



## 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 11 - Divide and Conquer Algorithms

---

**Instructor:** Abhishek Kumar Umrawal

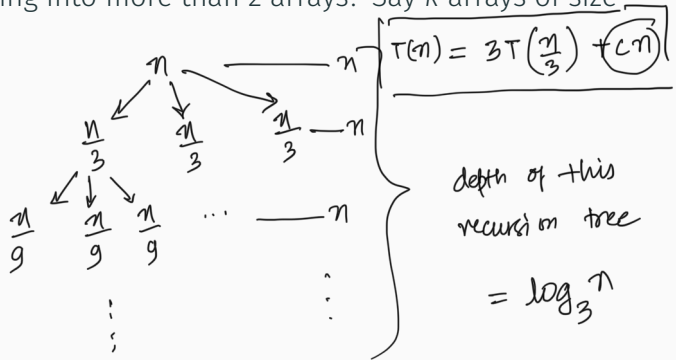
September 28, 2023

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?

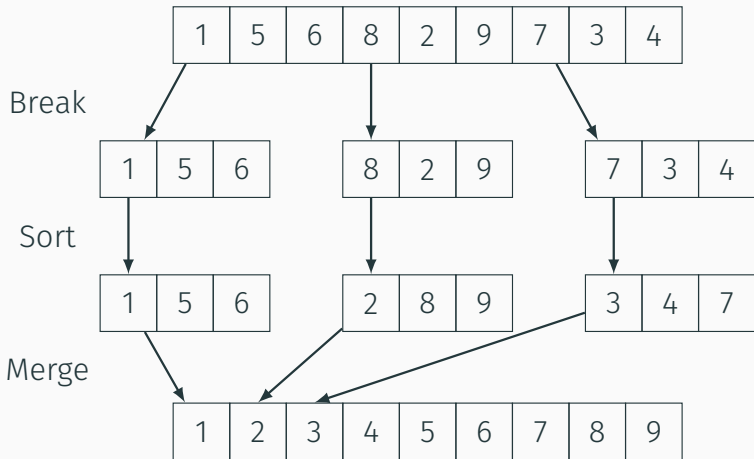
Eg.  $k=3$



$$k=3: O(n \log_3 n)$$

# Pre-lecture brain teaser

Simpler case: Break into 3 lists:



## 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?

What does the recurrence for  $k = 3$  look like?

Recurrence for  $k=3$ :

$$T(n) = 3T\left(\frac{n}{3}\right) + O(n)$$

Last slide  $\rightarrow$

$$T_3(n) = O(n \log_3 n)$$

$k=2$ :

$$T_2(n) = O(n \log_2 n)$$

Relationship:

$$\log_3 n = \underbrace{\log_3 2}_{\text{constant}} \log_2 n \Rightarrow$$

(same!)

## 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?

What does the recurrence for  $k = 3$  look like?

$$T(n) = 3T\left(\frac{n}{3}\right) + cn$$

What is the solution to this recurrence?

## 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?

What does the recurrence for  $k = 3$  look like?

$$T(n) = 3T\left(\frac{n}{3}\right) + cn$$

What is the solution to this recurrence?

$$T(n) = 3T\left(\frac{n}{3}\right) + cn = O(n \log n)$$



## 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?

What does the recurrence for more general  $k$  look like?

## 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?

What does the recurrence for more general  $k$  look like?

$$T(n) = kT\left(\frac{n}{k}\right) + cn$$

What is the solution to this recurrence?

$$T(n) = kT\left(\frac{n}{k}\right) + cn = O(\underbrace{n \log_k n})$$

## 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?

What does the recurrence for more general  $k$  look like?

$$T(n) = kT\left(\frac{n}{k}\right) + cn$$

What is the solution to this recurrence?

$$T(n) = kT\left(\frac{n}{k}\right) + cn = O(n \log n)$$

So why don't we use smaller lists?

# Learning Objectives

---

# Learning Objectives

At the end of the lecture, you should be able to understand

- the idea of divide and conquer and how recursion forms a basis of it,
- the quicksort algorithm and its runtime analysis,
- the selection problem, quickselect algorithm and its runtime analysis, and
- the multiplication of numbers problem, a simple divide and conquer algorithm, and Karatsuba's algorithm, and runtime analysis of these algorithms.

