then, $T(n) = O(n \log n)$.

Time Analysis

- Let k be the rank of the chosen pivot. Then, $T(n) = T(k-1) + T(n-k) + O(n)$
- If $k = \lceil n/2 \rceil$ then $T(n) = T(\lceil n/2 \rceil - 1) + T(\lfloor n/2 \rfloor) + O(n) \leq 2T(n/2) + O(n)$. Then, $T(n) = O(n \log n)$.
- Typically, pivot is the first or last element of array. Then,

$$T(n) = \max_{1 \leq k \leq n} (T(k-1) + T(n-k) + O(n))$$

In the worst case $T(n) = T(n-1) + O(n)$, which means $T(n) = O(n^2)$. Happens if array is already sorted and pivot is always first element.

Selecting in Unsorted Lists

The Selection Problem

Big problem with QuickSort is that the pivot might not be the median.

How long would it take us to find the median of an unsorted list?

The Selection Problem

Big problem with QuickSort is that the pivot might not be the median.

How long would it take us to find the median of an unsorted list?

Sort, then $A[n/2]$. **Is this the optimal way?**

Rank of element in an array

A: an unsorted array of n integers

For $0 \leq j \leq n-1$, element of rank j is the $j+1$-th smallest element in A.

	0	1	2	3	4	5	6	7	8
Unsorted array	16	14	34	20	12	5	3	19	11
Rank	5	4	8	7	3	1	0	6	2

Problem - Selection

Input Unsorted array A of n integers **and** integer j

Goal Find the j-th smallest number in A (rank j number)

Median: $j = \lfloor n/2 \rfloor$

Problem - Selection

Input Unsorted array A of n integers **and** integer j

Goal Find the j-th smallest number in A (rank j number)

Median: $j = \lfloor n/2 \rfloor$

Simplifying assumption for sake of notation: elements of A are distinct

Algorithm I

- Sort the elements in A
- Pick jth element in sorted order

Time taken $= O(n \log n)$

Algorithm I

- Sort the elements in A
- Pick jth element in sorted order

Time taken $= O(n \log n)$

Do we need to sort? Is there an $O(n)$ time algorithm?

Algorithm II

If j is small or $n - j$ is small then

- Find j smallest/largest elements in A in $O(jn)$ time. (How?)
- Time to find median is $O(n^2)$.

Quick select

QuickSelect

- Pick a pivot element a from A
- Partition A based on a.
  $A_{\text{less}} = \{x \in A \mid x \leq a\}$ and $A_{\text{greater}} = \{x \in A \mid x > a\}$
- $|A_{\text{less}}| = j$: return a
- $|A_{\text{less}}| > j$: recursively find jth smallest element in A_{less}
- $|A_{\text{less}}| < j$: recursively find kth smallest element in A_{greater} where $k = j - |A_{\text{less}}|$.

Example

16	14	34	20	12	5	3	19	11

Time Analysis

- Partitioning step: $O(n)$ time to scan A
- How do we choose pivot? Recursive running time?

Time Analysis

- Partitioning step: $O(n)$ time to scan A
- How do we choose pivot? Recursive running time?

Suppose we always choose pivot to be $A[1]$.

Time Analysis

- Partitioning step: $O(n)$ time to scan A
- How do we choose pivot? Recursive running time?

Suppose we always choose pivot to be $A[1]$.

Say A is sorted in increasing order and $j = n$.
How long does this new algorithm take?

Does this help with QuickSort?

Should we combine this with QuickSort

Does this help with QuickSort?

Should we combine this with QuickSort

Of course not! It takes $O(n^2)$ which is already the worse case of QuickSort. Need another method....

Does this help with QuickSort?

Looking at the quicksort recurrence again:

$$T(n) = T(k-1) + T(n-k) + O(n)$$

Does k *need* to be $n/2$?

Does this help with QuickSort?

Looking at the quicksort recurrence again:

$$T(n) = T(k-1) + T(n-k) + O(n)$$

Does k *need* to be $n/2$?

What if $k = \frac{3}{5}n$?

Does this help with QuickSort?

