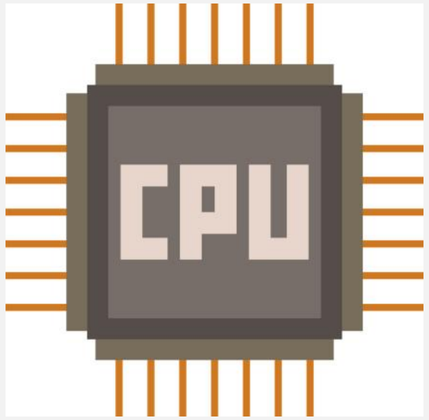
WEEK 13
PARALLEL ALGORITHMS

2024-04-11

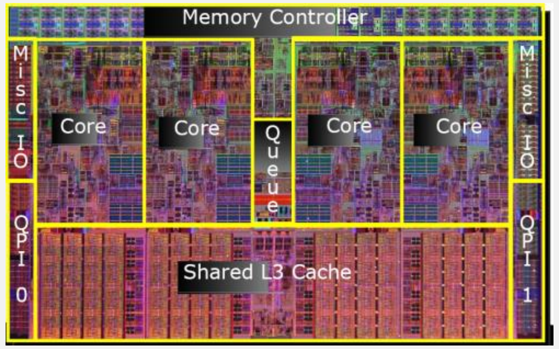
WEAKNESSES OF TRADITIONAL ALGORITHMS

- Traditional algorithms are designed for a single processor so that every computer can run.
- There is a limitation of "working alone". Even if the recursive algorithms use friends to help, but the friends are just a clone of the same processor.

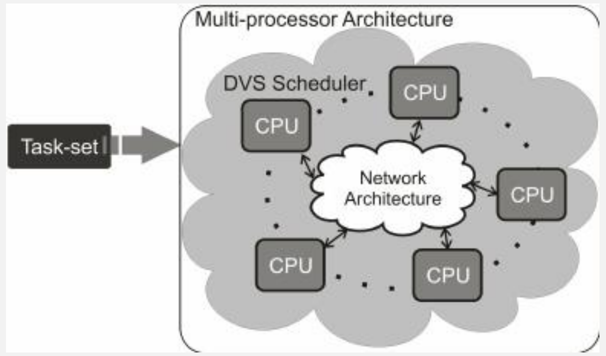


MULTI-PROCESSOR ARCHITECTURE

- Modern processors (after y2k) are designed with multi-processor (multi-core) architecture
- So they are able to handle multi-task at the same time.
- Multi-processor can be implemented in the form of connecting computers into a network so that a parallel task can be assigned to multiple nodes



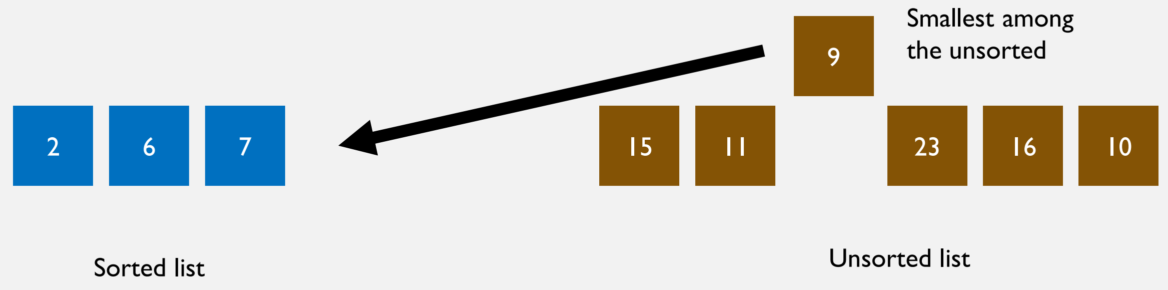
https://hexus.net/tech/reviews/cpu/16187-intel-core-i7-x58-chipset-systems-go-fsb-invited/?page=3

