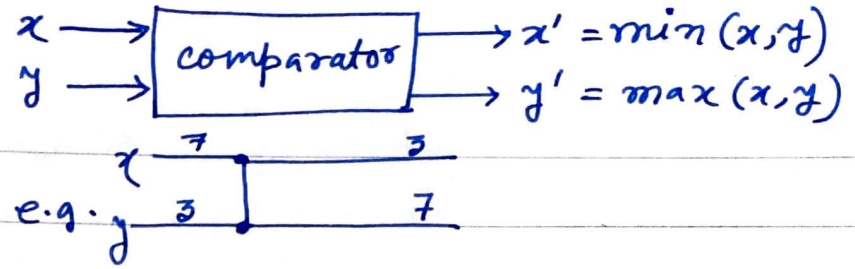
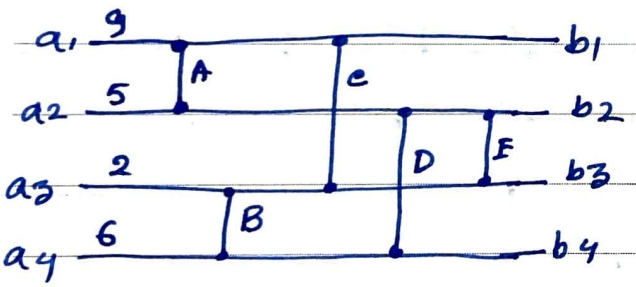
Network Sorting:

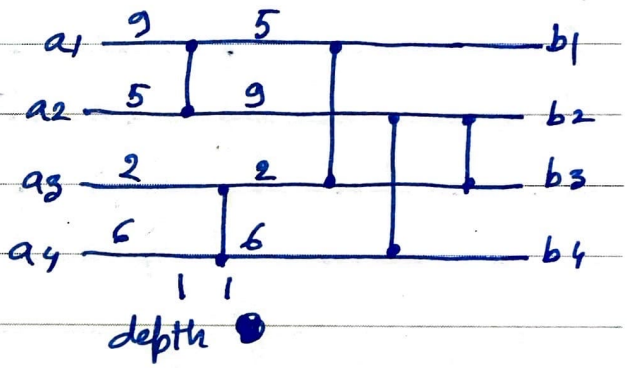
Comparison network differs from RAM's in two important respects. First, they can only perform comparisons. Thus, an algorithm such as counting sort cannot be implemented on a comparison network. Second, unlike the RAM model, in which operations occur serially - that is, one after another -- operations in a comparison network may occur at the same time, or "in parallel." This characteristic allows the construction of comparison networks that sort n values in sublinear time. In all the previous sorting algorithms, we could compare an element A[i] of the array A with an element A[j] of the array, as may be required by the specific algorithm, i.e., there was no restriction between such i and j values other than the requirement of the algorithm concerned.

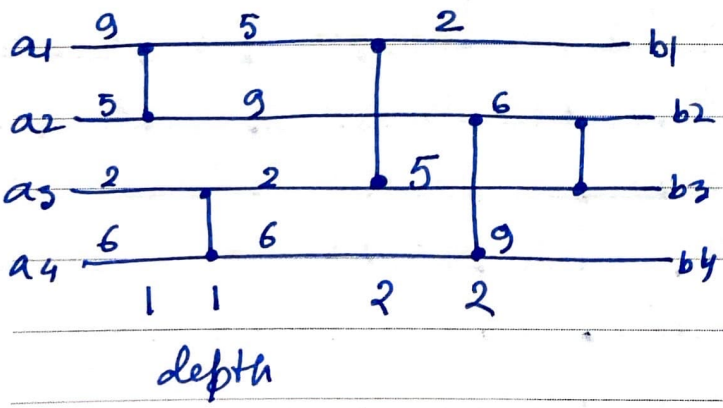
In constrast to this, assume that the elements are available on data lines of a network (similar to databus of a computer system), instead of being available in a RAM, e.g., within an array stored in the main memory of the system. Also assume that there are some pre-fixed comparator modules, where each comparator module takes two datalines as its input, and outputs the sorted values on its two output data lines,

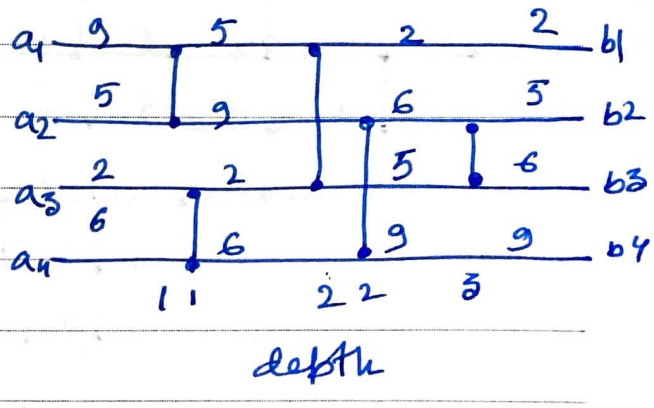


A wire transmits a value from place to place. Wires can connect the output of one comparator to input of another, but otherwise they are either network input wires or network output wires. Therefore, a comparison network contains n input wires $a_1, a_2, \ldots, a_n$ through which the values to be sorted enter the network, and n output wires $b_1, b_2, \ldots, b_n$ which produce the results computed by the network.