Looking at the quicksort recurrence again:

$$T(n) = T(k-1) + T(n-k) + O(n)$$

Does k *need* to be $n/2$?

What if $k = \frac{3}{5}n$?

What if $k = \frac{7}{10}n$?

Does this help with QuickSort?

Looking at the quicksort recurrence again:

$$T(n) = T(k-1) + T(n-k) + O(n)$$

Does k *need* to be $n/2$?

What if $k = \frac{3}{5}n$?

What if $k = \frac{7}{10}n$?

we only need to be able to find a rough median! .... How do we do that?

Median of Medians

Divide and Conquer Approach

Idea

- Break input A into many subarrays: $L_1, \dots L_k$.
- Find median m_i in each subarray L_i.
- Find the median x of the medians $m_1, \dots, m_k$.
- Intuition: The median x should be close to being a good median of all the numbers in A.
- Use x as pivot in previous algorithm.

Deterministic selection - example

Given an array $A = [0, ..., n-1]$ of n numbers and an index i, where $0 \leq i \leq n-1$, find the i^{th} smallest element of A.

For instance, assume $n = 20$ and $i = 10$.

3	2	14	6	0	16	8	9	13	12	7	17	10	1	11	15	5	18	4	19

The smallest element of rank 10 would be 10. But how do we figure that out?

Do median of medians.....

Call Median-of-Medians(A, 10)

Deterministic selection - example

Given an array $A = [0, ..., n-1]$ of n numbers and an index i, where $0 \leq i \leq n-1$, find the i^{th} smallest element of A.

For instance, assume $n = 20$ and $i = 10$.

3	2	14	6	0	16	8	9	13	12	7	17	10	1	11	15	5	18	4	19

The smallest element of rank 10 would be 10. But how do we figure that out?

Do median of medians.....

Call Median-of-Medians(A, 10)

First thing we need to do is find the pivot!

Deterministic selection - example

Given an array $A = [0, ..., n-1]$ of n numbers and an index i, where $0 \leq i \leq n-1$, find the i^{th} smallest element of A.

For instance, assume $n = 20$ and $i = 10$.

3	2	14	6	0	16	8	9	13	12	7	17	10	1	11	15	5	18	4	19

The smallest element of rank 10 would be 10. But how do we figure that out?

Do median of medians.....

Call Median-of-Medians(A, 10)

First thing we need to do is find the pivot!

Deterministic selection - example

First we reorganize:

3	16	7	15
2	8	17	5
14	0	10	18
6	13	1	4
0	12	11	19

Deterministic selection - example

First we reorganize:

3	16	7	15
2	8	17	5
14	0	10	18
6	13	1	4
0	12	11	19

Then we sort each column:

0	8	1	4
2	9	7	5
3	12	10	15
6	13	11	18
14	16	17	19

Deterministic selection - example

First we reorganize:

3	16	7	15
2	8	17	5
14	0	10	18
6	13	1	4
0	12	11	19

Then we sort each column:

0	8	1	4
2	9	7	5
3	12	10	15
6	13	11	18
14	16	17	19

Still need the pivot. Find median of medians

Deterministic selection - example

0	8	1	4
2	9	7	5
3	12	10	15
6	13	11	18
14	16	17	19

Deterministic selection - example

0	8	1	4
2	9	7	5
3	12	10	15
6	13	11	18
14	16	17	19

- Call Median-of-Medians([3,12,10,15], floor(len/2) = 2)
- Can sort this in linear time.
- Get back 12.
- **12** is our new pivot!

Deterministic selection - example

Back to our original array! Use the pivot (=12) to break it up into two.

3	2	14	6	0	16	8	9	13	12	7	17	10	1	11	15	5	18	4	19

3	2	6	0	8	9	7	10	1	11	5	4

12

14	16	13	17	15	18	19

We know the following:

- $\text{len}(A_{Lower}) = 12$
- $\text{len}(A_{Upper}) = 7$
- Want $k = 10$

Deterministic selection - example

Back to our original array! Use the pivot (=12) to break it up into two.