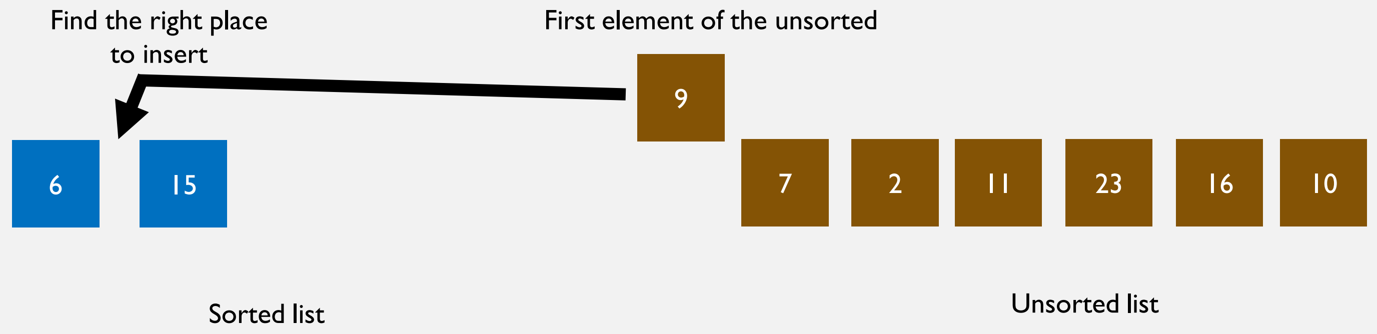


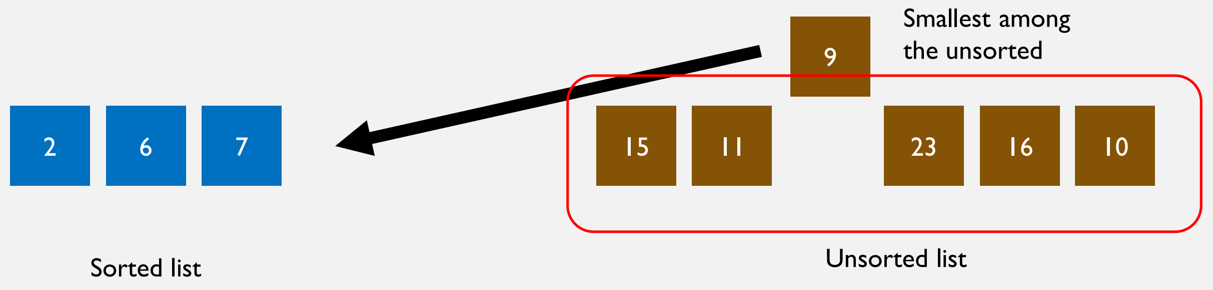
https://www.researchgate.net/figure/Abstraction-of-a-multiple-processors-environment-connected-by-abstract-network_fig2_227943038

CAN WE SORT FASTER WITH MULTI-PROCESSOR?

- Recall the selection sort and insertion sort







The unsorted list depends on which object has been chosen in the current iteration.

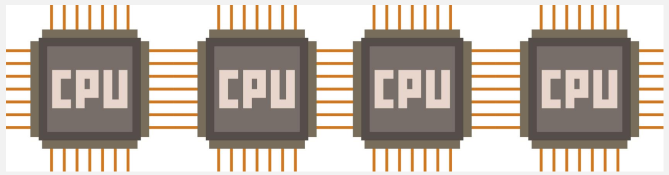
Even we have more than one processor, we have to wait until the current object has settled with the sorted list, then we can process the remaining object.

Iterative sorting algorithms have the bottleneck in sense that only one object can be processed at a time. Multi-processor is unable to improve the speed.

New kind of algorithm must be designed to utilize multi-processer with the full potential.

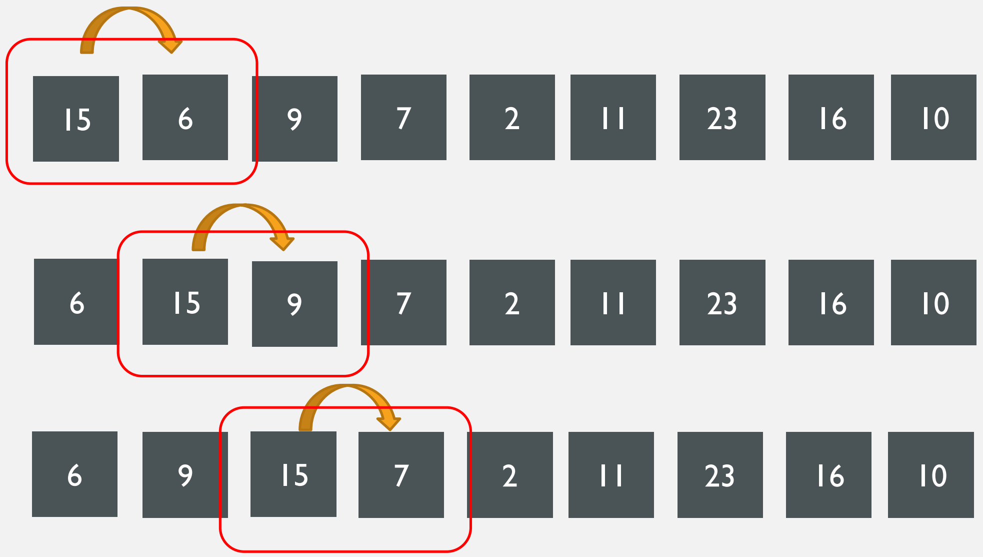
GENERAL IDEA FOR PARALLEL ALGORITHMS

- Always make the processors busy
- Tasks assigned to multi-processor must be independent of order, i.e., can be done without waiting for resources from other process in the same level