# Quick Sort

---

# Quick Sort

Quick Sort [Hoare] Tony Hoare (1959-60), A British Mathematician,

Turing Award, FRS  
FREng

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.  $\leftarrow O(n)$
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.
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

- array: 16, 12, 14, 20, 5, 3, 18, 19, 1
- pivot: 16

See visualizer:

[hackerearth.com/practice/algorithms/sorting/quick-sort/visualize](https://hackerearth.com/practice/algorithms/sorting/quick-sort/visualize)

# Time Analysis

- Let  $k$  be the rank of the chosen pivot. Then,

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

# Time Analysis

- Let  $k$  be the rank of the chosen pivot. Then,

$$T(n) = T(\overbrace{k} - 1) + T(n - \underline{k}) + O(n)$$

- If  $\underline{k} = \lceil n/2 \rceil$  then

$$\underline{T(n)} = T(\lceil n/2 \rceil - 1) + T(\lfloor n/2 \rfloor) + O(n) \leq \underline{2T(n/2) + O(n)}.$$

Then,  $T(n) = O(n \log n)$ .

$$T(n) \leq 2T(n/2) + O(n)$$

$$T(n) = O(n \log n)$$

# 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,

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

*(Handwritten arrows point from the '1' in the max subscript to the '1' in the first T term, and from the 'n' in the max subscript to the 'n' in the second T term.)*

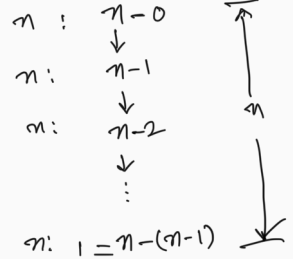
In the worst case  $T(n) = T(n - 1) + O(n)$ , which means  $T(n) = O(n^2)$ . Happens if array is already sorted and pivot is always first element.

$k=1 \mid n: T(n) = T(0) + T(n-1) + O(n)$

How to  
Solve?

$$T(n) = T(n-1) + \underline{O(n)} \rightarrow \text{Tree:}$$

$$T(n) = O(n \cdot n) \Leftarrow \\ = O(n^2)$$



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

<u>Unsorted array</u>	16	14	34	20	12	5	3	19	11
<u>Ranks</u>	6	5	9	8	4	2	1	7	3
<u>Sort of array</u> sorted	3	5	11	12	14	16	19	20	34

## 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:**  $j = \lfloor (n+1)/2 \rfloor$

E.g.  $9 \ 5 \ 1 \ 7 \ 2 \quad n=5$

$1 \ 2 \ \underline{\underline{5}} \ 7 \ 9$

$$\frac{5+1}{2} = \lfloor 3 \rfloor = 3$$

## 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:  $j = \lfloor (n + 1)/2 \rfloor$

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

# Algorithm I

- Sort the elements in A
- Pick  $j$ th element in sorted order

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

# Algorithm I

- Sort the elements in  $A$
- Pick  $j$ th 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  $O(n^2)$ .

$j$ th ranked element: time:  $O(jn)$

E.g. 1st ranked " : time:  $O(1n)$

$$\text{median: } j = \left\lfloor \frac{n+1}{2} \right\rfloor = O\left(\frac{n}{2} \cdot n\right) = O(n^2)$$



## Quick select

---

# QuickSelect

- Pick a pivot element  $a$  from  $A$

- Partition  $A$  based on  $a$ .

$$\underline{A_{\text{less}} = \{x \in A \mid x \leq a\}} \text{ and } \underline{A_{\text{greater}} = \{x \in A \mid x > a\}}$$

- $|A_{\text{less}}| = j$ : return  $a$

Card./  
length

- $|A_{\text{less}}| > j$ : recursively find  $j$ th smallest element in  $A_{\text{less}}$

- $|A_{\text{less}}| < j$ : recursively find  $k$ th smallest element in  $A_{\text{greater}}$   
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?

$$O(n^2)$$

## 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:

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

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

when  $k = n/2$  :  $T(n) = O(n \log n)$

## 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  $k = \frac{3}{5}n$ ?

## 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

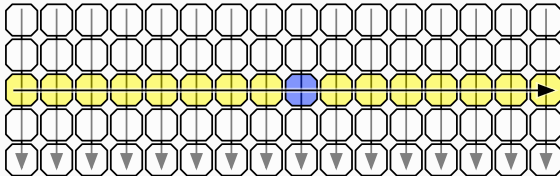
- Break input  $A$  into many subarrays:  $L_1, \dots, L_k$ .
- Find median  $m_i$  in each subarray  $L_i$ .
- Find the median  $x$  of the medians  $m_1, \dots, m_k$ .
- Intuition: The median  $x$  should be close to being a good median of all the numbers in  $A$ .
- Use  $x$  as pivot in previous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