3	2	14	6	0	16	8	9	13	12	7	17	10	1	11	15	5	18	4	19

3	2	6	0	8	9	7	10	1	11	5	4

12

14	16	13	17	15	18	19

We know the following:

- $\text{len}(A_{Lower}) = 12$
- $\text{len}(A_{Upper}) = 7$
- Want $k = 10$

Call Median-of-Medians$(A_{Lower}, 10)$

Deterministic selection - example

Then we do this again:

3	2	6	0	8	9	7	10	1	11	5	4

Deterministic selection - example

Then we do this again:

3	2	6	0	8	9	7	10	1	11	5	4

First we reorganize:

3	9	
2	7	5
6	10	4
0	1	
8	11	

Deterministic selection - example

Then we do this again:

3	2	6	0	8	9	7	10	1	11	5	4

First we reorganize:

3	9	
2	7	5
6	10	4
0	1	
8	11	

Then we sort each column:

0	1	
2	7	4
3	9	5
6	10	
8	11	

Deterministic selection - example

0	1	
2	7	4
3	9	5
6	10	
8	11	

Deterministic selection - example

0	1	
2	7	4
3	9	5
6	10	
8	11	

- Call Median-of-Medians([3,9,5], floor(n/2) = 1)
- Can sort this in linear time.
- Get back 5.
- **5** is our new pivot!

Deterministic selection - example

Back to our original array! Use the pivot (=5) to break it up into two (well three).

3	2	6	0	8	9	7	10	1	11	5	4

3	2	0	1	4

5

6	8	9	7	10	11

We know the following:

- $\text{len}(A_{Lower}) = 5$
- $\text{len}(A_{Upper}) = 6$
- Want $k = 10$ (pivot is of rank 5)

Deterministic selection - example

Back to our original array! Use the pivot (=5) to break it up into two (well three).

3	2	6	0	8	9	7	10	1	11	5	4

3	2	0	1	4

5

6	8	9	7	10	11

We know the following:

- $\text{len}(A_{Lower}) = 5$
- $\text{len}(A_{Upper}) = 6$
- Want $k = 10$ (pivot is of rank 5)

Call Median-of-Medians$(A_{Upper}, 10 - (5 + 1) = 4)$

Deterministic selection - example

Then we do this again:

6	8	9	7	10	11

Deterministic selection - example

Then we do this again:

6	8	9	7	10	11

First we reorganize:

6	
8	
9	11
7	
10	

Deterministic selection - example

Then we do this again:

6	8	9	7	10	11

First we reorganize:

6	
8	
9	11
7	
10	

Then we sort each column:

6	
7	
8	11
9	
10	

Deterministic selection - example

6	
7	
8	11
9	
10	

Deterministic selection - example

6	
7	
8	11
9	
10	

- Call Median-of-Medians([8,11], floor(len/2) = 1)
- Can sort this in linear time.
- Get back 11.
- **11** is our new pivot!

Deterministic selection - example

Back to our original array! Use the pivot (=11) to break it up into partitions.

6	8	9	7	10	11

6	8	9	7	10

11

We know the following:

- len(A_{Lower}) $= 5$
- len(A_{Upper}) $= 0$
- Want $k = 4$ (pivot is of rank 5)

Deterministic selection - example

Back to our original array! Use the pivot (=11) to break it up into partitions.

6	8	9	7	10	11

6	8	9	7	10

11

We know the following:

- $\text{len}(A_{Lower}) = 5$
- $\text{len}(A_{Upper}) = 0$
- Want $k = 4$ (pivot is of rank 5)

Deterministic selection - example

Final Step!

6	8	9	7	10

Can sort in linear time!

