

# Sorting Networks and Threshold Functions in Small Depth

Ramprasad Saptharishi

Chennai Mathematical Institute  
Third Year B.Sc Mathematics

November 9th, 2006

# Table of Contents

- 1 Preliminaries
- 2 Sorting Networks
- 3  $NC^2$  Sorting Networks
  - Bitonic Sort
  - Batcher's Sorting Network
- 4 Limitations
- 5  $MAJ \in \text{non-uniform } NC^1$
- 6 Greater Advantage
- 7 Summary

# Model of Computation

Computation over a circuit

# Model of Computation

## Computation over a circuit

- Basis being the  $\vee, \wedge, \neg$  gates having the obvious properties.

# Model of Computation

## Computation over a circuit

- Basis being the  $\vee, \wedge, \neg$  gates having the obvious properties.
- Forms a DAG with the inputs at the leaf level and output(s) as the sink node(s).

# Model of Computation

## Computation over a circuit

- Basis being the  $\vee, \wedge, \neg$  gates having the obvious properties.
- Forms a DAG with the inputs at the leaf level and output(s) as the sink node(s).
- The different constraints on the DAG defines different complexity classes.

# The AC and SAC Classes

$AC^i$ :

- Unbounded fanin gates

# The $AC$ and $SAC$ Classes

$AC^i$ :

- Unbounded fanin gates
- Polynomial size

# The AC and SAC Classes

$AC^i$ :

- Unbounded fanin gates
- Polynomial size
- Depth  $O(\log n)^i$

# The AC and SAC Classes

$AC^i$ :

- Unbounded fanin gates
- Polynomial size
- Depth  $O(\log n)^i$

$SAC^i$ :

- Unbounded fanin  $\wedge$  gates, bounded fanin  $\vee$  gates, no  $\neg$  gates.  
Negations of the input nodes given separately.

# The AC and SAC Classes

$AC^i$ :

- Unbounded fanin gates
- Polynomial size
- Depth  $O(\log n)^i$

$SAC^i$ :

- Unbounded fanin  $\wedge$  gates, bounded fanin  $\vee$  gates, no  $\neg$  gates. Negations of the input nodes given separately.
- Polynomial size

# The AC and SAC Classes

$AC^i$ :

- Unbounded fanin gates
- Polynomial size
- Depth  $O(\log n)^i$

$SAC^i$ :

- Unbounded fanin  $\wedge$  gates, bounded fanin  $\vee$  gates, no  $\neg$  gates. Negations of the input nodes given separately.
- Polynomial size
- Depth  $O(\log n)^i$

# The AC and SAC Classes

$AC^i$ :

- Unbounded fanin gates
- Polynomial size
- Depth  $O(\log n)^i$

$SAC^i$ :

- Unbounded fanin  $\wedge$  gates, bounded fanin  $\vee$  gates, no  $\neg$  gates. Negations of the input nodes given separately.
- Polynomial size
- Depth  $O(\log n)^i$

And  $AC = \bigcup AC^i, SAC = \bigcup SAC^i$ .

# The $NC$ Class

$NC^i$ :

- *Bounded fanin gates*

# The $NC$ Class

$NC^i$ :

- *Bounded* fanin gates
- Polynomial size

# The $NC$ Class

$NC^i$ :

- *Bounded* fanin gates
- Polynomial size
- Depth  $O(\log n)^i$

# The NC Class

$NC^i$ :

- *Bounded* fanin gates
- Polynomial size
- Depth  $O(\log n)^i$

And  $NC = \bigcup NC^i$

# Monotone Functions

For binary strings  $x, y$ , we can have the following partial order set up over them:  $x \leq y$  if  $x_i \leq y_i$  for all  $i$ .

A function  $f$  on bits  $x_1, \dots, x_n$  is monotone if  $x \leq y \implies f(x) \leq f(y)$

# Monotone Functions

For binary strings  $x, y$ , we can have the following partial order set up over them:  $x \leq y$  if  $x_i \leq y_i$  for all  $i$ .

A function  $f$  on bits  $x_1, \dots, x_n$  is monotone if  $x \leq y \implies f(x) \leq f(y)$

A circuit  $C$  is monotone if it doesn't have any  $\neg$  gates.

# Monotone Functions

For binary strings  $x, y$ , we can have the following partial order set up over them:  $x \leq y$  if  $x_i \leq y_i$  for all  $i$ .

A function  $f$  on bits  $x_1, \dots, x_n$  is monotone if  $x \leq y \implies f(x) \leq f(y)$

A circuit  $C$  is monotone if it doesn't have any  $\neg$  gates.

