Sorting Algorithms

R. K. Ghosh

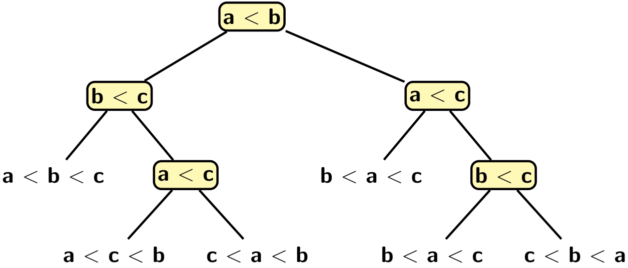
IIT Kanpur

Advance Sorting Algorithms

Lower Bound for Sorting

- In a comparison based sorting, one of the five tests are made: $a_i < a_j$, $a_i \leq a_j$, $a_i = a_j$, $a_i \geq a_j$, $a_i > a_j$
- WLOG assume all inputs are distinct.
- So, $a_i = a_j$ is useless.
- All other four comparisons produce identical results on relative ordering.
- So, only comparison of the form $a_i \leq a_j$ is made.

Decision Tree Model for Sorting 3 Elements



Lower Bound for Sorting

- In a decision tree each internal node represents a comparison.
- Left subtree of an internal node represents all subsequent comparisons when $a_i \leq a_j$.
- Right subtree of an internal node represents all subsequent comparisons when $a_i > a_j$.
- Each leaf node represents a sorting order.
- So tracing a path in decision tree amounts to finding correct permutation for sorting the list of elements.
- Since $n!$ permutations are possible, a decision tree should have $n!$ leaves, one for each permutation.

Lower Bound for Sorting

- A binary tree of height h has no more than 2^h nodes.
- So, the minimum height of the decision tree for sorting n elements should have a height $\log(n!)$

$$
\begin{aligned}
\log n! &= \sum_{i=2}^{n} \log i \\
&= \sum_{2}^{n/2-1} \log i + \sum_{n/2}^{n} \log i \\
&\geq \frac{n}{2} \log(n/2) \\
&= \Omega(n \log n)
\end{aligned}
$$

Quick Sort Average Case Analysis

- All permutations are equally likely.
- Then first pivot is a random element from $1, \ldots, n$
- Pivots are used at all recursive levels and are random elements.
- The recurrence formula for the number of comparisons:

$$T(n) = n - 1 + \frac{1}{n} \sum_{1 \leq k \leq n} (T(i) + T(n - i - 1)), \text{ where } n \geq 2.$$

- Initially pivot is moved out of the way by placing it either at the end or at the beginning.
- One comparison with each of remaining elements: $n - 1$.

Quick Sort Average Case Analysis

$$T(n) = n - 1 + \frac{2}{n} \sum_{1 \leq k \leq n} T(i-1)$$

Now multiply both sides by n and substract $T(n-1)$ from $T(n)$

$$nT(n) - (n-1)T(n-1) = n(n-1) - n(n-2) + 2T(n-1)$$

Rearrange terms to simplify

$$nT(n) = 2n - 2 + (n+1)T(n-1)$$

Now try to solve the above recurrence.

Quick Sort Average Case Analysis

$$
\begin{aligned}
\frac{T(n)}{n+1} &= \frac{2}{n+1} - \frac{2}{n(n+1)} + \frac{T(n-1)}{n} \\
&\leq \frac{T(n-1)}{n} + \frac{2}{n+1}, \text{ neglecting -ve term} \\
&\leq \frac{T(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1}, \text{ unfolding } T(n-1) \\
&= \vdots \text{ (continuing unfolding of recursion)} \\
&\leq \frac{T(1)}{2} + 2 \sum_{2 \leq k \leq n} \frac{1}{k+1} \\
&\leq 2 \sum_{1 \leq k \leq n} \frac{1}{k}
\end{aligned}
$$