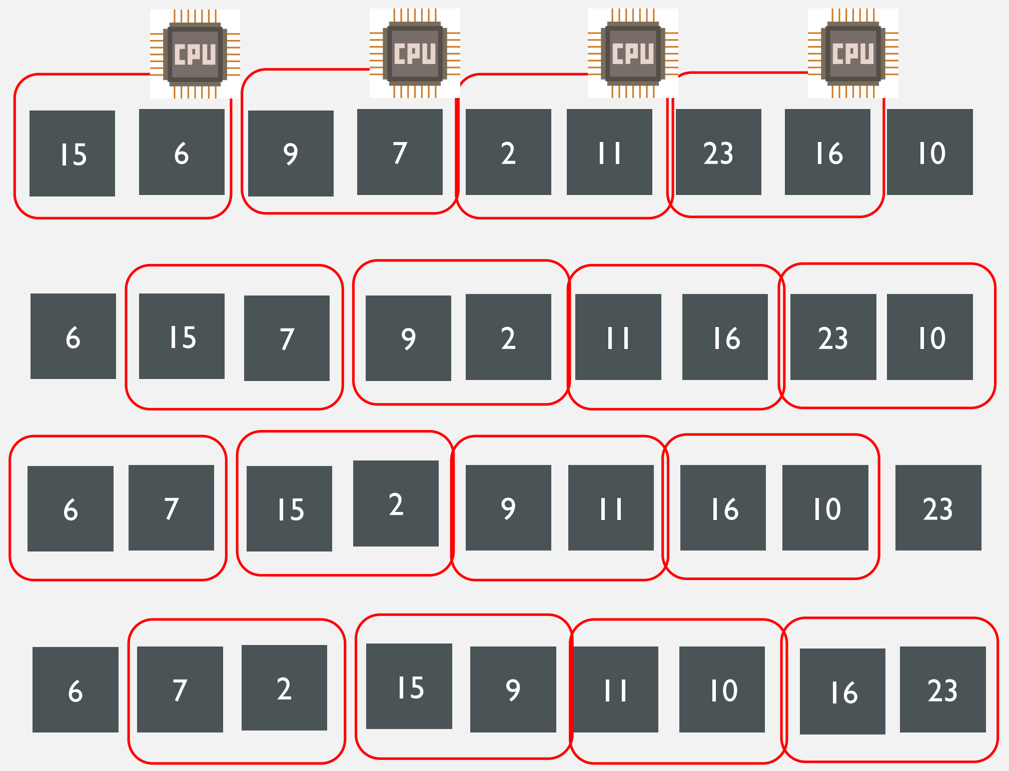
- We assume that the number of processors is more than enough, and see how fast can we improve the time complexity.

TRADITIONAL BUBBLE SORT



- Compare and exchange adjacent objects until they are all sorted
- Bubble sort takes $O(n^2)$ iterations with single processor
- Is there anything to improve by using multi-processor?
  - Swapping the second pair of objects depends on the result of the first pair swapping.
  - What to do in order to make a multiple swapping at the same time?

PARALLEL BUBBLE SORT



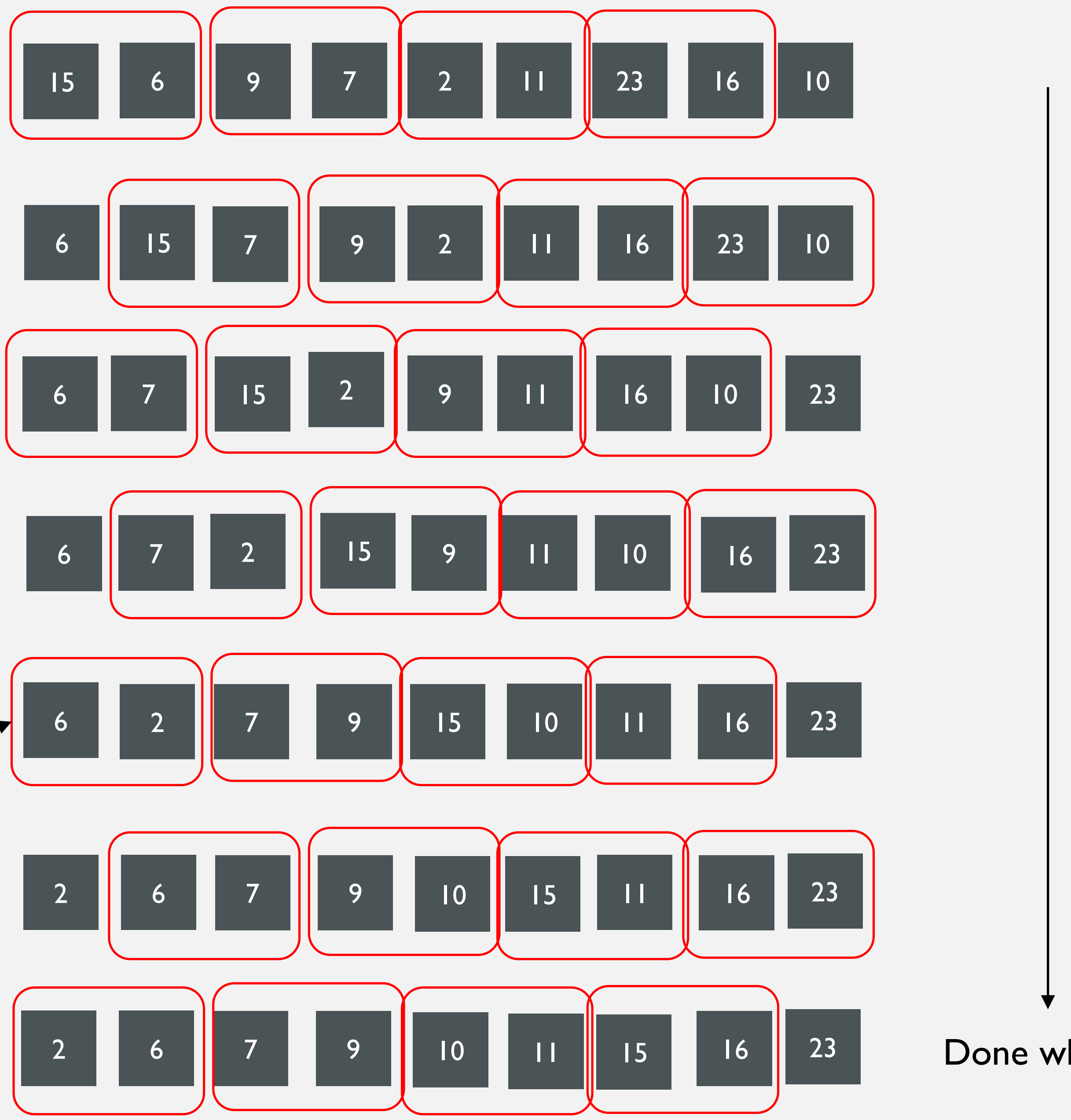
- Also called odd-even sort or brick sort
- If the second pair must wait for the result of the first pair, we can do compare and exchange at the third pair
- While working with the third pair, we can work with the 5^{th} pair since there is no overlap between them
- Parallel bubble sort assign "compare and exchange" to all of the odd pairs at the same time
- Then the next iteration will be done on the even pairs
- So we can work with $\frac{n}{2}$ pairs at once in just 1 unit of time.