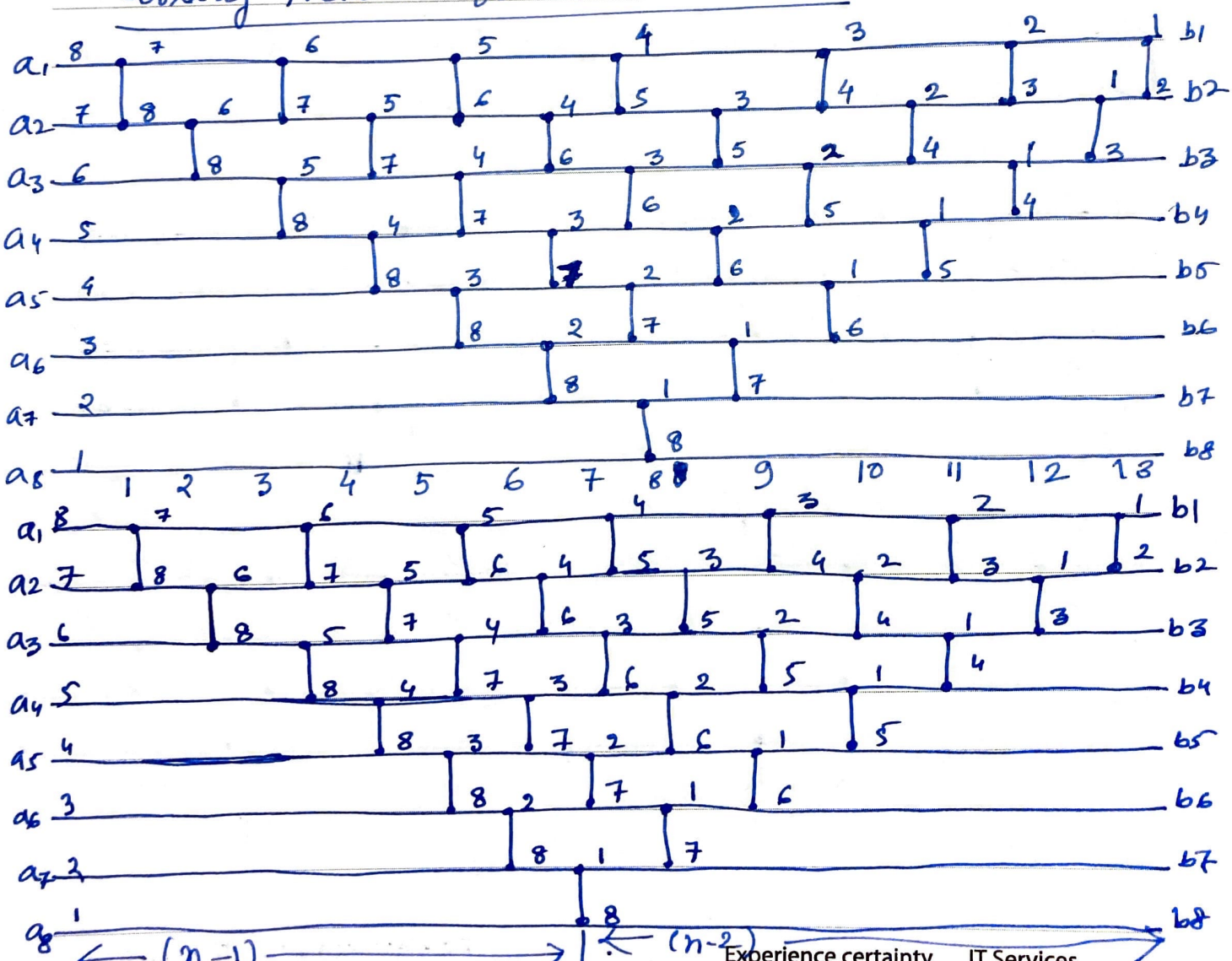


A sorting network is a comparison network for which the output sequence is monotonically increasing for every input sequence.

Depth of wires:

An input wire of a comparison network has depth 0. Now if a comparator has two input wires with depth d_x and d_y, then its output wires have depth $\max(d_x, d_y)+1$.

Sorting Network for insertion sort



Odd-Even Transposition Sort:

5 odd 3 even 1 odd 4 even 2 odd 6

C1 C2 C3 C4 C5

odd	3	5	1	4	2	6
even	3	1	5	2	4	6
odd	1	3	2	5	4	6
even	1	2	3	4	5	6
odd	1	2	3	4	5	6
even	1	2	3	4	5	6

5 odd 3 even 4 odd 2 even 1

C1 C2 C3 C4

odd	3	5	2	4	1
even	3	2	5	1	4
odd	2	3	1	5	4
even	2	1	3	4	5
odd	1	2	3	4	5

n steps are required.

no. of comparator required in sorting network =

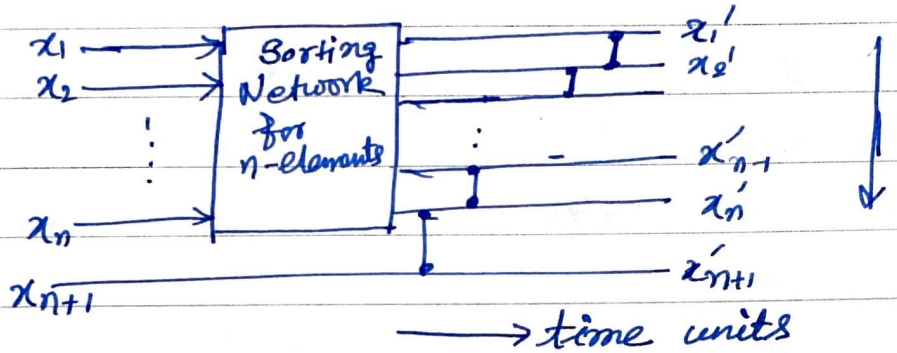
$$(n-1) + (n-2) + \cdots + 1$$

$$= \frac{n(n-1)}{2}$$

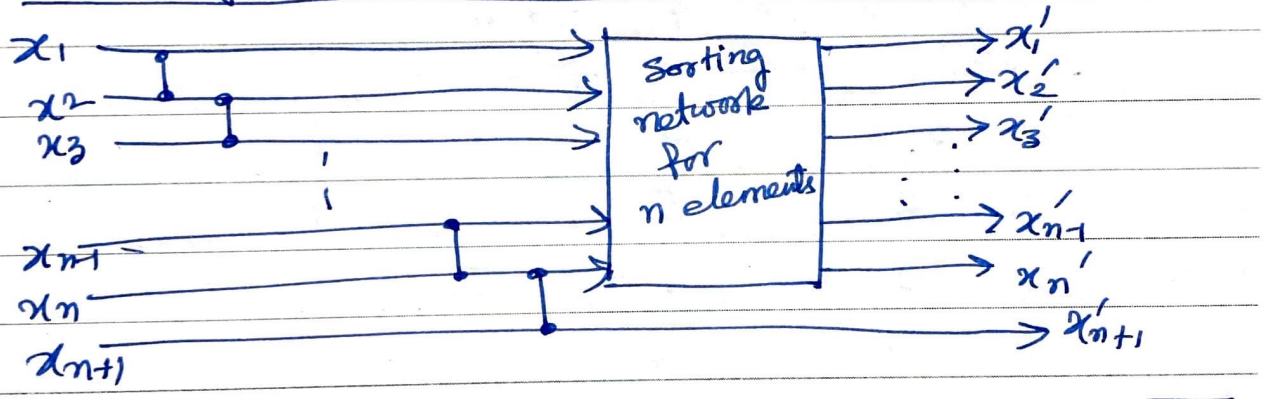
total time required $= (n-1)+(n-2) = 2n-3$.

The simple ways to construct a sorting network for n+1 elements when an n-element sorting network is given.

① By using the principle of insertion:



② By using the principle of bubble or selection:



In odd-even transposition sort, total n steps are required.

Zero-One Principle:

There may be numerous possible network structure to sort a given set of elements. Proving the correctness of any given sorting network certainly involves proving the fact (usually through well-formed logical arguments) that the given network correctly sorts all possible input permutations of the n elements, leading to a perfectly sorted sequence of the input values.

Here is the big trouble! To argue for all possible $n!$ permutations of the input values is usually a very difficult task. However, the following fact, as stated in the form of "zero one principle", simplifies this task, by reducing the total number of test permutations of input values from $n!$ to 2^n. $n! \geq 2^n$, for $n \geq 4$.

Zero-One Principle:

The zero-one principle says that if a sorting network works correctly when each input is drawn from the set $\{0, 1\}$, then it works correctly on arbitrary input numbers. i.e. if a network with n input lines sorts all possible 2^n sequences of 0's and 1's into non-decreasing order, it will sort any arbitrary sequence of n numbers into non-decreasing order.