**Claim:** A boolean function is monotone *if and only if* it can be evaluated by a monotone circuit.

# Threshold Functions

$Th_k^n$ : Circuit with  $n$  inputs and one output gate. It outputs a 1 if there are at least  $k$  of the  $n$  inputs as 1.

# Threshold Functions

$Th_k^n$ : Circuit with  $n$  inputs and one output gate. It outputs a 1 if there are at least  $k$  of the  $n$  inputs as 1.

$$MAJ \equiv Th_{n/2}^n.$$

# Threshold Functions

$Th_k^n$ : Circuit with  $n$  inputs and one output gate. It outputs a 1 if there are at least  $k$  of the  $n$  inputs as 1.

$MAJ \equiv Th_{n/2}^n$ . Can this be done in monotone  $NC^2$ ?

# Threshold Functions

$Th_k^n$ : Circuit with  $n$  inputs and one output gate. It outputs a 1 if there are at least  $k$  of the  $n$  inputs as 1.

$MAJ \equiv Th_{n/2}^n$ . Can this be done in monotone  $NC^2$ ?

*Can this be done in  $NC^1$ ?*

# Table of Contents

- 1 Preliminaries
- 2 **Sorting Networks**
- 3  $NC^2$  Sorting Networks
  - Bitonic Sort
  - Batcher's Sorting Network
- 4 Limitations
- 5  $MAJ \in \text{non-uniform } NC^1$
- 6 Greater Advantage
- 7 Summary

# The Sorting Approach

Obviously if you can sort the inputs, then you can find the majority of the inputs.

# The Sorting Approach

Obviously if you can sort the inputs, then you can find the majority of the inputs.

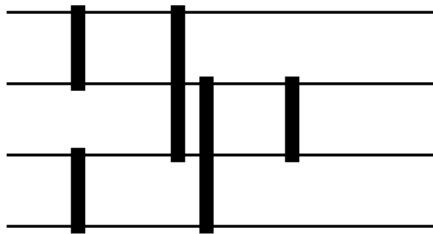
Qn: Can you sort in small depth?

# The Sorting Approach

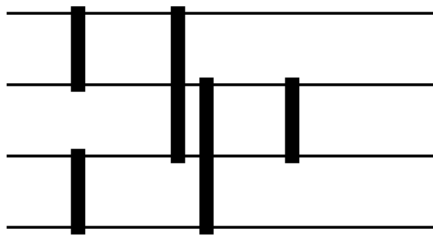
Obviously if you can sort the inputs, then you can find the majority of the inputs.

Qn: Can you sort in small depth?  $NC^2$ ?

# Sorting Networks: The model

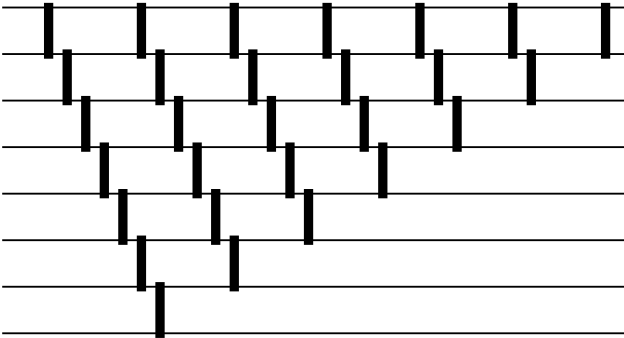


# Sorting Networks: The model

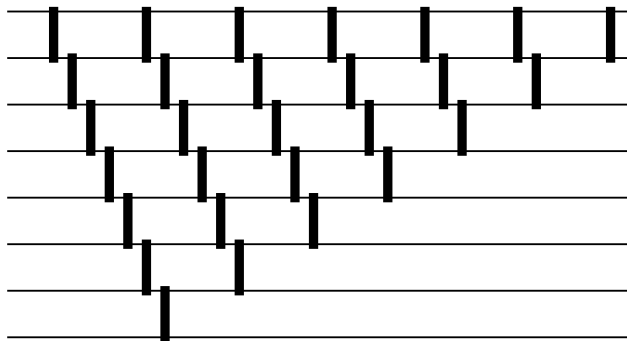


- Inputs lines running across, with *swap* gadgets in between.
- Depth and size have natural definitions.
- Non-adaptive

# Bubblesort Network



# Bubblesort Network



Depth =  $O(n)$

Size =  $O(n^2)$

# Table of Contents

- 1 Preliminaries
- 2 Sorting Networks
- 3  $NC^2$  Sorting Networks**
  - Bitonic Sort
  - Batcher's Sorting Network
- 4 Limitations
- 5  $MAJ \in \text{non-uniform } NC^1$
- 6 Greater Advantage
- 7 Summary

# Outline

- 1 Preliminaries
- 2 Sorting Networks
- 3  $NC^2$  Sorting Networks
  - Bitonic Sort
  - Batcher's Sorting Network
- 4 Limitations
- 5  $MAJ \in \text{non-uniform } NC^1$
- 6 Greater Advantage
- 7 Summary

# Bitonic Sort

A *Bitonic sequence* is one that has at most 2 changes in its monotonicity.