## Example

11	7	3	42	174	310	1	92	87	12	19	15
----	---	---	----	-----	-----	---	----	----	----	----	----

## Example

11	7	3	42	174	310	1	92	87	12	19	15
----	---	---	----	-----	-----	---	----	----	----	----	----





## Choosing the pivot

- Partition array  $A$  into  $\lceil n/5 \rceil$  lists of 5 items each.

$$L_1 = \{A[1], A[2], \dots, A[5]\}, L_2 = \{A[6], \dots, A[10]\}, \dots,$$

$$L_i = \{A[5i + 1], \dots, A[5i + 5]\}, \dots,$$

$$L_{\lceil n/5 \rceil} = \{A[5\lceil n/5 \rceil - 4], \dots, A[n]\}.$$

- For each  $i$  find median  $b_i$  of  $L_i$  using brute-force in  $O(1)$  time. Total  $O(n)$  time  $\rightarrow O\left(\lceil \frac{n}{5} \rceil\right) = O(n)$
- Let  $B = \{b_1, b_2, \dots, b_{\lceil n/5 \rceil}\}$
- Find median  $b$  of  $B$

## Choosing the pivot

- Partition array  $A$  into  $\lceil n/5 \rceil$  lists of 5 items each.  
 $L_1 = \{A[1], A[2], \dots, A[5]\}, L_2 = \{A[6], \dots, A[10]\}, \dots,$   
 $L_i = \{A[5i + 1], \dots, A[5i + 5]\}, \dots,$   
 $L_{\lceil n/5 \rceil} = \{A[5\lceil n/5 \rceil - 4], \dots, A[n]\}.$
- For each  $i$  find median  $b_i$  of  $L_i$  using brute-force in  $O(1)$  time. Total  $O(n)$  time
- Let  $B = \{b_1, b_2, \dots, b_{\lceil n/5 \rceil}\}$
- Find median  $b$  of  $B$

Median of  $B$  is an approximate median of  $A$ . That is, if  $b$  is used as a pivot to partition  $A$ , then  $|A_{\text{less}}| \leq 7n/10$  and  $|A_{\text{greater}}| \leq 7n/10$ .

Lemma:

# Algorithm for Selection

**select**( $A, j$ ):

Form lists  $L_1, L_2, \dots, L_{\lceil n/5 \rceil}$  where  $L_i = \{A[5i-4], \dots, A[5i]\}$

Find median  $b_i$  of each  $L_i$  using brute-force

Find median  $b$  of  $B = \{b_1, b_2, \dots, b_{\lceil n/5 \rceil}\}$

Partition  $A$  into  $A_{\text{less}}$  and  $A_{\text{greater}}$  using  $b$  as pivot

if  $(|A_{\text{less}}|) = j$  return  $b$

else if  $(|A_{\text{less}}|) > j$

return **select**( $A_{\text{less}}, j$ )

else

return **select**( $A_{\text{greater}}, j - |A_{\text{less}}|$ )

# Algorithm for Selection

**select**( $A, j$ ):

Form lists  $L_1, L_2, \dots, L_{\lceil n/5 \rceil}$  where  $L_i = \{A[5i-4], \dots, A[5i]\}$

Find median  $b_i$  of each  $L_i$  using brute-force

Find median  $b$  of  $B = \{b_1, b_2, \dots, b_{\lceil n/5 \rceil}\}$

Partition  $A$  into  $A_{\text{less}}$  and  $A_{\text{greater}}$  using  $b$  as pivot

if  $(|A_{\text{less}}|) = j$  return  $b$

else if  $(|A_{\text{less}}|) > j$

return **select**( $A_{\text{less}}, j$ )

else

return **select**( $A_{\text{greater}}, j - |A_{\text{less}}|$ )

How do we find median of  $B$ ?