Compare and exchange the odd pairs

Even pairs

Odd pairs again

Compare and exchange takes $O(1)$ time



At most n iterations

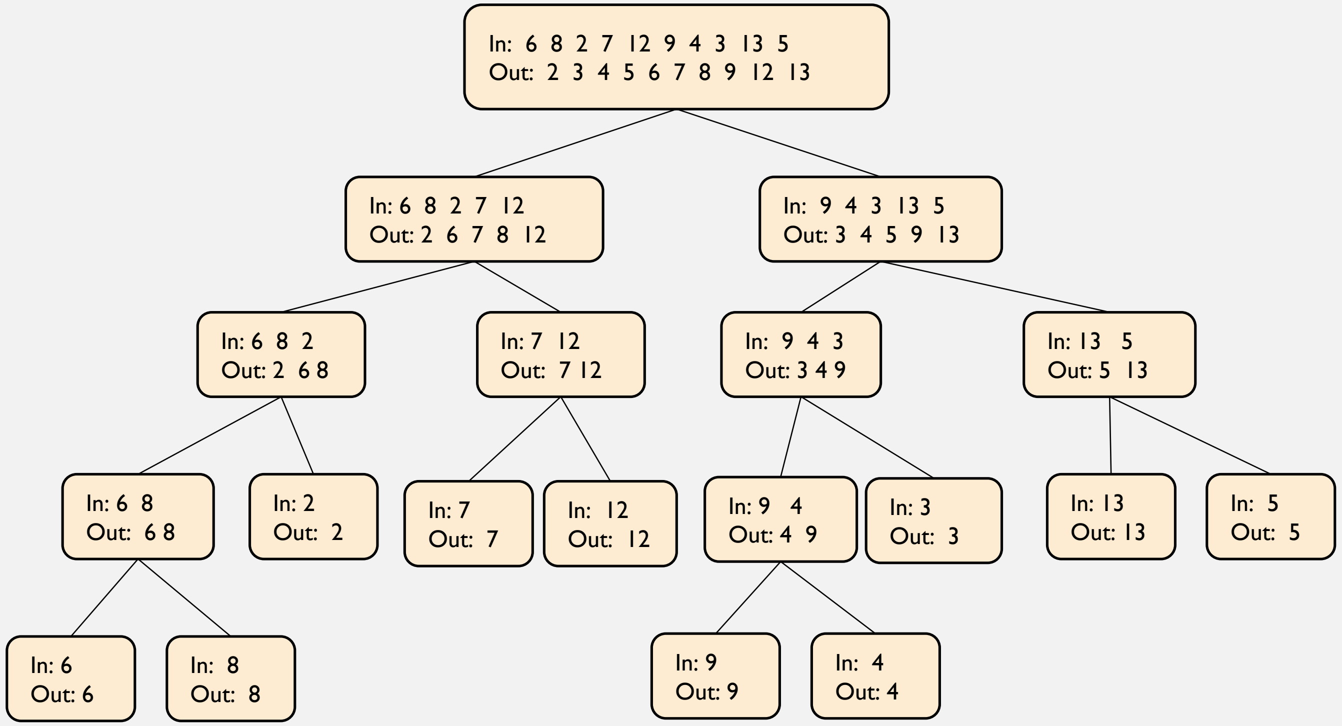
Done when everything is sorted

TIME COMPLEXITY

- Compare and exchange takes $O(1)$ time
- There are at most n iterations
- Parallel bubble sort $O(n)$ iterations for unlimited processors

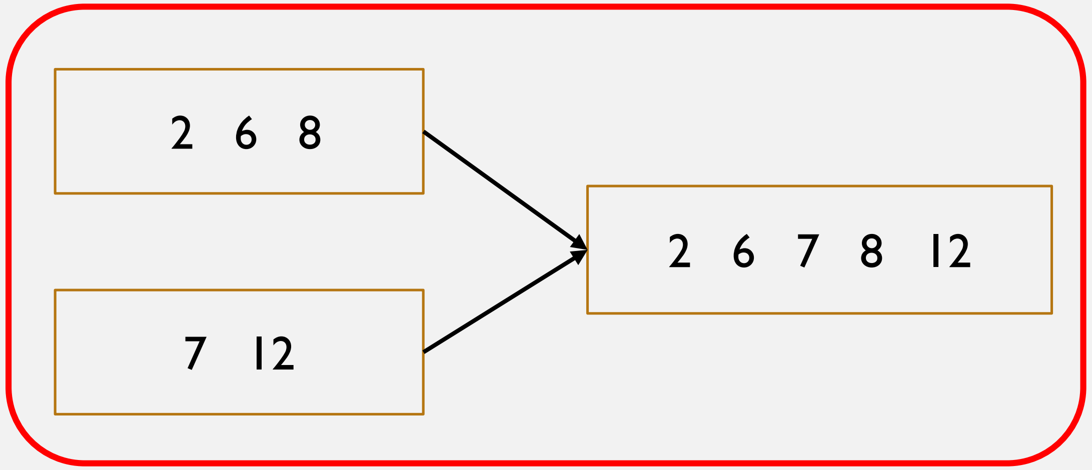
"Sorting in linear time !!!!"