6	7	8	9	10

Return $Sorted(A[k] = A[4]) = 10$

Algorithm for Selection

```
select(A, j):
    Form lists L_1, L_2, ..., L_⌈n/5⌉ where L_i = {A[5i − 4], ..., A[5i]}
    Find median b_i of each L_i using brute-force
    Find median b of B = {b_1, b_2, ..., b_⌈n/5⌉}
    Partition A into A_less and A_greater using b as pivot
    if (|A_less|) = j return b
    else if (|A_less|) > j)
        return select(A_less, j)
    else
        return select(A_greater, j − |A_less|)
```

Algorithm for Selection

```
select(A, j):
    Form lists L_1, L_2, ..., L_⌈n/5⌉ where L_i = {A[5i − 4], ..., A[5i]}
    Find median b_i of each L_i using brute-force
    Find median b of B = {b_1, b_2, ..., b_⌈n/5⌉}
    Partition A into A_less and A_greater using b as pivot
    if (|A_less|) = j return b
    else if (|A_less|) > j)
        return select(A_less, j)
    else
        return select(A_greater, j − |A_less|)
```

How do we find median of B?

Algorithm for Selection

```
select(A, j):
    Form lists L_1, L_2, ..., L_{⌈n/5⌉} where L_i = {A[5i − 4], ..., A[5i]}
    Find median b_i of each L_i using brute-force
    Find median b of B = {b_1, b_2, ..., b_{⌈n/5⌉}}
    Partition A into A_less and A_greater using b as pivot
    if (|A_less|) = j return b
    else if (|A_less|) > j)
        return select(A_less, j)
    else
        return select(A_greater, j − |A_less|)
```

How do we find median of B? Recursively!

Running time of deterministic median selection

Running time of deterministic median selection

$$T(n) \leq T(\lceil n/5 \rceil) + \max\{T(|A_{\text{less}}|), T(|A_{\text{greater}})|\} + O(n)$$

Running time of deterministic median selection

$$T(n) \leq T(\lceil n/5 \rceil) + \max\{T(|A_{\text{less}}|), T(|A_{\text{greater}})|\} + O(n)$$

From Lemma,

$$T(n) \leq T(\lceil n/5 \rceil) + T(\lfloor 7n/10 \rfloor) + O(n)$$

and

$$T(n) = O(1) \qquad n < 10$$

Running time of deterministic median selection

$$T(n) \leq T(\lceil n/5 \rceil) + \max\{T(|A_{\text{less}}|), T(|A_{\text{greater}})|\} + O(n)$$

From Lemma,

$$T(n) \leq T(\lceil n/5 \rceil) + T(\lfloor 7n/10 \rfloor) + O(n)$$

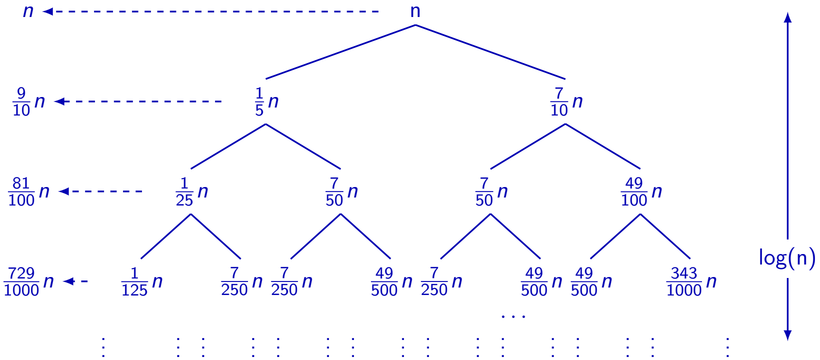
and

$$T(n) = O(1) \qquad n < 10$$

Exercise: show that $T(n) = O(n)$

Recursion tree fill-in

If the workload is decreasing at every level, then total work is dominated by the root.



$$T(n) \leq T(\lceil n/5 \rceil) + T(\lfloor 7n/10 \rfloor) + O(n) = O(n)$$

What about QuickSort?

How would we use the median of medians approach for quicksort?

What about QuickSort?

How would we use the median of medians approach for quicksort?

Just use MoM if find pivot!

- Original recurrence: $T(n) = T(k-1) + T(n-k) + O(n)$
- With MoM: $T(n) = T(\frac{3}{10}n) + T(\frac{7}{10}n) + O(n) + O(n)$

Median of Medians Algorithm

Due to:M. Blum, R. Floyd, D. Knuth, V. Pratt, R. Rivest, and R. Tarjan.