Proof:

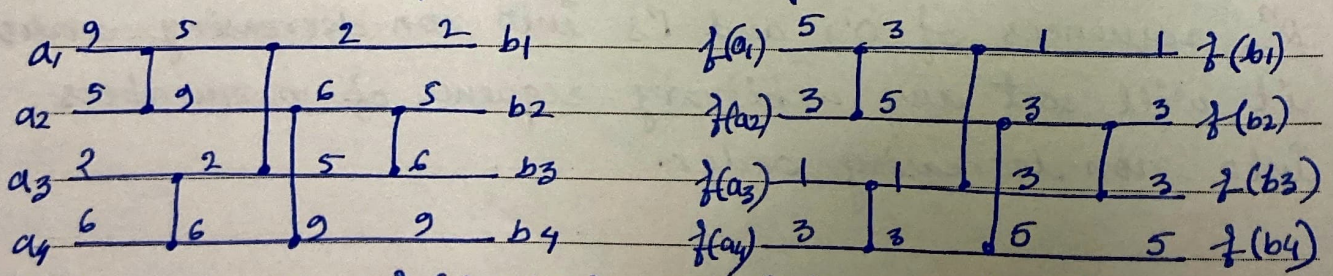
Lemma: If a comparison network transforms the input sequence $a = \langle a_1, a_2, \ldots, a_n \rangle$ into the output sequence $b = \langle b_1, b_2, \ldots, b_n \rangle$, then for any monotonically increasing function f, the network transforms the input sequence $f(a) = \langle f(a_1), f(a_2), \ldots, f(a_n) \rangle$ into the output sequence $f(b) = \langle f(b_1), f(b_2), \ldots, f(b_n) \rangle$.

Proof of Lemma: Consider a comparator whose input values are x and y. The upper output of the comparator is $\min(x,y)$ and the lower output is $\max(x,y)$. Suppose, we now apply $f(x)$ and $f(y)$ to the inputs of the comparator. The operation of the comparator yields the value $\min(f(x), f(y))$ on the upper output and the value $\max(f(x), f(y))$ on the lower output. Since f is monotonically increasing, $x \le y$ implies $f(x) \le f(y)$.

$\therefore \min(f(x), f(y)) = f(\min(x,y))$.

& $\max(f(x), f(y)) = f(\max(x,y))$.

As an example, a monotonically increasing function $f(x) = \lceil x/2 \rceil$ applied to to inputs.

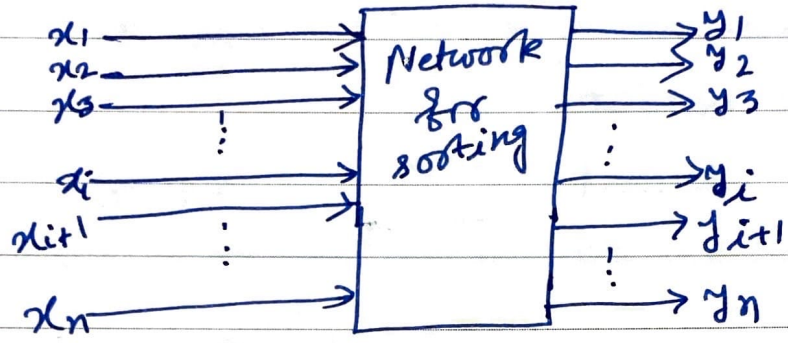


Let $x \le y$ thus $f(x) \le f(y)$ $\therefore \min(x,y) = x$ & $\min(f(x), f(y)) = f(x)$

$\therefore f(\min(x,y)) = f(x) =$ $\min(f(x), f(y))$

Proof:

Assume that a network sorts all possible 2^n sequences of 0's and 1's given at its input, but it fails to sort a given sequence (permutation) of some arbitrary n values, taken from the domain of its input values.



Specifically, assume that for the input values $x_1, x_2, \cdots x_n$, the network produces an output sequence which is not sorted with the deviation of required ordering between y_i and y_{i+1} at the output, i.e., Let

$$y_1 \le y_2 \le \cdots \le y_i > y_{i+1} \le y_{i+2} \le \cdots \le y_n$$

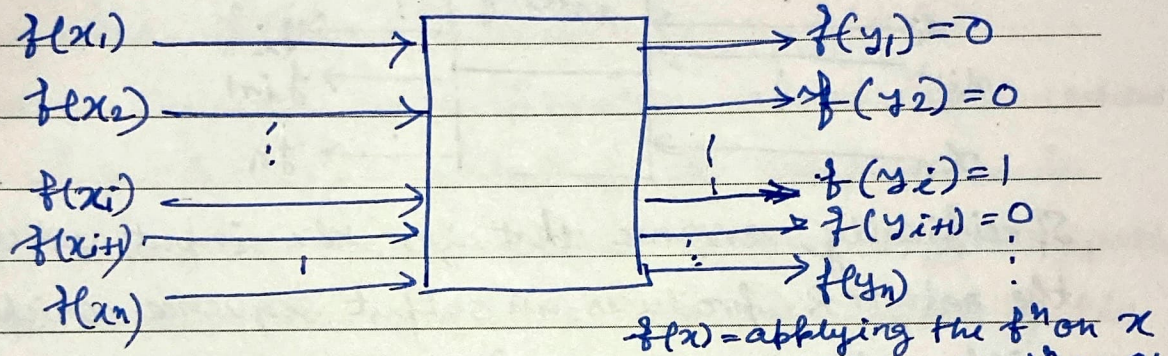
this should have been $y_i \le y_{i+1}$!!

If the above situation happens, then let us chose the monotonic function $f(x)$ as follows: (to map an arbitrary sequence to a sequence of '0' & '1')

$$f(x) = \begin{cases} 0 & \text{for all } x \le y_{i+1} \\ 1 & \text{for all } x > y_{i+1} \end{cases}$$

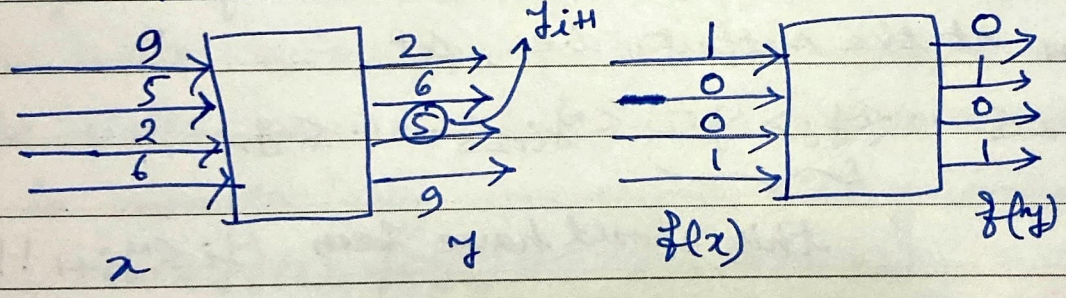
Now, $f(x) \le f(y)$ whenever $x \le y$, and a given network transforms $\langle x_1, x_2, \cdots, x_n \rangle$ int $\langle y_1, y_2, \cdots, y_n \rangle$, then it is easy to see that the network will transform $\langle f(x_1), f(x_2), \cdots, f(x_n) \rangle$ into $\langle f(y_1), f(y_2), \cdots, f(y_n) \rangle$.

If $y_i > y_{i+1}$ for some i, then the monotonic function f takes all numbers $< y_{i+1}$ into '0' and all numbers $> y_{i+1}$ int '1'. With the above monotonic function, we now apply the inputs $f(x_1), f(x_2), \ldots, f(x_n)$ to the network as follows, to get the outputs $f(y_1), f(y_2), \ldots, f(y_n)$ at the output.