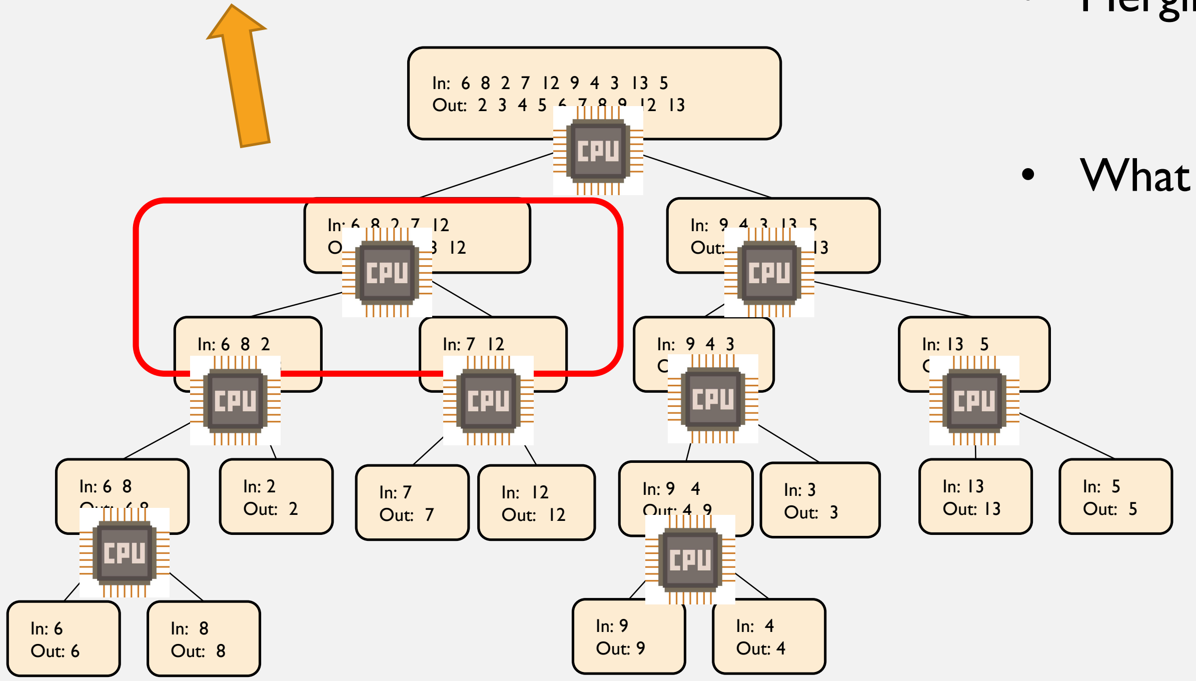
PARALLEL MERGE SORT



- Recall: Merge sort is a recursive sorting
- It seems like assigning friends to help, but friends are just a copy of ourselves.
- Where should we ask multi-processor to do in parallel?

Merging lists

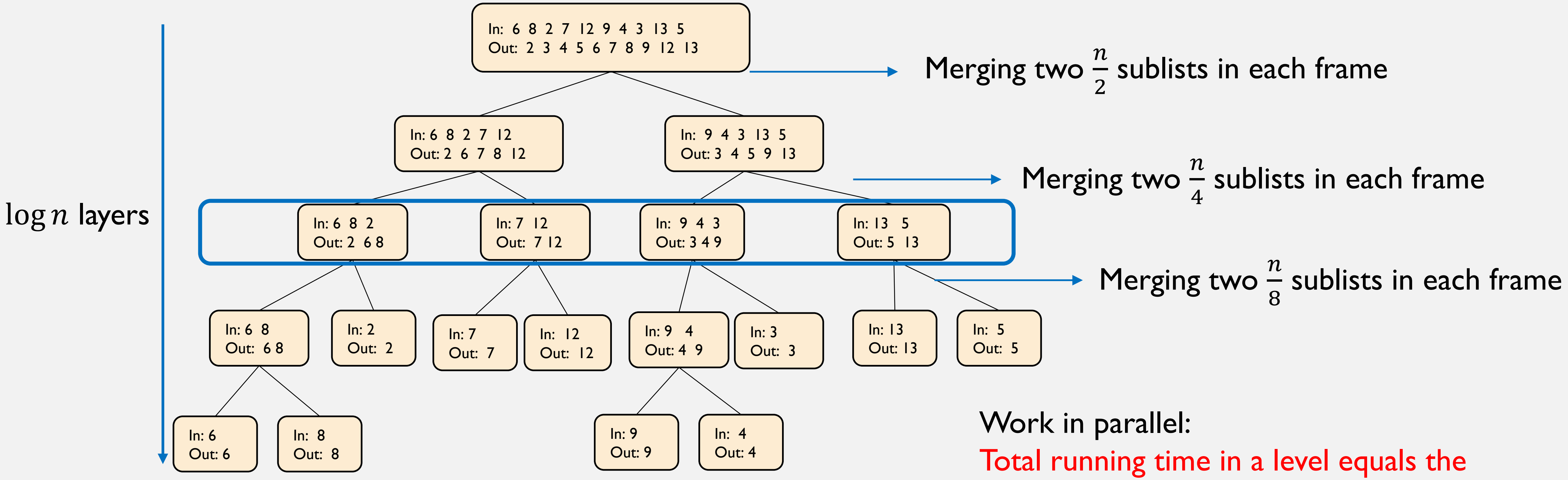




- Merging is where the actions take place.
- Merging two sublists of size $\frac{n}{2}$ takes n loops, hence $O(n)$ time.