# Bitonic Sort

A *Bitonic sequence* is one that has at most 2 changes in its monotonicity.

Examples: 000011111111000, 11110011111111.

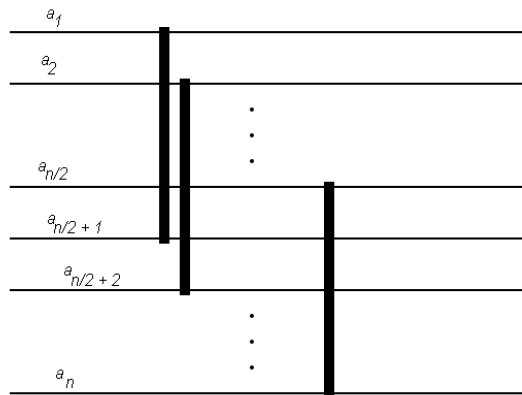
# Bitonic Sort

A *Bitonic sequence* is one that has at most 2 changes in its monotonicity.

Examples: 000011111111000, 11110011111111.

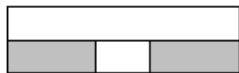
Recursion: Divide it into two parts  $A$  and  $B$  such that each of them is bitonic, and every element of  $A$  is less than or equal to every element in  $B$ .

# The $B_n$ Network

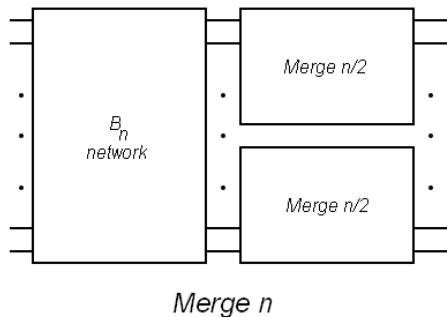


A *swap* gadget connecting  $i$  and  $\frac{n}{2} + i$ ; a depth 1 gadget.

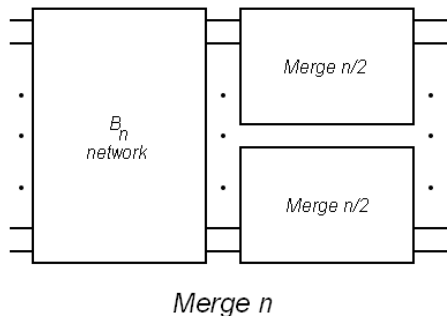
# $B_n$ on a Bitonic Sequence



## Sorting a bitonic sequence



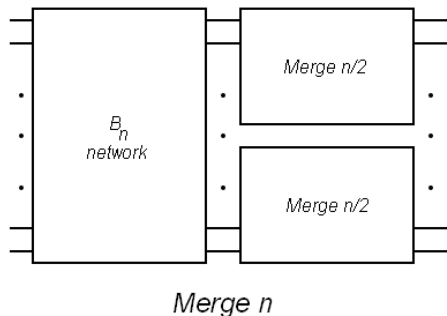
## Sorting a bitonic sequence



Recurrence Relation:

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

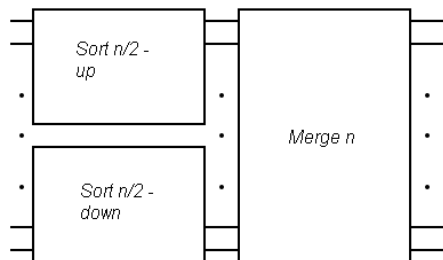
## Sorting a bitonic sequence



Recurrence Relation:

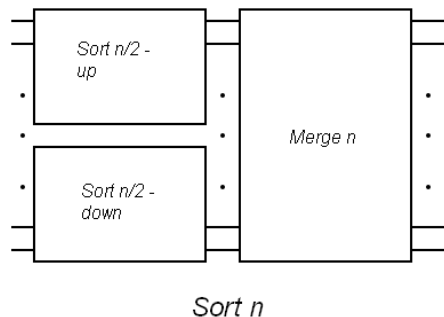
$$D(n) = D\left(\frac{n}{2}\right) + c \Rightarrow D(n) = O(\log n)$$