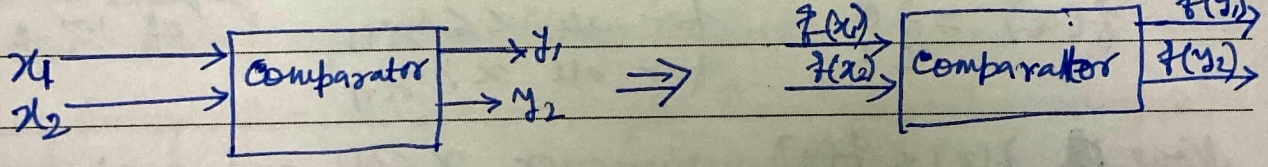


$f(x)$ = applying the f^n on x
$f(y)$ = applying the f^n on y

example:



Note that, any comparator which takes inputs x_1 and x_2 and produces the outputs y_1 and y_2 as follows,



it also produces, because of monotonicity of the function f, the outputs of $f(y_1)$ and $f(y_2)$ when inputs are $f(x_1)$ and $f(x_2)$. That is, instead of the value

y_i, the same line of the network will contain $f(y_i)$ when the inputs are changed from x_1, x_2 to $f(x_1)$ and $f(x_2)$, respectively.

It implies then that given an input sequence of '0's and '1's, it produces an output sequence

0 0 .... 1 0 ... 0 ... , which is not sorted. This contradicts our original assumption that the network can correctly sort any given sequence of '0' and '1's at its input.

This contradiction proves then that the network can sort any arbitrary sequence of input values (not necessarily '0's and '1's) taken from its input domain. [Proved]

Thus, by virtue of the zero-one principle, to test that a network will be able to sort any of the $n!$ permutation of any arbitrarty set of n values at its input, we need to consider only all possible 2^n sequence of '0's and '1's only given at its input and to test whether the network can sort all such sequences of '0's and '1's correctly.

The test space is then reduced from $n!$ permutations values to only 2^n values of '0's and '1's. A direct and useful application of this zero-one principle is in proving the correctness of the Batcher's odd-even merge network.

Batcher's odd-Even Merge Algorithm:

Consider two sorted sequences U and V with the elements of U and V as

$U = \langle u_1, u_2, \ldots, u_m \rangle$

$V = \langle v_1, v_2, \ldots, v_n \rangle$

To merge U and V to have a single sorted sequence, we proceed as follows: (Note that: $u \leq u_2 \leq \ldots \leq u_m$ and $v_1 \leq v_2 \leq \cdots \leq v_n$)

① Take out the odd-numbered values from U, to form a subsequence O_u and also the odd-numbered values from V, to form a subsequences O_v, i.e.,

$O_u = \langle u_1, u_3, u_5, \ldots \rangle$

and $O_v = \langle v_1, v_3, v_5, \ldots \rangle$

② Recursively merge (by calling the same batcher's odd-even merge algorithm) O_u and O_v to form a single sorted sequence $A = \langle a_1, a_2, \cdots \rangle$

③ Take out the even-numbered elements from both U and V to form the subsequences E_u and E_v, respectively.

i.e., $E_u = \langle u_2, u_4, u_6, \ldots \rangle$

$E_v = \langle v_2, v_4, v_6 \ldots \rangle$

④ Merge E_u and E_v by recursive calls to the Batcher's odd-even merge algorithm to form a single sorted sequence $B = \langle b_1, b_2, \cdots \rangle$

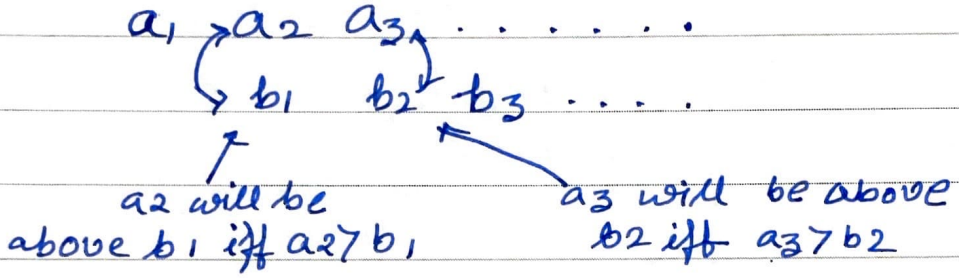
(Note that $a_1 \leq a_2 \leq \cdots$ and $b_1 \leq b_2 \leq \cdots$)

⑤ Write the sequence B below the sequence A by shifting B values one place to the right:

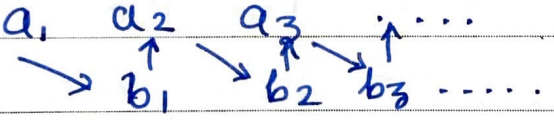
a_1 a_2 a_3 .....

b_1 b_2 b_3 ....

⑥ compare-interchange a_{i+1} and b_i values for $i \geq 1, \ldots$ putting the lower of the two at the bottom and higher of the two at the upper line.



⑦ Read the two sequences in this order after this compare-interchange



The sequence $\langle a_1, b_1, a_2, b_2, a_3, b_3, \cdots \rangle$ will be sorted!!

Example 1:

Let $U = \langle 1, 3, 6, 7, 9 \rangle$

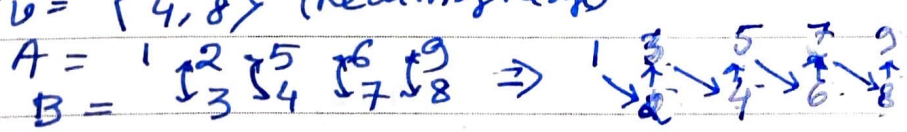
$V = \langle 2, 4, 5, 8 \rangle$

$O_U = \langle 1, 6, 9 \rangle$ ⟹ $A = \langle 1, 2, 5, 6, 9 \rangle$

$O_V = \langle 2, 5 \rangle$ (Recurring Merge)

$E_U = \langle 3, 7 \rangle$ ⟹ $B = \langle 3, 4, 7, 8 \rangle$

$E_V = \langle 4, 8 \rangle$ (Recurring Merge)



∴ sorted sequence $\langle 1, 2, 3, 4, 5, 6, 7, 8, 9 \rangle$

Merging ⟨1, 6, 9⟩ and ⟨2, 5⟩

$O_u = \langle 1, 9 \rangle$, $O_v = \langle 2 \rangle$ $\Longrightarrow$ $A = \langle 1, 2, 9 \rangle$

$E_u = \langle 6 \rangle$, $E_v = \langle 5 \rangle$ $\Longrightarrow$ $B = \langle 5, 6 \rangle$

A = 1 2 9 ⇒ A = 1 5 9 = ⟨1, 2, 5, 6, 9⟩
B = 5 6 B = 2 6

Merging ⟨1, 9⟩ and ⟨2⟩

$O_u = \langle 1 \rangle$, $O_v = \langle 2 \rangle$ $\Longrightarrow$ $A = \langle 1, 2 \rangle$

$E_u = \langle 9 \rangle$, $E_v = \phi$ $\Longrightarrow$ $B = \langle 9 \rangle$

A = 1 2 ⇒ A = 1 9 = ⟨1, 2, 9⟩
B = 9 B =

Example 2:

$U = \langle 3, 4, 6, 7, 9 \rangle$

$V = \langle 1, 2, 5, 8, 10 \rangle$

$O_u = \langle 3, 6, 9 \rangle$, $O_v = \langle 1, 5, 10 \rangle$ $\Longrightarrow$ $A = \langle 1, 3, 5, 6, 9, 10 \rangle$

$E_u = \langle 4, 7 \rangle$, $E_v = \langle 2, 8 \rangle$ $\Longrightarrow$ $B = \langle 2, 4, 7, 8 \rangle$

A = 1 3 5 6 9 10 ⇒
2 4 7 8

Output = ⟨1, 2, 3, 4, 5, 6, 7, 8, 9, 10⟩

Merging ⟨3, 6, 9⟩ and ⟨1, 5, 10⟩

$O_u = \langle 3, 9 \rangle$, $O_v = \langle 1, 10 \rangle$ $\Longrightarrow$ $A = \langle 1, 3, 9, 10 \rangle$