- What if we assign multi-processor to help merging?

TIME COMPLEXITY



Work in parallel:

Total running time in a level equals the running time of just one processor

TIME COMPLEXITY

Level	Input size	Work in stack frame	~~Number of frames~~	~~Total work (recursive)~~	Total time (parallel)
0	n	$f(n) = n$	~~1~~	~~n~~	n
1	$\frac{n}{2}$	$f\left(\frac{n}{2}\right) = \frac{n}{2}$	~~2~~	~~n~~	$\frac{n}{2}$
2	$\frac{n}{2^2}$	$f\left(\frac{n}{2^2}\right) = \frac{n}{2^2}$	~~2^2~~	~~n~~	$\frac{n}{4}$
…	…	…	~~…~~	~~n~~	
$k = \log_2 n$	$\frac{n}{2^k} = 1$	$T(1) = 1$	~~2^k~~	~~n~~	1

$\sum total\ time \approx 2n$

Time complexity = $O(n)$

Does not matter anymore!!

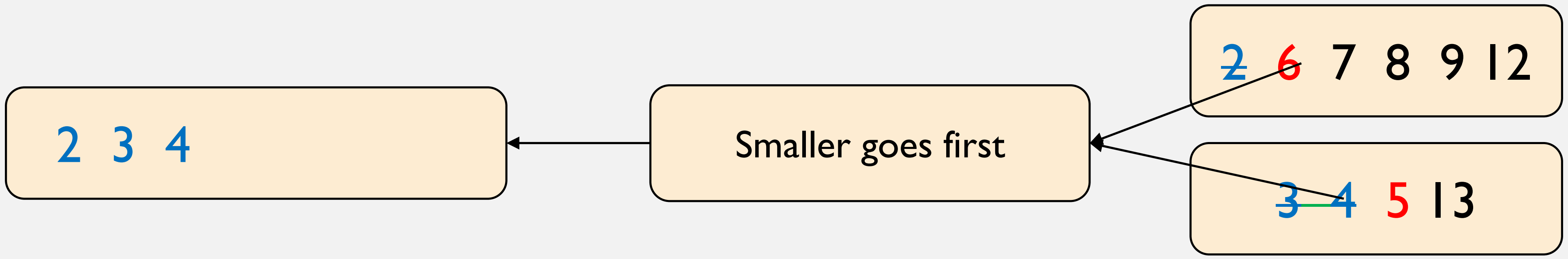
ODD-EVEN MERGE SORT

Level	Input size	Work in stack frame
0	n	$f(n) = n$
1	$\frac{n}{2}$	$f\left(\frac{n}{2}\right) = \frac{n}{2}$
2	$\frac{n}{2^2}$	$f\left(\frac{n}{2^2}\right) = \frac{n}{2^2}$
…	…	…
$k = \log_2 n$	$\frac{n}{2^k} = 1$	$T(1) = 1$

- Weakness of the merge sort is that the time complexity of merging depends on n.
- Merging itself cannot be parallelized, once assign a processor to merge, it must be done within this single processor.
- Odd-even merge sort makes the parallel merge sort faster by assigning multi-processors to contribute to merging lists of any size uniformly.

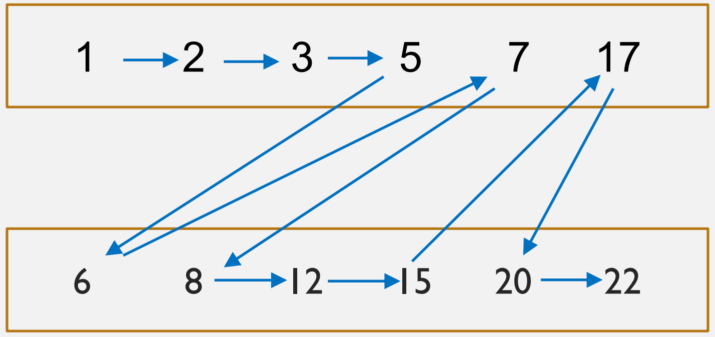
TRADITIONAL MERGING

- Merging used to be a single processor task.
- "Smaller goes first" must be considering the head of the remaining sublists only.
- Even if we have more processors, the remaining sublists depends on the current choice. It has to wait until one of the heads is taken.
- It is not obvious to parallelize merging algorithm.