# The Final Network



*Sort  $n$*

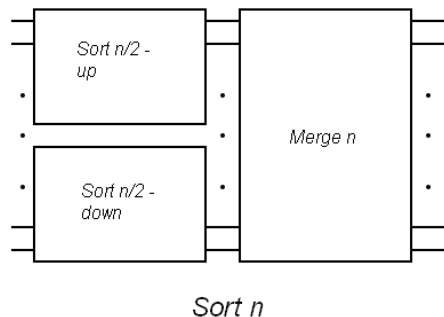
# The Final Network



Recurrence Relation:

$$D(n) = D\left(\frac{n}{2}\right) + O(\log n)$$

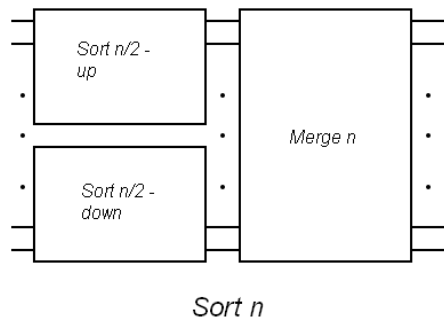
# The Final Network



Recurrence Relation:

$$D(n) = D\left(\frac{n}{2}\right) + O(\log n) \Rightarrow D(n) = O(\log^2 n)$$

## The Final Network



Recurrence Relation:

$$D(n) = D\left(\frac{n}{2}\right) + O(\log n) \Rightarrow D(n) = O(\log^2 n)$$

Thus we have a  $O(\log^2 n)$  depth sorting network and thus  $O(\log^2 n)$  depth circuit for majority.

# Outline

- 1 Preliminaries
- 2 Sorting Networks
- 3  $NC^2$  Sorting Networks**
  - Bitonic Sort
  - **Batcher's Sorting Network**
- 4 Limitations
- 5  $MAJ \in \text{non-uniform } NC^1$
- 6 Greater Advantage
- 7 Summary

# The Batcher's Sorting Network

The beauty of this network basically lies in the *merge* operation, it can be done in constant depth.

# The Batcher's Sorting Network

The beauty of this network basically lies in the *merge* operation, it can be done in constant depth.

- Sort the first half and the second half in parallel, recursively

# The Batcher's Sorting Network

The beauty of this network basically lies in the *merge* operation, it can be done in constant depth.

- Sort the first half and the second half in parallel, recursively
- Do the *odd-even merge*

# The Odd-Even Merge

# The Odd-Even Merge

*Input:* 2 sorted sequences of length  $n$ , say  $a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n$

# The Odd-Even Merge

*Input:* 2 sorted sequences of length  $n$ , say  $a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n$

Recursively merge the sequences  $\{a_1, a_3, \dots, b_1, b_3, \dots\}$  and  $\{a_2, a_4, \dots, b_2, b_4, \dots\}$ , to output the 2 sequences say  $a'_1, a'_2, \dots, a'_n, b'_1, b'_2, \dots, b'_n$ .

# The Odd-Even Merge

*Input:* 2 sorted sequences of length  $n$ , say  $a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n$

Recursively merge the sequences  $\{a_1, a_3, \dots, b_1, b_3, \dots\}$  and  $\{a_2, a_4, \dots, b_2, b_4, \dots\}$ , to output the 2 sequences say  $a'_1, a'_2, \dots, a'_n, b'_1, b'_2, \dots, b'_n$ .

**Claim:** The smallest element is  $a'_1$ , the largest element is  $b'_n$ . The  $i$ th smallest and  $(i + 1)$ th smallest are between  $a'_i$  and  $b'_{i-1}$ .

# The Odd-Even Merge

*Input:* 2 sorted sequences of length  $n$ , say  $a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n$

Recursively merge the sequences  $\{a_1, a_3, \dots, b_1, b_3, \dots\}$  and  $\{a_2, a_4, \dots, b_2, b_4, \dots\}$ , to output the 2 sequences say  $a'_1, a'_2, \dots, a'_n, b'_1, b'_2, \dots, b'_n$ .

**Claim:** The smallest element is  $a'_1$ , the largest element is  $b'_n$ . The  $i$ th smallest and  $(i + 1)$ th smallest are between  $a'_i$  and  $b'_{i-1}$ .

Which means we can then have a merge layer of depth 1 to sort the list.