$E_u = \langle 6 \rangle$, $E_v = \langle 5 \rangle$ $\Longrightarrow$ $B = \langle 5, 6 \rangle$

A = 1 3 9 10 ⇒ 1 5 9 10 = ⟨1, 3, 5, 6, 9, 10⟩
B = 5 6 3 6

Merging ⟨3, 9⟩ and ⟨1, 10⟩

$O_u = \langle 3 \rangle$, $O_v = \langle 1 \rangle$ $\Longrightarrow$ $A = \langle 1, 3 \rangle$

$E_u = \langle 9 \rangle$, $E_v = \langle 10 \rangle$ $\Longrightarrow$ $B = \langle 9, 10 \rangle$

A = 1 → 3
B = 9 10

⇒ A = 1 → 2 → 3 → 10
B =

⇒ ⟨1, 3, 9, 10⟩

Merging ⟨4, 7⟩ and ⟨2, 8⟩

$O_u = \langle 4 \rangle$, $O_v = \langle 2 \rangle$ ⟹ $A = \langle 2, 4 \rangle$

$E_u = \langle 7 \rangle$, $E_v = \langle 8 \rangle$ ⟹ $B = \langle 7, 8 \rangle$

A = 2 → 4
B = 7 8

⇒ 2 → 7 → 4 → 8 = ⟨2, 4, 7, 8⟩

Algorithm for Batcher's odd-even Merging:

Input: Two sorted sequences U and V given as

$U = \langle u_1, u_2, \cdots, u_m \rangle$ and $V = \langle v_1, v_2, \cdots, v_n \rangle$

i.e., $|U| = m$ and $|V| = n$.

Output: sorted sequence S consisting of $m+n$ elements taken from U and V, $S = \langle s_1, s_2, s_3, \cdots, s_{m+n} \rangle$.

Batcher(U, V, S):

step 1: If $|U| = 0$ or $|V| = 0$ then the algorithm terminates with $S = V$ or $S = U$, respectively.

step 2: If $|U| = 1$ and $|V| = 1$, then compare-interchange $(v_1 : u_1)$ (i.e., if $(u_1 > v_1)$ then swap (u_1, v_1); $S = \langle v_1, u_1 \rangle$ and then stop.

step 3: Form the odd sequence O_u and O_v from U and V, respectively.

step 4: Form the even sequence E_u and E_v, from U and V, respectively.

step 5: Batcher (O_u, O_v, A);

where $A = \langle a_1, a_2, \cdots, a_{\lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil} \rangle$

step 6: Batcher (E_u, E_v, B);

where $B = \langle b_1, b_2, b_3, \ldots, b_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor} \rangle$

step 7: compare-interchange

$(b_1 : a_2), (b_2 : a_3), (b_3 : a_4), \ldots, (b_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor} : a^*)$

[where $a^* = a_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor + 1}$ which does not exist if both m & n are even] to generate the sequence

$S = \langle a_1, b_1, a_2, b_2, a_3, \ldots, b_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor}, a^*, a^{**} \rangle$

where $a^{**} = a_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor + 2}$, which does not exist unless both m & n are odd.

return;

$U = \langle u_1, u_2, u_3, \ldots, u_m \rangle$ $V = \langle v_1, v_2, v_3, \ldots, v_n \rangle$

$O_u = \langle u_1, u_3, \ldots, u_{2\lceil \frac{m}{2} \rceil - 1} \rangle$ $O_v = \langle v_1, v_3, \ldots, v_{2\lceil \frac{n}{2} \rceil - 1} \rangle$

$\therefore A = \langle a_1, a_2, a_3, \ldots, a_{\lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil} \rangle$

$E_u = \langle u_2, u_4, u_6, \ldots, a_{2\lfloor \frac{m}{2} \rfloor} \rangle$ $E_v = \langle v_2, v_4, v_6, \ldots, b_{2\lfloor \frac{n}{2} \rfloor} \rangle$

$B = \langle b_1, b_2, b_3, \ldots, b_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor} \rangle$

Explanation of step 7:

$|O_u| = |\lceil \frac{m}{2} \rceil|$, $|O_v| = |\lceil \frac{n}{2} \rceil|$ $\therefore |A| = |\lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil|$

$|E_u| = |\lfloor \frac{m}{2} \rfloor|$, $|E_v| = |\lfloor \frac{n}{2} \rfloor|$ $\therefore |B| = |\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor|$

case 1: Both m and n are even

i.e., $\lceil \frac{m}{2} \rceil = \lfloor \frac{m}{2} \rfloor$ and $\lceil \frac{n}{2} \rceil = \lfloor \frac{n}{2} \rfloor$ $\therefore |A| = |B|$

a_1 a_2 a_3 a_4 ... $a_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor - 1}$ $a_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor}$

b_1 b_2 b_3 ... $b_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor - 2}$ $b_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor - 1}$ $b_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor}$

So, a^* does not exist.

case 2: Both m and n are odd

i.e., $\lceil \frac{m}{2} \rceil = \lfloor \frac{m}{2} \rfloor + 1$ and $\lceil \frac{n}{2} \rceil = \lfloor \frac{n}{2} \rfloor + 1$

$a_1\ a_2\ a_3\ a_4 \ldots\ldots a_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor}\ a_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor + 1}\ a_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor + 2}$

$b_1\ b_2\ b_3 \ldots\ldots b_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor - 1}\ b_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor}$

So, $a^{**} = a_{\lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil} = a_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor + 2}$

case 3: only one of m and n is odd, the other is even

i.e., $\lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil = \lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor + 1$ [$\because$ either m or n is odd]

$a_1\ a_2\ a_3\ a_4 \ldots\ldots a_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor}\ a_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor + 1}$

$b_1\ b_2\ b_3 \ldots\ldots b_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor - 1}\ b_{\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor}$

So, $a^{*} = \lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor + 1$

Proof of Correctness:

The only thing to prove is that step 7 works correctly. Based on the input values of 0's and 1's only both the sequence of A and B will look like:

$A = \langle 0,0,0,\ldots\ldots,0,1,1,1,\ldots,1 \rangle$

$B = \langle 0,0,0,\ldots\ldots,0,1,1,1,\ldots,1 \rangle$

Let, there are l and k 0 in U and V respectively.