# Algorithm for Selection

**select**( $A, j$ ):

Form lists  $L_1, L_2, \dots, L_{\lceil n/5 \rceil}$  where  $L_i = \{A[5i-4], \dots, A[5i]\}$

Find median  $b_i$  of each  $L_i$  using brute-force

Find median  $b$  of  $B = \{b_1, b_2, \dots, b_{\lceil n/5 \rceil}\}$

Partition  $A$  into  $A_{\text{less}}$  and  $A_{\text{greater}}$  using  $b$  as pivot

if  $(|A_{\text{less}}|) = j$  return  $b$

else if  $(|A_{\text{less}}|) > j$

return **select**( $A_{\text{less}}, j$ )

else

return **select**( $A_{\text{greater}}, j - |A_{\text{less}}|$ )

How do we find median of  $B$ ? Recursively!

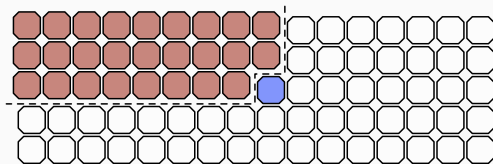
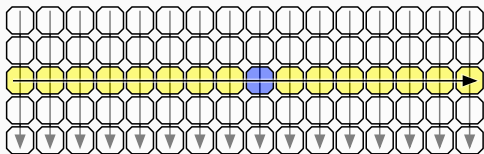
---

Median of medians is a good median

---

## Median of Medians: Proof of Lemma

There are at least  $3n/10$  elements smaller than the median of medians  $b$ .



## Median of Medians: Proof of Lemma

There are at least  $3n/10$  elements smaller than the median of medians  $b$ .

At least half of the  $\lfloor n/5 \rfloor$  groups have at least 3 elements smaller than  $b$ , except for the group containing  $b$  which has 2 elements smaller than  $b$ . Hence number of elements smaller than  $b$  is:

$$3 \lfloor \frac{\lfloor n/5 \rfloor + 1}{2} \rfloor - 1 \geq \underline{3n/10}$$

Induction/

Sketch



## Median of Medians: Proof of Lemma

There are at least  $3n/10$  elements smaller than the median of medians  $b$ .

$$|A_{\text{greater}}| \leq 7n/10.$$

Via symmetric argument,

$$|A_{\text{less}}| \leq 7n/10.$$

## Running time of deterministic median selection

---

## Running time of deterministic median selection


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

$$T(n) \leq T\left(\lceil \frac{n}{5} \rceil\right) + T\left(\frac{7}{10}n\right) + O(n)$$

## 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

$$\underline{T(n) = O(1)} \quad \underline{n < 10}$$

## 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,

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

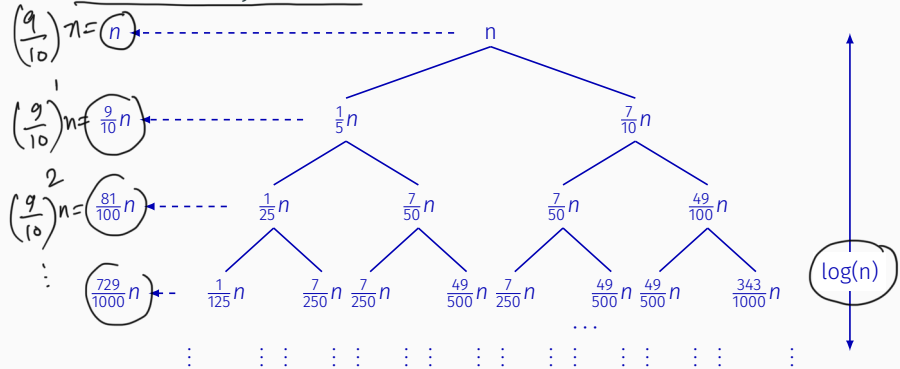
and

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

**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?

$$T(n) = \left(\frac{9}{10}\right)^0 n + \left(\frac{9}{10}\right)^1 n + \dots + \left(\frac{9}{10}\right)^{\log n} n$$

→ Geometric series

$$T(n) = O(n)$$

# 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?)

$$76 \mid 35 \mid$$



## Egyptian multiplication: 1850BC (3870 years ago?)

$$\begin{array}{r|l|l} 76 & 35 & \\ 76 & 34 + 1 & 76 \end{array}$$

## Egyptian multiplication: 1850BC (3870 years ago?)

$$\begin{array}{r|l|l} 76 & 35 & \\ 76 & 34 + 1 & 76 \\ 76 & 34 & \end{array}$$

## 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.

Handwritten multiplication of 3141 by 2718. The calculation is shown in two parts. On the left, the standard vertical multiplication is shown with the final product circled. On the right, the same calculation is shown with a vertical line separating the partial products from the final sum.