Quick Sort Average Case Analysis

Approximate the sum by integral $\int_1^n \frac{1}{x} dx$, then

$$\frac{T(n)}{n+1} = 2\ln(n)$$
$$T(n) = 2(n+1)\ln(n) \approx 2(n+1)1.39\log_2 n$$

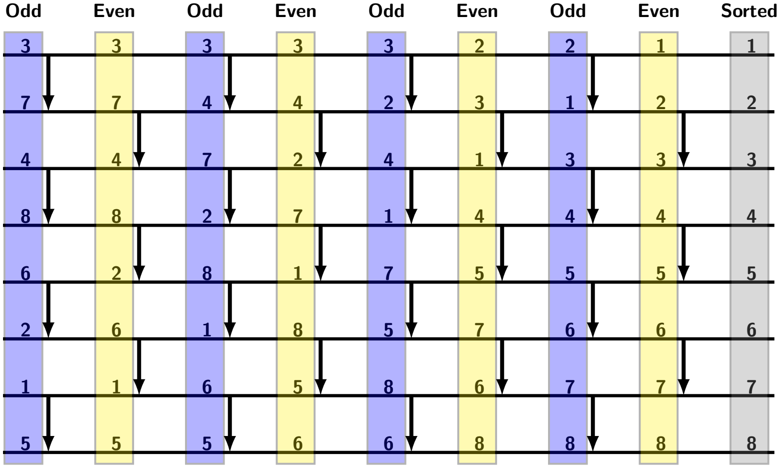
Batcher's Odd Even Sort

- Built on the idea of bubble sort.
- In bubble sort, the heaviest element sinks down.
- Starting with the first element, in each pass an adjacent pair of elements are compared.
- In odd-even sort, there are two distinct phases: Odd and even.
- In an odd phase all odd elements are compared with the adjacent elements.
  - If the pair is in wrong order the members are swapped.
- The same is repeated with the even elements for an even phase.
- Alternates between (odd, even) and (even, odd) comparison phases until list is sorted.

Batcher's Odd Even Sort

```
OddEvenSort(n) {
    for (i = 1; i <= n; i++) {
        if (odd(i)) {
            for (j = 0; j <= n/2-1; j++) {
                compareExchange(a_{2j+1}, a_{2j+2});
            }
        }
        if (even(i)) {
            for (j = 1; j <= n/2-1; j++) {
                compareExchange(a_{2j}, a_{2j+1});
            }
        }
    }
}
```

Odd-Even Sort Example



Odd-Even Sort Complexity

- After n phases of odd-even exchanges, list is sorted.
- Each phase requires at most $n/2$ comparisons.
- So complexity is O(n^2).
- But it is easily parallelized.
- Parallel time complexity is O(n).

Bitonic Sequence

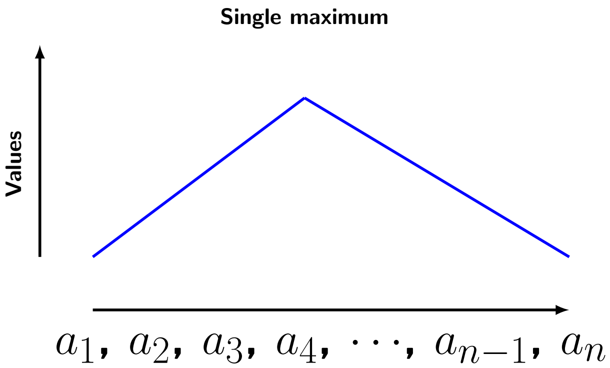
- A bitonic sequence is a sequence that has no more than one local maximum and no more than one local minimum.
- Either monotonically increases then monotonically decreases.
- Or, else monotonically decreases then monotonically increases.