$U = \langle \overbrace{000\ldots00}^{l} 111\ldots11 \rangle$ $V = \langle \overbrace{000\ldots00}^{k} 111\ldots11 \rangle$

$O_U = \langle \overbrace{00\ldots0}^{\lceil l/2 \rceil} 111\ldots1 \rangle$ $O_V = \langle \overbrace{00\ldots0}^{\lceil k/2 \rceil} 111\ldots1 \rangle$

$A = \langle \overbrace{000\ldots00}^{\lceil l/2 \rceil + \lceil k/2 \rceil} 111\ldots1 \rangle$

$E_U = \langle \overbrace{000\ldots00}^{\lfloor l/2 \rfloor} 111\ldots1 \rangle$ $E_V = \langle \overbrace{000\ldots00}^{\lfloor k/2 \rfloor} 111\ldots1 \rangle$

$B = \langle \overbrace{000\ldots00}^{\lfloor l/2 \rfloor + \lfloor k/2 \rfloor} 111\ldots1 \rangle$

$\therefore \; (\lceil \ell/2 \rceil + \lceil k/2 \rceil) - (\lfloor \ell/2 \rfloor + \lfloor k/2 \rfloor) = 0, 1, 2$

Example: $U = \langle \overset{1}{0}\, 0\, \overset{3}{0}\, 0\, \overset{5}{0}\, 1\, \overset{7}{1}\, 1 \rangle$

$V = \langle \underset{1}{0}\, 0\, \underset{3}{0}\, 0\, \underset{5}{0}\, 1\, 1 \rangle$

$O_u = \langle 0\,0\,0\,1 \rangle$, $(\ell = 5)$, $\lceil \ell/2 \rceil = 3$

$O_v = \langle 0\,0\,1 \rangle$, $(k = 4)$, $\lceil k/2 \rceil = 2$

On the other hand, the number of 0's in B will be $\lfloor \ell/2 \rfloor + \lfloor k/2 \rfloor$ $U = \langle \overset{2}{0}\,0\,\overset{4}{0}\,0\,\overset{6}{0}\,1\,\overset{8}{1}\,1 \rangle$ $V = \langle \overset{2}{0}\,0\,\overset{4}{0}\,0\,\overset{6}{1}\,1 \rangle$

$E_u = \langle 0\,0\,1\,1 \rangle$, $\lfloor \ell/2 \rfloor = \lfloor \frac{5}{2} \rfloor = 2$; $E_v = \langle 0\,0\,1 \rangle$, $\lfloor k/2 \rfloor = \lfloor \frac{4}{2} \rfloor = 2$

So, when we compare-interchange elements of A and B in step 7, this looks like:

① A = 0 0 0 . . . . 0 1 1 1

B = 0 0 . . . . 0 0 1 1

Both ℓ and k are even, difference in # 0's = 0

② A = 0 0 0 . . . . 0 1 1 1

B = 0 0 . . . . 0 1 1 1

When one of ℓ and k is odd and the other is even, difference in # 0's = 1

③ A = 0 0 0 0 . . . . . 0 0 1 1 1

B = 0 0 0 . . . . 0 1 1 1 1

Both ℓ and k are odd, difference in # 0's = 2

In each of case ① and ② above, the elements in A and B are already sorted – no interchange is necessary in the required order of outputting from A and B.

Only in case ③, we have the situation

0 0 0 0 . . . . 0 0 1 1 . . . 1
0 0 0 . . . . 0 1 1 1 . . . 1

Interchange ⇓

0 0 0 0 . . . . 0 1 1 1 . . . 1
0 0 0 . . . . 0 0 1 1 . . . 1

Hence, the sequence becomes sorted. Hence this proves that the algorithm (Batcher) produces a sorted sequence by merging the input sequences U and V.

Number of comparators required for odd-even merging

① Let, m and n both are even, then a^* does not exist.

$m = 2p$, $n = 2q$. # no. of comparator $= \lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor - 1$

$\therefore \lfloor \frac{2p}{2} \rfloor + \lfloor \frac{2q}{2} \rfloor - 1$

$= p + q - 1$

[ex: a_1 (a_2 b_1) (a_3 b_2) b_3]

and $\lfloor \frac{m+n-1}{2} \rfloor = \lfloor \frac{2p+2q-1}{2} \rfloor = \lfloor p + q - \frac{1}{2} \rfloor = p + q - 1$

② Only one is odd, then a^* i.e., $\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor + 1$ exists.

$m = 2p+1$, $n = 2q$ - # no. of comparator $= \lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor$

$\therefore \lfloor m/2 \rfloor + \lfloor n/2 \rfloor = p + q$

and $\lfloor \frac{m+n-1}{2} \rfloor = \lfloor \frac{2p+1+2q-1}{2} \rfloor = \lfloor \frac{2p+2q}{2} \rfloor = p + q$

③ m and n both are odd, then a^{**} i.e., $\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor + 2$ exists.

$m = 2p+1$, $n = 2q+1$ # no. of comparator $= \lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor$

$\therefore \lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor = p + q$

and $\lfloor \frac{m+n-1}{2} \rfloor = \lfloor \frac{2p+1+2q+1-1}{2} \rfloor = \lfloor \frac{2p+2q+1}{2} \rfloor = \lfloor p + q + \frac{1}{2} \rfloor$

$= p + q$

mn for $mn \leq 1$

$C_m(m,n) = C_m(\lceil m/2 \rceil, \lceil n/2 \rceil) + C_m(\lfloor m/2 \rfloor + \lfloor \frac{n}{2} \rfloor) + \lfloor \frac{m+n-1}{2} \rfloor$

Let, $m = 2^t$ and $n = 2^t$ then $\lceil m/2 \rceil = \lfloor m/2 \rfloor$ and $\lceil n/2 \rceil = \lfloor n/2 \rfloor$

$\therefore C_m(2^t, 2^t) = 2C_m(2^{t-1}, 2^{t-1}) + \lfloor \frac{2 \cdot 2^t - 1}{2} \rfloor$

$\Rightarrow C_m(2^t, 2^t) = 2C_m(2^{t-1}, 2^{t-1}) + \lfloor 2^t - \frac{1}{2} \rfloor$

$\Rightarrow C_m(2^t, 2^t) = 2C_m(2^{t-1}, 2^{t-1}) + 2^t - 1$

using Back substitution method:

$C_m(2^t, 2^t) = 2C_m(2^{t-1}, 2^{t-1}) + 2^t - 1$

$= 2[2C_m(2^{t-2}, 2^{t-2}) + 2^{t-1} - 1] + 2^t - 1$

$= 2^2 C_m(2^{t-2}, 2^{t-2}) + 2 \cdot 2^{t-1} - 2 + 2^t - 1$

$= 2^2 C_m(2^{t-2}, 2^{t-2}) + 2 \cdot 2^t - (2+1)$

$= 2^2[2C_m(2^{t-3}, 2^{t-3}) + 2^{t-2} - 1] + 2 \cdot 2^t - (2+1)$

$= 2^3 C_m(2^{t-3}, 2^{t-3}) + 3 \cdot 2^t - (2^2 + 2 + 1)$

Let t steps are there. So after t steps

$C_m(2^t, 2^t) = 2^t C_m(2^{t-t}, 2^{t-t}) + t \cdot 2^t - (2^{t-1} + \cdots + 2^2 + 2 + 1)$

$= 2^t C_m(2^0, 2^0) + t \cdot 2^t - (2^{t-1} + \cdots + 2^2 + 2 + 1)$

$\because C_m(1,1) = 1 \therefore = 2^t + t \cdot 2^t - (2^{t-1} + \cdots + 2^2 + 2 + 1)$

$= (t+1)2^t - (2^t - 1)$

$= t \cdot 2^t + 2^t - 2^t + 1 = t \cdot 2^t + 1$

Number of comparators required for sorting 2^t elements

$C_S(2^t) = C_S(2^{t-1}) + C_S(2^{t-1}) + C_m(2^{t-1}, 2^{t-1})$

$\Rightarrow C_S(2^t) = 2C_S(2^{t-1}) + C_m(2^{t-1}, 2^{t-1})$

$= 2C_S(2^{t-1}) + (t-1) \cdot 2^{t-1} + 1$

dividing both side by 2^t, we get:

$$\frac{C_s(2^t)}{2^t} = \frac{2C_s(2^{t-1})}{2^t} + \frac{(t-1)\cdot 2^{t-1}}{2^t} + \frac{1}{2^t}$$

$$\frac{C_s(2^t)}{2^t} = \frac{C_s(2^{t-1})}{2^{t-1}} + \frac{(t-1)}{2} + \frac{1}{2^t}$$