# The Odd-Even Merge

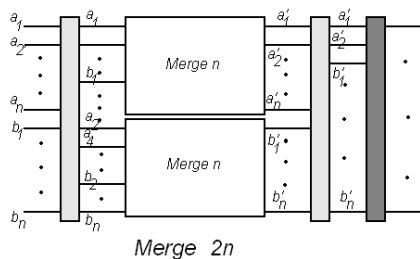
*Input:* 2 sorted sequences of length  $n$ , say  $a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n$

Recursively merge the sequences  $\{a_1, a_3, \dots, b_1, b_3, \dots\}$  and  $\{a_2, a_4, \dots, b_2, b_4, \dots\}$ , to output the 2 sequences say  $a'_1, a'_2, \dots, a'_n, b'_1, b'_2, \dots, b'_n$ .

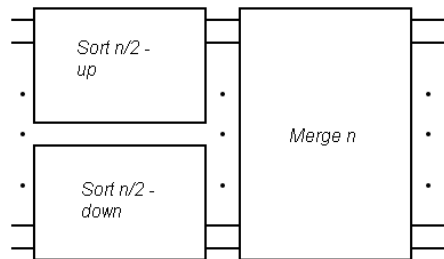
**Claim:** The smallest element is  $a'_1$ , the largest element is  $b'_n$ . The  $i$ th smallest and  $(i + 1)$ th smallest are between  $a'_i$  and  $b'_{i-1}$ .

Which means we can then have a merge layer of depth 1 to sort the list. And it's easy to see that this takes  $O(\log n)$  depth and hence the total sorting operation is of depth  $O(\log^2 n)$  depth.

# Batcher's Network



*Merge 2n*



*Sort n*

# Table of Contents

- 1 Preliminaries
- 2 Sorting Networks
- 3  $NC^2$  Sorting Networks
  - Bitonic Sort
  - Batcher's Sorting Network
- 4 Limitations**
- 5  $MAJ \in \text{non-uniform } NC^1$
- 6 Greater Advantage
- 7 Summary

# A Perfect Partitioner

A *Perfect Partitioner* is a network that partitions the inputs into two halves  $A$  and  $B$  such that *every* element of  $A$  is less than *every* element of  $B$ .

# A Perfect Partitioner

A *Perfect Partitioner* is a network that partitions the inputs into two halves  $A$  and  $B$  such that *every* element of  $A$  is less than *every* element of  $B$ .

It is clear that once we have this *perfect partitioner* we can sort the inputs in  $O(\log n)$  steps using it.

# A Perfect Partitioner

A *Perfect Partitioner* is a network that partitions the inputs into two halves  $A$  and  $B$  such that *every* element of  $A$  is less than *every* element of  $B$ .

It is clear that once we have this *perfect partitioner* we can sort the inputs in  $O(\log n)$  steps using it.

Does there exist a *quick perfect partitioner*?

# A Perfect Partitioner

A *Perfect Partitioner* is a network that partitions the inputs into two halves  $A$  and  $B$  such that *every* element of  $A$  is less than *every* element of  $B$ .

It is clear that once we have this *perfect partitioner* we can sort the inputs in  $O(\log n)$  steps using it.

Does there exist a *quick perfect partitioner*?

Unfortunately no...

# A Perfect Partitioner

A *Perfect Partitioner* is a network that partitions the inputs into two halves  $A$  and  $B$  such that every element of  $A$  is less than every element of  $B$ .

It is clear that once we have this *perfect partitioner* we can sort the inputs in  $O(\log n)$  steps using it.

Does there exist a *quick perfect partitioner*?

Unfortunately no...

## Theorem

Any perfect partitioner will have depth at least  $\log n$ .

# A Perfect Partitioner

A *Perfect Partitioner* is a network that partitions the inputs into two halves  $A$  and  $B$  such that every element of  $A$  is less than every element of  $B$ .

It is clear that once we have this *perfect partitioner* we can sort the inputs in  $O(\log n)$  steps using it.

Does there exist a *quick perfect partitioner*?

Unfortunately no...

## Theorem

Any perfect partitioner will have depth at least  $\log n$ .

Perfect partitioner approach  $\rightarrow O(\log^2 n)$  depth.

# The Philosophy

*Perfection is costly, dare to err and rectify them later, you shall be more efficient.*

# The Philosophy

*Perfection is costly, dare to err and rectify them later, you shall be more efficient.*

This philosophy was used in the [AjtaiKolmosSzemeridi83] paper to give an  $O(\log n)$  sorting network. The *approximate* partitioner was done cleverly using expander graphs.