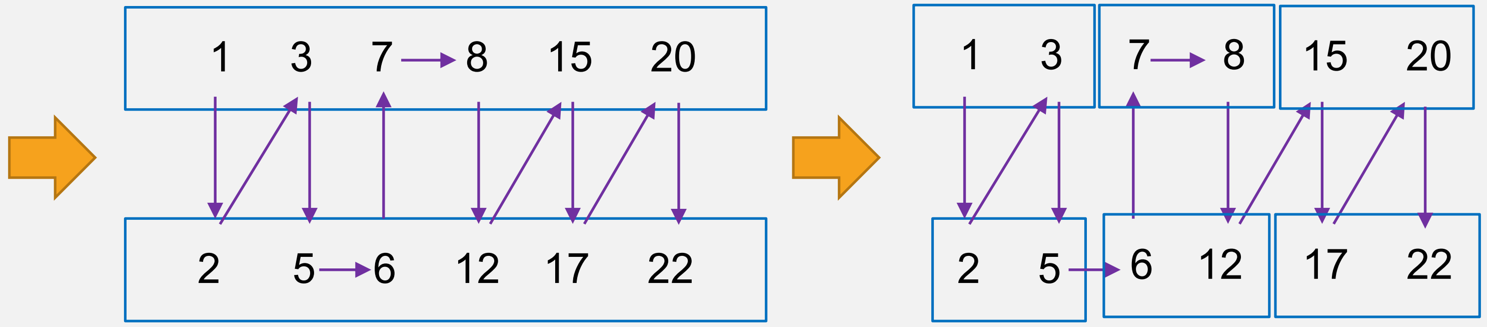


ODD-EVEN MERGE

- Expectation
  - If the two sublists are in a good shape, i.e., smaller object comes from the head of the first and the second sublists alternatively (or repeat for not more than 2 times), merging can be parallelized.

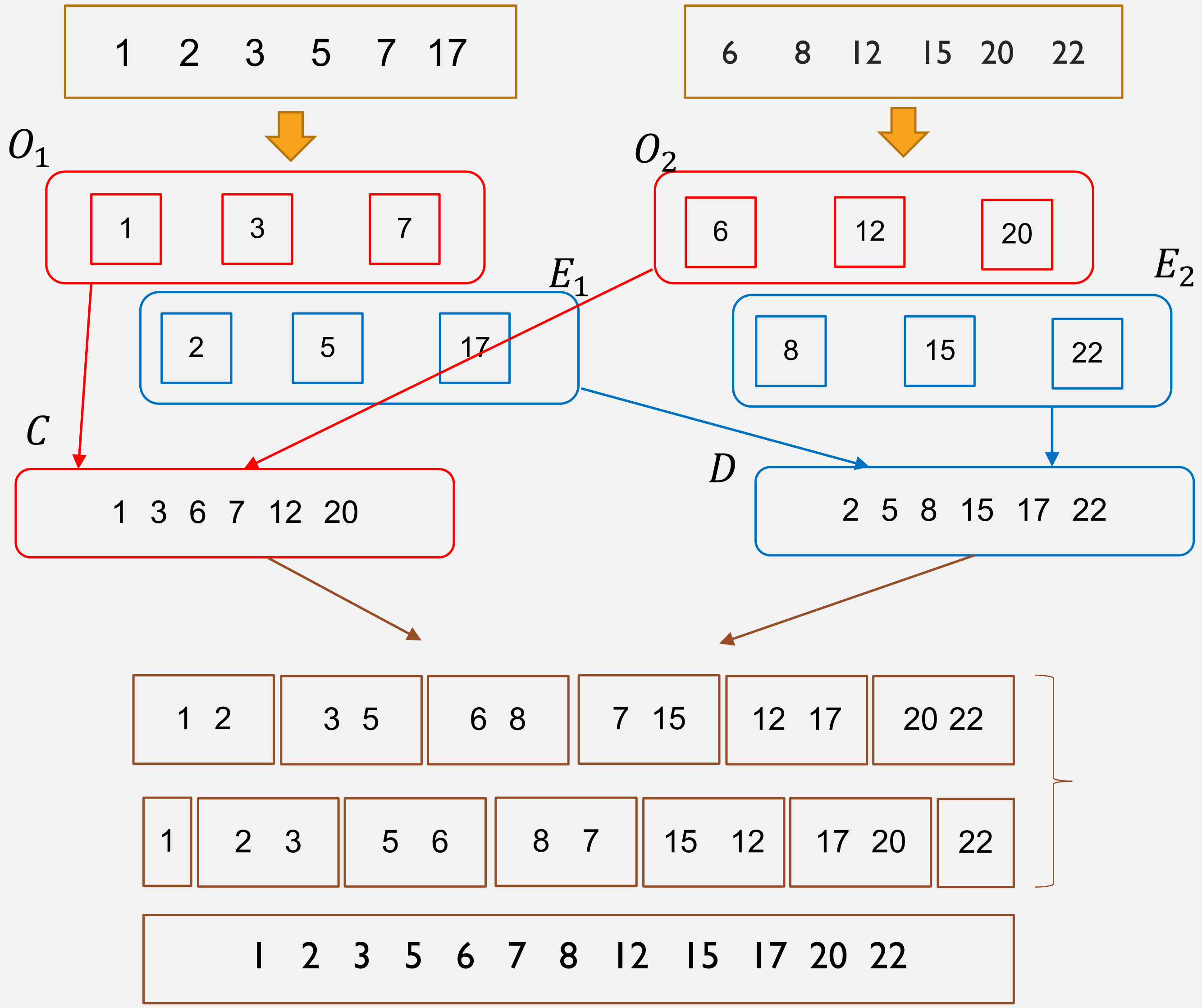


Cannot do parallel merge



Can be made parallel

ODD-EVEN MERGE



- Generate O_1, O_2 and E_1, E_2 sublists, containing ODD and EVEN objects from sublist1 and sublist2

- Merge O_1 and O_2 into C, and merge E_1 and E_2 into D by using the recursive parallel merge (this algorithm itself).
- Claim that C and D now are in the good shape for assigning to parallel processor to merge.