Let, $\frac{C_s(2^t)}{2^t} = x_t$

$\therefore x_t = x_{t-1} + \frac{(t-1)}{2} + \frac{1}{2^t}$

Now base case $x_1 = \frac{C_s(2^1)}{2^1} = \frac{1}{2}$ [∵ for two elements single comparator is required]

using back substitution:

$$x_t = x_{t-1} + \frac{(t-1)}{2} + \frac{1}{2^t}$$

$$= x_{t-2} + \frac{(t-2)}{2} + \frac{1}{2^{t-1}} + \frac{(t-1)}{2} + \frac{1}{2^t}$$

$$= x_{t-2} + \frac{(t-2)}{2} + \frac{(t-1)}{2} + \frac{1}{2^{t-1}} + \frac{1}{2^t}$$

$$= x_{t-3} + \frac{(t-3)}{2} + \frac{(t-2)}{2} + \frac{(t-1)}{2} + \frac{1}{2^{t-2}} + \frac{1}{2^{t-1}} + \frac{1}{2^t}$$

Let $(t-1)$ steps are there;

$$x_t = x_{t-(t-1)} + \frac{(t-(t-1))}{2} + \cdots + \frac{(t-3)}{2} + \frac{(t-2)}{2} + \frac{(t-1)}{2} + \frac{1}{2^{t-(t-2)}} + \cdots + \frac{1}{2^{t-1}} + \frac{1}{2^t}$$

$$= x_1 + \left[\frac{1}{2} + \cdots + \frac{(t-3)}{2} + \frac{(t-2)}{2} + \frac{(t-1)}{2}\right] + \left[\frac{1}{2^2} + \cdots + \frac{1}{2^{t-1}} + \frac{1}{2^t}\right]$$

$$= \frac{t(t-1)}{4} + \left[\frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^{t-1}} + \frac{1}{2^t}\right]$$

$$= \frac{t(t-1)}{4} + 1 - \frac{1}{2^t}$$

$$= \frac{t^2-t}{4} + 1 - \frac{1}{2^t}$$

$$\therefore \frac{C_s(2^t)}{2^t} = \frac{t^2-t}{4} + 1 - \frac{1}{2^t} \Rightarrow C_s(2^t) = \frac{(t^2-t)2^t}{4} + 2^t - 1$$

$\therefore C_S(2^t) = (t^2-t)\cdot 2^{t-2} + 2^t - 1$

$= 2^{t-2}(t^2-t+2^2) - 1$

$= \frac{2^t}{2^2}(t^2-t+4) - 1$

$\because m=n=2^t \quad \therefore C_S(2^t) = \frac{n}{2^2}(\log^2 n - \log n + 1) - 1$

$= \frac{n}{4}\log^2 n - \frac{n\log n}{4} + \frac{n}{2^2} - 1 = O(n\log^2 n)$

Merging time for 2^t elements:

$T_B(m,n) = \max\left[T_B(\lceil m/2\rceil, \lceil n/2\rceil), T_B(\lfloor m/2\rfloor, \lfloor n/2\rfloor)\right] + 1$

$T_B(m,0) = T_B(0,n) = 0$ and $T_B(1,1) = 1$

$T_B(m, n+1) \geq T_B(m,n)$

$\Rightarrow T_B(m,n) = 1 + \lceil \log(\max(m,n)) \rceil$

Special case $m=n=2^t \quad \therefore T_B(2^t, 2^t) = 1+t$ [as $m=n=t$ $\therefore t=\log m = \log n$]

Sorting time for 2^t elements:

$T_S(2^t) = T_S(2^{t-1}) + T_B(2^{t-1}, 2^{t-1})$

$= T_S(2^{t-1}) + t + t +$

$= T_S(2^{t-1}) + t$

$= T_S(2^{t-2}) + t + (t-1)$

Let $(t-1)$ steps are there;