# AKS Sorters: Sketch

- Split the inputs into two halves, which are *roughly* sorted.
- Keep sending back elements to compensate for errors.
- Do this for constantly many levels to get a better approximate sorting.
- Do this for  $O(\log n)$  steps to sort all but a finite chunk of inputs.  
Brute force sort these elements.

## Why isn't this popular?

- AKS sorting has depth  $O(\log n)$  compared to bitonic/batcher that has depth  $O((\log n)^2)$ .
- Clearly AKS is asymptotically much faster than batcher and bitonic
- Why do people still prefer the other two over AKS?

## Why isn't this popular?

- AKS sorting has depth  $O(\log n)$  compared to bitonic/batcher that has depth  $O((\log n)^2)$ .
- Clearly AKS is asymptotically much faster than batcher and bitonic
- Why do people still prefer the other two over AKS?
- How big are the constants behind the  $O$ 's?

## Why isn't this popular?

- AKS sorting has depth  $O(\log n)$  compared to bitonic/batcher that has depth  $O((\log n)^2)$ .
- Clearly AKS is asymptotically much faster than batcher and bitonic
- Why do people still prefer the other two over AKS?
- How big are the constants behind the  $O$ 's?

Bitonic can be done in  $4(\log n)^2$

## Why isn't this popular?

- AKS sorting has depth  $O(\log n)$  compared to bitonic/batcher that has depth  $O((\log n)^2)$ .
- Clearly AKS is asymptotically much faster than batcher and bitonic
- Why do people still prefer the other two over AKS?
- How big are the constants behind the  $O$ 's?

Bitonic can be done in  $4(\log n)^2$

AKS Sorting can be done in around  $6000(\log n)$  and hence works better for  $n > 2^{1500}$

## Why isn't this popular?

- AKS sorting has depth  $O(\log n)$  compared to bitonic/batcher that has depth  $O((\log n)^2)$ .
- Clearly AKS is asymptotically much faster than batcher and bitonic
- Why do people still prefer the other two over AKS?
- How big are the constants behind the  $O$ 's?

Bitonic can be done in  $4(\log n)^2$

AKS Sorting can be done in around  $6000(\log n)$  and hence works better for  $n > 2^{1500}$ , and also incredibly complex to implement.

## Why isn't this popular?

- AKS sorting has depth  $O(\log n)$  compared to bitonic/batcher that has depth  $O((\log n)^2)$ .
- Clearly AKS is asymptotically much faster than batcher and bitonic
- Why do people still prefer the other two over AKS?
- How big are the constants behind the  $O$ 's?

Bitonic can be done in  $4(\log n)^2$

AKS Sorting can be done in around  $6000(\log n)$  and hence works better for  $n > 2^{1500}$ , and also incredibly complex to implement.

Any other proofs?

# Table of Contents

- 1 Preliminaries
- 2 Sorting Networks
- 3  $NC^2$  Sorting Networks
  - Bitonic Sort
  - Batcher's Sorting Network
- 4 Limitations
- 5  $MAJ \in \text{non-uniform } NC^1$
- 6 Greater Advantage
- 7 Summary

# A Randomized Approach: Valiant's Construction

# A Randomized Approach: Valiant's Construction

- If  $MAJ(x_1, \dots, x_n) = 1$  then for each  $x_i$ , the  $\Pr[x_i = 1] \geq \frac{1}{2} + \frac{2}{n+1}$

# A Randomized Approach: Valiant's Construction

- If  $MAJ(x_1, \dots, x_n) = 1$  then for each  $x_i$ , the  $\Pr[x_i = 1] \geq \frac{1}{2} + \frac{2}{n+1}$
- Use appropriate circuitry to amplify this probability

# A Randomized Approach: Valiant's Construction

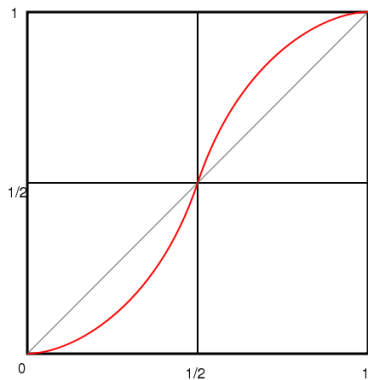
- If  $MAJ(x_1, \dots, x_n) = 1$  then for each  $x_i$ , the  $\Pr[x_i = 1] \geq \frac{1}{2} + \frac{2}{n+1}$
- Use appropriate circuitry to amplify this probability
- After amplifying enough, use a union bound to show that atleast one circuit should be right on all inputs