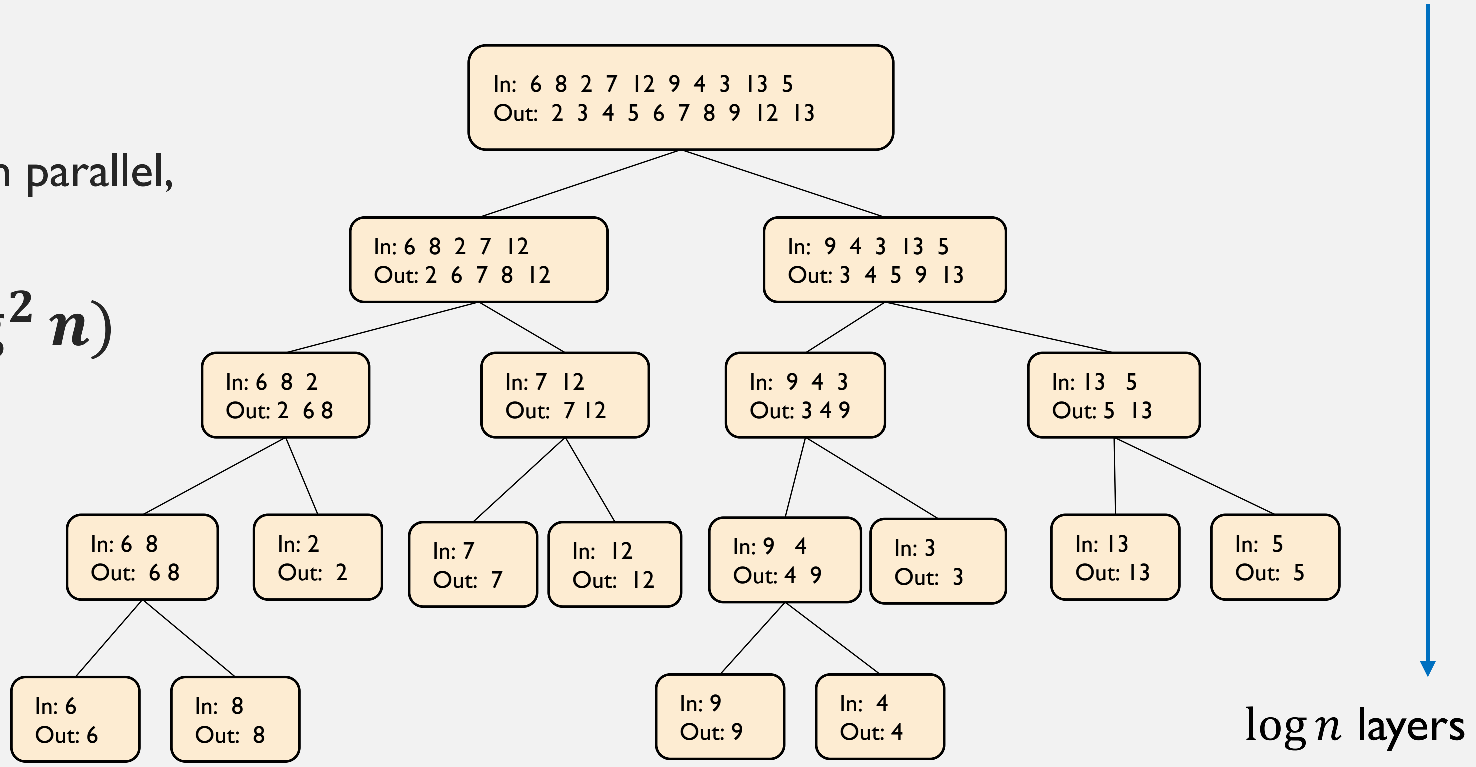
- Perform odd-even sort for 2 iterations.
  Then they are merged successfully.

TIME COMPLEXITY (ODD-EVEN **MERGING ONLY**)

- Odd-even merge is a recursive process that has $\log n$ levels of recursion
- In each level, halving, interleaving, and pairwise merging can be parallelized. So the time complexity in each level is constant.
- Overall odd-even merging takes $O(\log n)$ time

ODD-EVEN MERGE **SORT**

- We then use odd-even merge to replace the traditional merging in the merge sort.
- There are $\log n$ levels in the merge sort.
- Each level must perform odd-even merging in parallel, which takes $O(\log n)$ time.
- Overall odd-even merge sort takes $\boldsymbol{O(\log^2 n)}$


In: 6 8 2 7 12 9 4 3 13 5
Out: 2 3 4 5 6 7 8 9 12 13
In: 6 8 2 7 12
Out: 2 6 7 8 12
In: 9 4 3 13 5
Out: 3 4 5 9 13
In: 6 8 2
Out: 2 6 8
In: 7 12
Out: 7 12
In: 9 4 3
Out: 3 4 9
In: 13 5
Out: 5 13
In: 6 8
Out: 6 8
In: 2
Out: 2
In: 7
Out: 7
In: 12
Out: 12
In: 9 4
Out: 4 9
In: 3
Out: 3
In: 13
Out: 13
In: 5
Out: 5
In: 6
Out: 6
In: 8
Out: 8
In: 9
Out: 9
In: 4
Out: 4
$\log n$ layers