“Time bounds for selection”.
Journal of Computer System Sciences (JCSS), 1973.

Median of Medians Algorithm

Due to:M. Blum, R. Floyd, D. Knuth, V. Pratt, R. Rivest, and R. Tarjan.

"Time bounds for selection".
Journal of Computer System Sciences (JCSS), 1973.

How many Turing Award winners in the author list?

Median of Medians Algorithm

Due to:M. Blum, R. Floyd, D. Knuth, V. Pratt, R. Rivest, and R. Tarjan.

"Time bounds for selection".
Journal of Computer System Sciences (JCSS), 1973.

How many Turing Award winners in the author list?

All except Vaughan Pratt!

Median of Medians Algorithm

Due to:M. Blum, R. Floyd, D. Knuth, V. Pratt, R. Rivest, and R. Tarjan.

"Time bounds for selection".
Journal of Computer System Sciences (JCSS), 1973.

How many Turing Award winners in the author list?

All except Vaughan Pratt! **Favorite Knuth quote**: He once warned a correspondent, "Beware of bugs in the above code; I have only proved it correct, not tried it."

Takeaway Points

- Recursion tree method and guess and verify are the most reliable methods to analyze recursions in algorithms.
- Recursive algorithms naturally lead to recurrences.
- Some times one can look for certain type of recursive algorithms (reverse engineering) by understanding recurrences and their behavior.

Problem statement: Multiplying numbers + a slow algorithm

The Problem: Multiplying numbers

Given two large positive integer numbers b and c, with n digits, compute the number $b * c$.

Egyptian multiplication: 1850BC (3870 years ago?)

76	35	

Egyptian multiplication: 1850BC (3870 years ago?)

76	35	
76	$34 + 1$	76

Egyptian multiplication: 1850BC (3870 years ago?)

76	35	
76	$34 + 1$	76
76	34	

Egyptian multiplication: 1850BC (3870 years ago?)

76	35	
76	$34 + 1$	76
76	34	
152	17	

Egyptian multiplication: 1850BC (3870 years ago?)

76	35	
76	$34 + 1$	76
76	34	
152	17	
152	$16 + 1$	152

Egyptian multiplication: 1850BC (3870 years ago?)

76	35	
76	$34 + 1$	76
76	34	
152	17	
152	$16 + 1$	152
152	16	

Egyptian multiplication: 1850BC (3870 years ago?)

76	35	
76	$34 + 1$	76
76	34	
152	17	
152	$16 + 1$	152
152	16	
304	8	

Egyptian multiplication: 1850BC (3870 years ago?)

76	35	
76	$34 + 1$	76
76	34	
152	17	
152	$16 + 1$	152
152	16	
304	8	
608	4	

Egyptian multiplication: 1850BC (3870 years ago?)

76	35	
76	$34 + 1$	76
76	34	
152	17	
152	$16 + 1$	152
152	16	
304	8	
608	4	
1216	2	

Egyptian multiplication: 1850BC (3870 years ago?)

76	35	
76	$34 + 1$	76
76	34	
152	17	
152	$16 + 1$	152
152	16	
304	8	
608	4	
1216	2	
2432	1	2432

Egyptian multiplication: 1850BC (3870 years ago?)

76	35	
76	$34 + 1$	76
76	34	
152	17	
152	$16 + 1$	152
152	16	
304	8	
608	4	
1216	2	
2432	1	2432
		2660

The problem: Multiplying Numbers

Problem Given two n-digit numbers x and y, compute their product.

Grade School Multiplication
Compute "partial product" by multiplying each digit of y with x and adding the partial products.

$$
\begin{array}{r}
3141 \\
\underline{\times 2718} \\
25128 \\
3141 \\
21987 \\
\underline{6282} \\
8537238
\end{array}
$$

Time Analysis of Grade School Multiplication

- Each partial product: $\Theta(n)$
- Number of partial products: $\Theta(n)$
- Addition of partial products: $\Theta(n^2)$
- Total time: $\Theta(n^2)$

Multiplication using Divide and Conquer

Divide and Conquer

Assume n is a power of 2 for simplicity and numbers are in decimal.