## The $Th_2^3$ function

If each of the inputs has a probability of  $p$  to be turned on, then the  $Th_2^3$  gate will fire with probability  $f(p) = 3(1 - p)p^2 + p^3 = 3p^2 - 2p^3$

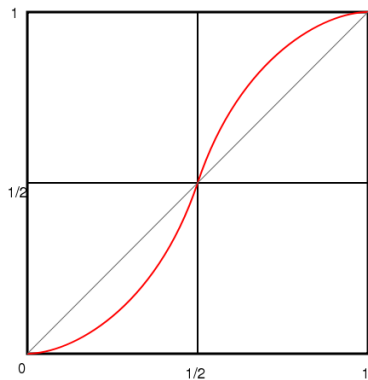
## The $Th_2^3$ function

If each of the inputs has a probability of  $p$  to be turned on, then the  $Th_2^3$  gate will fire with probability  $f(p) = 3(1 - p)p^2 + p^3 = 3p^2 - 2p^3$



## The $Th_2^3$ function

If each of the inputs has a probability of  $p$  to be turned on, then the  $Th_2^3$  gate will fire with probability  $f(p) = 3(1 - p)p^2 + p^3 = 3p^2 - 2p^3$



Also  $f'(p) = 6p(1 - p)$ , this is a good candidate for amplification purposes.

# Randomization

# Randomization

- Use  $Th_2^3$  gates to amplify probability

# Randomization

- Use  $Th_2^3$  gates to amplify probability
- Feed in inputs to the threshold gates as  $x_i$  with probability  $\frac{1}{2}$ , or randomly fix it as 0 or 1 with probability  $\frac{1}{4}$

# Randomization

- Use  $Th_2^3$  gates to amplify probability
- Feed in inputs to the threshold gates as  $x_i$  with probability  $\frac{1}{2}$ , or randomly fix it as 0 or 1 with probability  $\frac{1}{4}$
- Now, probability that the threshold gate gets an input as 1 is  $\frac{1}{2} + \frac{1}{n+1}$

# Randomization

- Use  $Th_2^3$  gates to amplify probability
- Feed in inputs to the threshold gates as  $x_i$  with probability  $\frac{1}{2}$ , or randomly fix it as 0 or 1 with probability  $\frac{1}{4}$
- Now, probability that the threshold gate gets an input as 1 is  $\frac{1}{2} + \frac{1}{n+1}$

We shall take the case where  $MAJ(x_1, x_2, \dots, x_n) = 0$ , the other case is symmetric.

# Advantage Amplification

$$\text{Step 1: } \frac{1}{2} - \frac{1}{n+1} \rightarrow \frac{1}{4}$$

## Advantage Amplification

$$\text{Step 1: } \frac{1}{2} - \frac{1}{n+1} \rightarrow \frac{1}{4}$$

By the Mean Value Theorem, we have

$$f\left(\frac{1}{2} - x\right) = f\left(\frac{1}{2}\right) - xf'(\theta)$$

for some  $\frac{1}{4} \leq \theta \leq \frac{1}{2}$ .

## Advantage Amplification

Step 1:  $\frac{1}{2} - \frac{1}{n+1} \rightarrow \frac{1}{4}$

By the Mean Value Theorem, we have

$$f\left(\frac{1}{2} - x\right) = f\left(\frac{1}{2}\right) - xf'(\theta)$$

for some  $\frac{1}{4} \leq \theta \leq \frac{1}{2}$ .

And hence, iterating this  $k$  times for  $\frac{1}{2} - \frac{1}{n+1}$ ,

$$f^k\left(\frac{1}{2} - \frac{1}{n+1}\right) \leq \frac{1}{2} - \left(\frac{9}{8}\right)^k \frac{1}{n+1}$$

## Advantage Amplification

Step 1:  $\frac{1}{2} - \frac{1}{n+1} \rightarrow \frac{1}{4}$

By the Mean Value Theorem, we have

$$f\left(\frac{1}{2} - x\right) = f\left(\frac{1}{2}\right) - xf'(\theta)$$

for some  $\frac{1}{4} \leq \theta \leq \frac{1}{2}$ .

And hence, iterating this  $k$  times for  $\frac{1}{2} - \frac{1}{n+1}$ ,

$$f^k\left(\frac{1}{2} - \frac{1}{n+1}\right) \leq \frac{1}{2} - \left(\frac{9}{8}\right)^k \frac{1}{n+1}$$

And hence with  $k = O(\log n)$  we can get it down to  $\frac{1}{4}$ .