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

The partial products are circled in the original image: 25128, 3141, 21987, and 6282. The final product 8537238 is also circled.

# 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

$b = 1234$  ( $n=4$ )

$b = \underbrace{b_{n-1}b_{n-2}\dots b_0}_{1234}$  and  $c = \underbrace{c_{n-1}c_{n-2}\dots c_0}$

$b = \underbrace{b_{n-1}\dots b_{n/2}}_{12}0\dots 0 + \underbrace{b_{n/2-1}\dots b_0}_{34}$

$b(x) = b_Lx + b_R$ , where  $x = 10^{n/2}$ ,  $b_L = b_{n-1}\dots b_{n/2}$  and  $b_R = b_{n/2-1}\dots b_0$

Similarly  $c(x) = c_Lx + c_R$  where  $c_L = c_{n-1}\dots c_{n/2}$  and  $c_R = c_{n/2-1}\dots c_0$

$b = 1200 + 34$

$b(x) = 12x + 34; x = 10^2$

$10^{n/2}$

## Example

$$\begin{aligned}\underline{1234 \times 5678} &= \underline{(12x + 34)} \times \underline{(56x + 78)} \\ &= 12 \cdot 56 \cdot x^2 + (12 \cdot 78 + 34 \cdot 56)x + 34 \cdot 78.\end{aligned}$$

for  $x = \cancel{10}^{10^2}$

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

# Divide and Conquer for multiplication

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

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



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

$$\begin{aligned} T(n) \leftarrow bc &= \underline{b(x)c(x)} = \underline{(b_Lx + b_R)(c_Lx + c_R)} \\ &= b_Lc_Lx^2 + (b_Lc_R + b_Rc_L)x + b_Rc_R \\ &= 10^n \underline{(b_Lc_L)} + 10^{n/2} (\underline{b_Lc_R} + \underline{b_Rc_L}) + \underline{(b_Rc_R)} \\ &= 4T(n/2) + 4 + O(n) \end{aligned}$$

# 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)

$$\boxed{T(n) = 4T(n/2) + O(n)} \quad T(1) = O(1)$$

$$T(n) = O(n^2)$$

# 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)

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

$T(n) = \Theta(n^2)$ . 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  $(c + di)$

$$\underline{(a + bi)(c + di)} = \underline{ac - bd + (ad + bc)i}$$

# 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

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

$$p(x)q(x) = \underline{acx^2 + (ad + bc)x + bd}.$$

## Gauss technique for polynomials

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

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

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

## Improving the Running Time

$$\underline{bc} = b(x)c(x) = \underline{(b_Lx + b_R)(c_Lx + c_R)}$$

## Improving the Running Time

$$\begin{aligned}bc &= b(x)c(x) = (b_Lx + b_R)(c_Lx + c_R) \\&= b_Lc_Lx^2 + (b_Lc_R + b_Rc_L)x + b_Rc_R\end{aligned}$$

## Improving the Running Time

$$\begin{aligned}bc &= b(x)c(x) = (b_Lx + b_R)(c_Lx + c_R) \\&= b_Lc_Lx^2 + (b_Lc_R + b_Rc_L)x + b_Rc_R \\&= (b_L * c_L)x^2 + \left( (b_L + b_R) * (c_L + c_R) - b_L * c_L - b_R * c_R \right)x \\&\quad + b_R * c_R\end{aligned}$$

## Improving the Running Time

$$\begin{aligned}bc &= b(x)c(x) = (b_Lx + b_R)(c_Lx + c_R) \\&= b_Lc_Lx^2 + (b_Lc_R + b_Rc_L)x + b_Rc_R \\&= (b_L * c_L)x^2 + \left( (b_L + b_R) * (c_L + c_R) - b_L * c_L - b_R * c_R \right)x \\&\quad + b_R * c_R\end{aligned}$$

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

# Improving the Running Time

$$\begin{aligned}bc &= b(x)c(x) = (b_Lx + b_R)(c_Lx + c_R) \\&= b_Lc_Lx^2 + (b_Lc_R + b_Rc_L)x + b_Rc_R \\&= (b_L * c_L)x^2 + \left( (b_L + b_R) * (c_L + c_R) - b_L * c_L - b_R * c_R \right)x \\&\quad + b_R * c_R\end{aligned}$$

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

## Time Analysis

Running time is given by

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

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

# State of the Art

1971 - made practical  
1968 proposed  
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.

CS/ECE 314 - B  
Fall 23