Split each number into two numbers with equal number of digits

- $b = b_{n-1}b_{n-2}\ldots b_0$ and $c = c_{n-1}c_{n-2}\ldots c_0$
- $b = b_{n-1}\ldots b_{n/2}0\ldots 0 + b_{n/2-1}\ldots b_0$
- $b(x) = b_L x + b_R$, where $x = 10^{n/2}$, $b_L = b_{n-1}\ldots b_{n/2}$ and $b_R = b_{n/2-1}\ldots b_0$
- Similarly $c(x) = c_L x + c_R$ where $c_L = c_{n-1}\ldots c_{n/2}$ and $c_R = c_{n/2-1}\ldots c_0$

Example

$$1234 \times 5678 = (12x + 34) \times (56x + 78) \qquad \text{for} \quad x = 100.$$
$$= 12 \cdot 56 \cdot x^2 + (12 \cdot 78 + 34 \cdot 56)x + 34 \cdot 78.$$

$$\begin{aligned} 1234 \times 5678 \quad &= \quad (100 \times 12 + 34) \times (100 \times 56 + 78) \\ &= \quad 10000 \times 12 \times 56 \\ & \qquad +100 \times (12 \times 78 + 34 \times 56) \\ & \qquad +34 \times 78 \end{aligned}$$

Divide and Conquer for multiplication

Assume n is a power of 2 for simplicity and numbers are in decimal.

- $b = b_{n-1}b_{n-2}\ldots b_0$ and $c = c_{n-1}c_{n-2}\ldots c_0$
- $b \equiv b(x) = b_L x + b_R$
  where $x = 10^{n/2}$, $b_L = b_{n-1}\ldots b_{n/2}$ and $b_R = b_{n/2-1}\ldots b_0$
- $c \equiv c(x) = c_L x + c_R$ where $c_L = c_{n-1}\ldots c_{n/2}$ and $c_R = c_{n/2-1}\ldots c_0$

Divide and Conquer for multiplication

Assume n is a power of 2 for simplicity and numbers are in decimal.

- $b = b_{n-1}b_{n-2}\dots b_0$ and $c = c_{n-1}c_{n-2}\dots c_0$
- $b \equiv b(x) = b_L x + b_R$
  where $x = 10^{n/2}$, $b_L = b_{n-1}\dots b_{n/2}$ and $b_R = b_{n/2-1}\dots b_0$
- $c \equiv c(x) = c_L x + c_R$ where $c_L = c_{n-1}\dots c_{n/2}$ and $c_R = c_{n/2-1}\dots c_0$

Therefore, for $x = 10^{n/2}$, we have

$$
\begin{aligned}
bc = b(x)c(x) &= (b_L x + b_R)(c_L x + c_R) \\
&= b_L c_L x^2 + (b_L c_R + b_R c_L)x + b_R c_R \\
&= 10^n b_L c_L + 10^{n/2}(b_L c_R + b_R c_L) + b_R c_R
\end{aligned}
$$

Time Analysis

$$bc = 10^n b_L c_L + 10^{n/2}(b_L c_R + b_R c_L) + b_R c_R$$

4 recursive multiplications of number of size $n/2$ each plus 4 additions and left shifts (adding enough 0's to the right)

Time Analysis

$$bc = 10^n b_L c_L + 10^{n/2}(b_L c_R + b_R c_L) + b_R c_R$$

4 recursive multiplications of number of size $n/2$ each plus 4 additions and left shifts (adding enough 0's to the right)

$$T(n) = 4T(n/2) + O(n) \qquad T(1) = O(1)$$

Time Analysis

$$bc = 10^n b_L c_L + 10^{n/2}(b_L c_R + b_R c_L) + b_R c_R$$

4 recursive multiplications of number of size $n/2$ each plus 4 additions and left shifts (adding enough 0's to the right)

$$T(n) = 4T(n/2) + O(n) \qquad T(1) = O(1)$$

$T(n) = \Theta(n^2)$. No better than grade school multiplication!