# Advantage Amplification

Step 2:  $\frac{1}{4} \rightarrow \frac{1}{2^n}$

## Advantage Amplification

Step 2:  $\frac{1}{4} \rightarrow \frac{1}{2^n}$

This is even simpler, we know that  $f(p) = 3p^2 - 2p^3 \leq 3p^2 \leq \frac{3}{4}p$  for  $p < \frac{1}{4}$ .

## Advantage Amplification

Step 2:  $\frac{1}{4} \rightarrow \frac{1}{2^n}$

This is even simpler, we know that  $f(p) = 3p^2 - 2p^3 \leq 3p^2 \leq \frac{3}{4}p$  for  $p < \frac{1}{4}$ .

Hence with  $O(\log n)$  iterations you can hit  $\frac{1}{2^n}$

# The Union Bound

We have a random circuit of  $O(\log n)$  depth that evaluate majority with a error probability of less than  $p = 2^{-poly(n)}$ . And one can show that the number of  $O(\log n)$  circuits is larger than  $\frac{1}{p}$  for suitable choice of the constants and the polynomials.

Since the probability of error is less than  $\frac{1}{\text{number of circuits}}$ , there should be atleast one such circuit that evaluates correctly for every input. And hence we have a circuit of  $O(\log n)$  depth for majority (non-uniform though).  $\square$

# Table of Contents

- 1 Preliminaries
- 2 Sorting Networks
- 3  $NC^2$  Sorting Networks
  - Bitonic Sort
  - Batcher's Sorting Network
- 4 Limitations
- 5  $MAJ \in \text{non-uniform } NC^1$
- 6 Greater Advantage
- 7 Summary

# More Advantage, easier to compute

- *MAJ* corresponds to inverse polynomial advantage.

# More Advantage, easier to compute

- *MAJ* corresponds to inverse polynomial advantage.
- What if the advantage was larger, say inverse polylog?

## More Advantage, easier to compute

- $MAJ$  corresponds to inverse polynomial advantage.
- What if the advantage was larger, say inverse polylog?
- Could it be done in a smaller class?

# More Advantage, easier to compute

- *MAJ* corresponds to inverse polynomial advantage.
- What if the advantage was larger, say inverse polylog?
- Could it be done in a smaller class?

## Ajtai-BenOr says “Yes”

Can amplify  $\frac{1}{2} \left(1 + \frac{1}{(\log n)^r}\right)$  to  $1 - \frac{1}{2^n}$  in  $AC^0$

# More Advantage, easier to compute

- *MAJ* corresponds to inverse polynomial advantage.
- What if the advantage was larger, say inverse polylog?
- Could it be done in a smaller class?

## Ajtai-BenOr says “Yes”

Can amplify  $\frac{1}{2} \left(1 + \frac{1}{(\log n)^r}\right)$  to  $1 - \frac{1}{2^n}$  in  $AC^0 \subsetneq NC^1$

This shall be presented in perhaps another talk.

# Table of Contents

- 1 Preliminaries
- 2 Sorting Networks
- 3  $NC^2$  Sorting Networks
  - Bitonic Sort
  - Batcher's Sorting Network
- 4 Limitations
- 5  $MAJ \in \text{non-uniform } NC^1$
- 6 Greater Advantage
- 7 Summary

# Summary

- $MAJ \in \text{non-uniform } NC^1$

# Summary

- $MAJ \in \text{non-uniform } NC^1$
- *Perfect Partitioner* approach will only yield us  $O(\log^2 n)$  depth networks

# Summary

- $MAJ \in \text{non-uniform } NC^1$
- *Perfect Partitioner* approach will only yield us  $O(\log^2 n)$  depth networks
- *Approximate Partitioners* can do better: AKS Sorters using  $\epsilon$  halvers

# Summary

- $MAJ \in \text{non-uniform } NC^1$
- *Perfect Partitioner* approach will only yield us  $O(\log^2 n)$  depth networks
- *Approximate Partitioners* can do better: AKS Sorters using  $\epsilon$  halvers
- AKS network provided the first proof that  $MAJ \in \text{uniform } NC^1$ . Huge constant, around 6000.

# Summary

- $MAJ \in \text{non-uniform } NC^1$
- *Perfect Partitioner* approach will only yield us  $O(\log^2 n)$  depth networks
- *Approximate Partitioners* can do better: AKS Sorters using  $\epsilon$  halvers
- AKS network provided the first proof that  $MAJ \in \text{uniform } NC^1$ . Huge constant, around 6000.
- $MAJ$  with inverse polylog promise can be solved in  $AC^0$ .

# Thank You

Slides and T<sub>E</sub>Xsources are available at  
<http://www.cmi.ac.in/~ramprasad/studenttalks/sorting>