$\Rightarrow T_S(2^t) = T_S(2^{t-t+1}) + 2+3+\cdots+(t-1)+t$

$= T_S(2^1) + 2+3+\cdots+(t-1)+t$

$\because T_S(2^1) = 1 \quad \therefore T_S(2^t) = 1+2+3+\cdots+(t-1)+t$

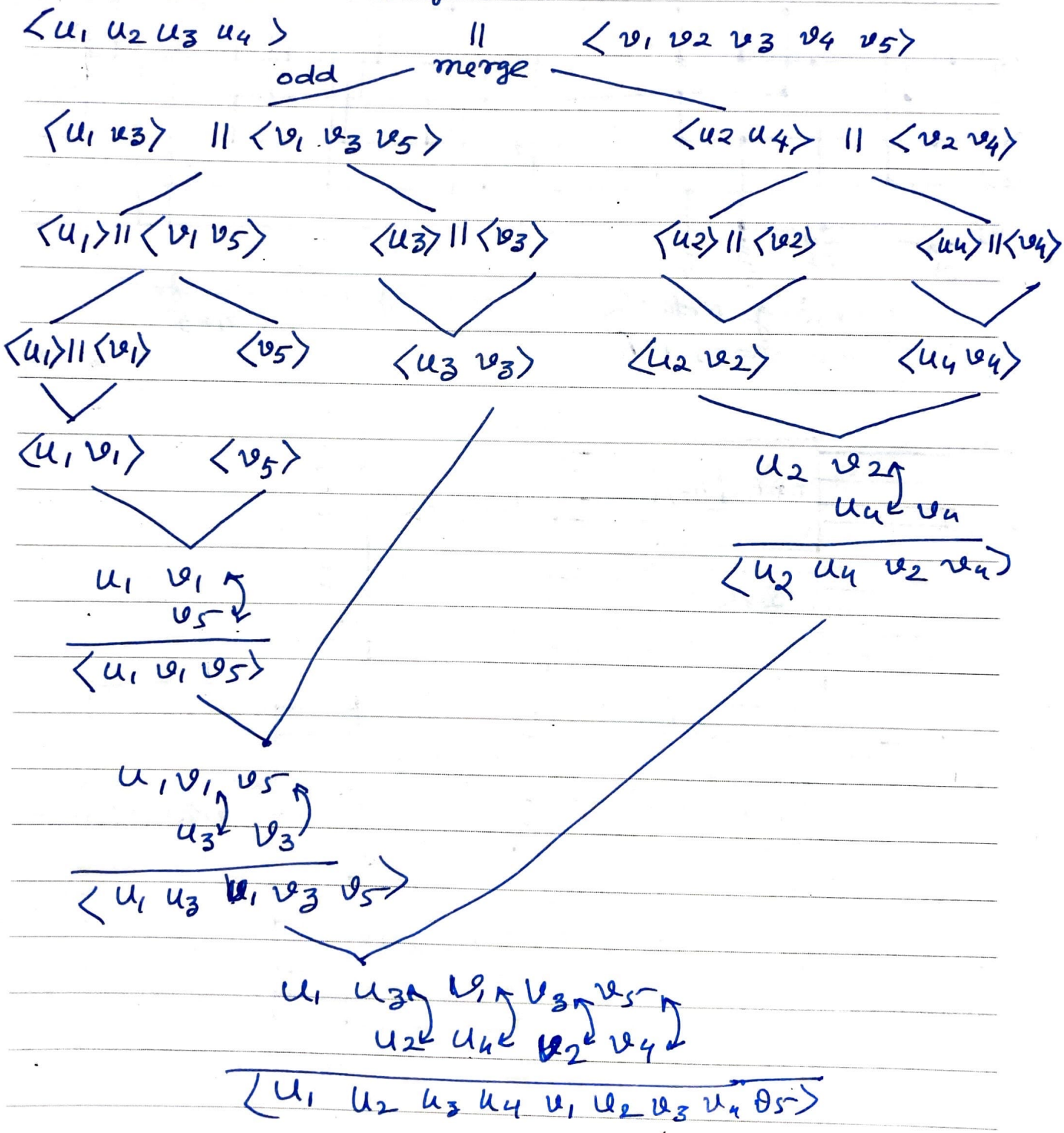
$= t(t+1)/2 = O(t^2)$

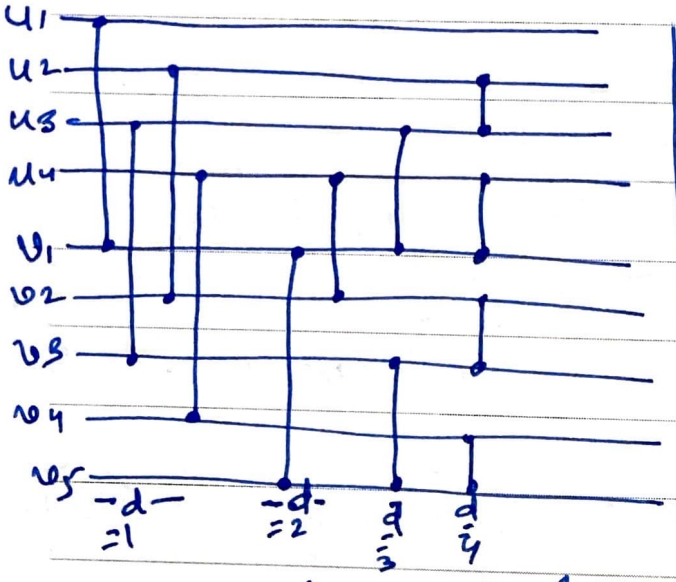
$= O(\log^2 n)$ as $n = 2^t$.

Example Network:

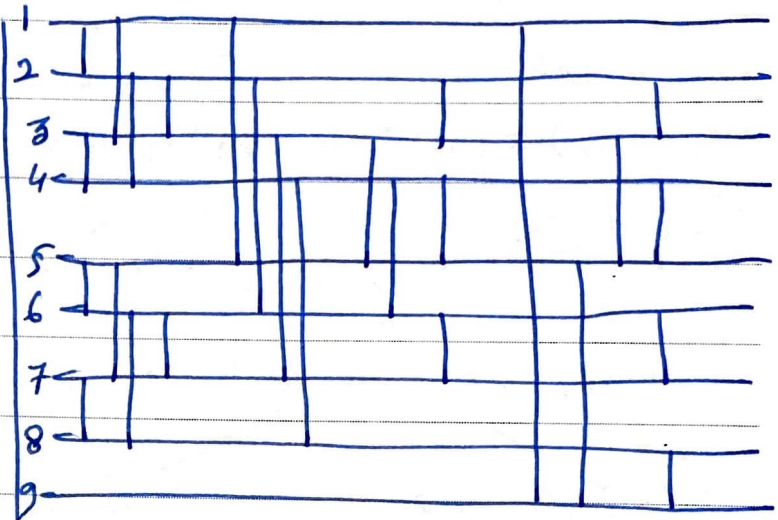
Let $m=4$, $n=5$

$(m,n) = (4,5)$ merging Network

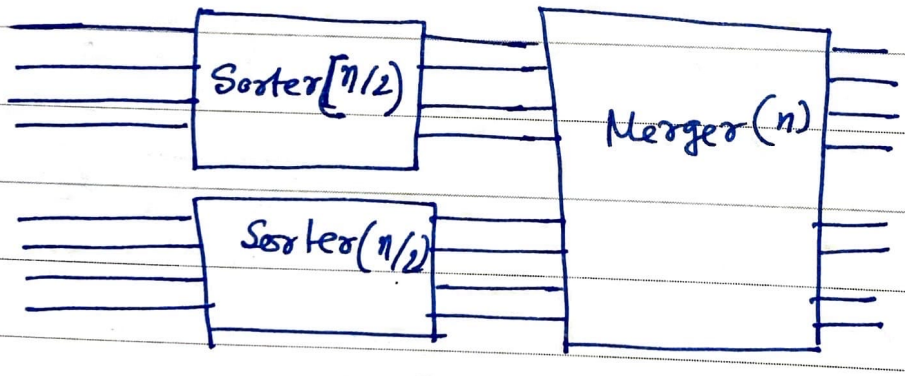


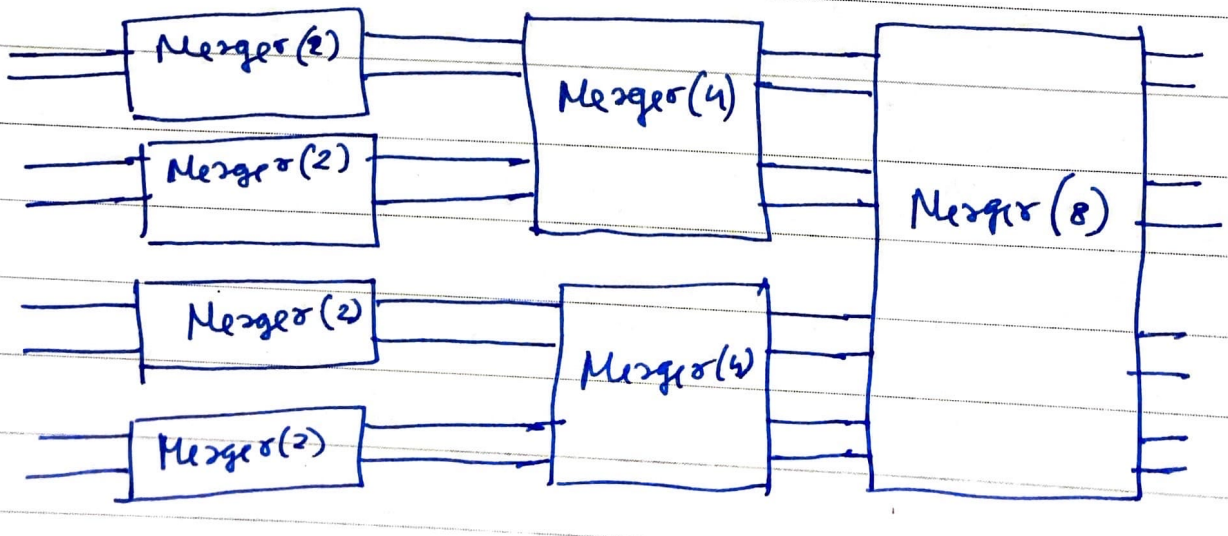


Merging Network for $m=4, n=5$



Sorting Network for $m=4, n=5$





Algorithm Batcher_Sort (i, j, S):

/* S is the sorted sequence */

if i == j then return S with this element;

if j == i+1 then compare-interchange the values on lines i and j and return S with the resulting values on lines i and j;

Batcher_Sort (i, $\left[\frac{j-i+1}{2}\right]$, A);

Batcher_Sort ($\left\lfloor\frac{j-i+1}{2}\right\rfloor$ + 1, j, B);

Batcher (A, B, S);

/* A and B are merged to S by Batcher's odd-even merge network */