Faster multiplication: Karatsuba's Algorithm

A Trick of Gauss

Carl Friedrich Gauss: 1777–1855 "Prince of Mathematicians"

Observation: Multiply two complex numbers: $(a + bi)$ and $(c + di)$

$$(a + bi)(c + di) = ac - bd + (ad + bc)i$$

A Trick of Gauss

Carl Friedrich Gauss: 1777–1855 "Prince of Mathematicians"

Observation: Multiply two complex numbers: $(a + bi)$ and $(c + di)$

$$(a + bi)(c + di) = ac - bd + (ad + bc)i$$

How many multiplications do we need?

A Trick of Gauss

Carl Friedrich Gauss: 1777–1855 "Prince of Mathematicians"

Observation: Multiply two complex numbers: $(a + bi)$ and $(c + di)$

$$(a + bi)(c + di) = ac - bd + (ad + bc)i$$

How many multiplications do we need?

Only 3! If we do extra additions and subtractions.
Compute $ac, bd, (a + b)(c + d)$. Then

Gauss technique for polynomials

$p(x) = ax + b$ and $q(x) = cx + d$.

$$p(x)q(x) = acx^2 + (ad + bc)x + bd.$$

Gauss technique for polynomials

$$p(x) = ax + b \qquad \text{and} \qquad q(x) = cx + d.$$

$$p(x)q(x) = acx^2 + (ad + bc)x + bd.$$

$$p(x)q(x) = acx^2 + \big((a + b)(c + d) - ac - bd\big)x + bd.$$

Improving the Running Time

$$bc = b(x)c(x) = (b_L x + b_R)(c_L x + c_R)$$

Improving the Running Time

$$
\begin{aligned}
bc = b(x)c(x) &= (b_L x + b_R)(c_L x + c_R) \\
&= b_L c_L x^2 + (b_L c_R + b_R c_L)x + b_R c_R
\end{aligned}
$$

Improving the Running Time

$$\begin{aligned} bc &= b(x)c(x) = (b_L x + b_R)(c_L x + c_R) \\ &= b_L c_L x^2 + (b_L c_R + b_R c_L)x + b_R c_R \\ &= (b_L * c_L)x^2 + \Big((b_L + b_R) * (c_L + c_R) - b_L * c_L - b_R * c_R\Big)x \\ &\qquad + b_R * c_R \end{aligned}$$

Improving the Running Time

$$\begin{aligned} bc &= b(x)c(x) = (b_L x + b_R)(c_L x + c_R) \\ &= b_L c_L x^2 + (b_L c_R + b_R c_L)x + b_R c_R \\ &= (b_L * c_L)x^2 + \Big((b_L + b_R) * (c_L + c_R) - b_L * c_L - b_R * c_R\Big)x \\ &\qquad + b_R * c_R \end{aligned}$$

Recursively compute only $b_L c_L, b_R c_R, (b_L + b_R)(c_L + c_R)$.

Improving the Running Time

$$\begin{aligned} bc &= b(x)c(x) = (b_L x + b_R)(c_L x + c_R) \\ &= b_L c_L x^2 + (b_L c_R + b_R c_L)x + b_R c_R \\ &= (b_L * c_L)x^2 + \Big((b_L + b_R) * (c_L + c_R) - b_L * c_L - b_R * c_R\Big)x \\ &\qquad + b_R * c_R \end{aligned}$$

Recursively compute only $b_L c_L, b_R c_R, (b_L + b_R)(c_L + c_R)$.

Time Analysis
Running time is given by

$$T(n) = 3T(n/2) + O(n) \qquad\qquad T(1) = O(1)$$

which means $T(n) = O(n^{\log_2 3}) = O(n^{1.585})$

State of the Art

Schönhage-Strassen 1971: $O(n \log n \log \log n)$ time using Fast-Fourier-Transform (FFT)

Martin Fürer 2007: $O(n \log n 2^{O(\log^* n)})$ time

Conjecture: There is an $O(n \log n)$ time algorithm.