Bitonic Sequence

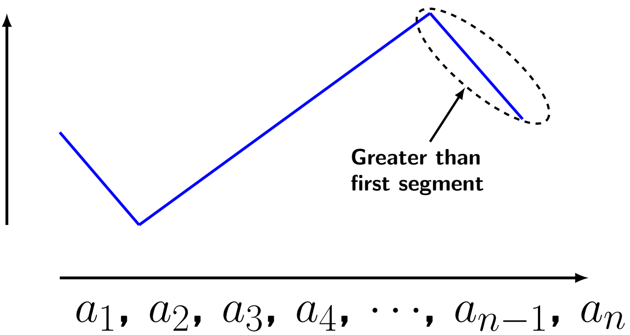
- Thus a sequence $B = \{b_1, b_2, b_3, \ldots, b_n\}$ is bitonic if and only if:
  1. $b_1 \leq b_2 \leq \cdots \leq b_k \geq b_{k+1} \geq \cdots \geq b_n$ for some $1 < k < n$.
  2. Or, $b_1 \geq b_2 \geq \cdots \geq b_k \geq b_{k+1} \leq \cdots \leq b_n$ for some $1 < k < n$,
  3. Or, if the property can be achieved by moving some elements circularly (left or right).
- Example of bitonic sequences are:

  seq 1: $< 1\ 4\ 6\ 8\ 3\ 2 >$ seq 2: $< 9\ 8\ 3\ 2\ 4\ 6 >$

Bitonic Sequence



Single maximum & single minimum



Characteristics of Bitonic Sequence

- Let B be a bitonic sequence of size n.
- Define $L(b)$ and $R(b)$ as follows.

$$
\begin{aligned}
L(b) &= \min(b_1, b_{1+n/2}), \min(b_2, b_{2+n/2}) \ldots, \min(b_{n/2}, b_n) \\
R(b) &= \max(b_1, b_{1+n/2}), \max(b_2, x_{2+n/2}) \ldots, \max(b_{n/2}, b_n)
\end{aligned}
$$

- Each element of $L(b)$ is less than every element of $R(b)$.
- Furthermore, $L(b)$ and $R(b)$ are bitonic sequences.

Zero On Principle

First let us prove the following result.

Lemma

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

Proof.

- A single max-comparator with two inputs x and y outputs $x' = \min(x, y)$ and $y' = \max(x, y)$ as shown on the left.

□

Zero On Principle

Proof (contd.)

- Let $f(x) = f(y)$, so

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

- Therefore,

- Then the claim trivially holds.

Zero On Principle

Proof(contd.)

- Now let $f(x) < f(y)$, then as f is monotonically increasing:
  - $\min(f(x), f(y)) = f(\min(x, y)) = f(x)$ and
  - $\max(f(x), f(y)) = f(\max(x, y)) = f(y)$
- So, for inputs $f(x), f(y)$, the output is $f(x), f(y)$ for
- The same arguments holds true for the case $f(x) > f(y)$, for inputs $f(x), f(y)$ output will be $f(y), f(x)$.

$\square$

Zero On Principle

Proof(contd.)

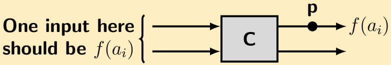
- Now apply induction to prove the claim that if the sorting network gets inputs $a_1, \ldots a_n$ (wire i carries input a_i), then for inputs $f(a_1), f(a_2), \ldots, f(a_n)$ the wire i will carry input $f(a_i)$.
- For depth 0, this is trivially true.
- Assume that it holds for all points in our circuits of depth at most i.

$\square$

Zero On Principle

Proof(contd.)

- Now consider a wire p in the circuit at dept $i+1$.



- Let the wire p belong to the output of a comparator C which carries a_i at depth $i+1$ for initial input sequence $\langle a_1, a_2, \ldots, a_n \rangle$.
- One of the input wires of C are at depth i must have carried a_i.

□

Zero On Principle

Proof(contd.)

- Now by induction hypothesis, if an input wire of C carried a_i for the input $\langle a_1, a_2, \ldots, a_n\rangle$, the same wire should carry $f(a_i)$ for the inputs $f(a_1), f(a_2), \ldots, f(a_n)$, where $f()$ is monotonically increasing.
- So, if the output wire p of C carries a_i for the input a_i at one of the input wires of C, then the same output wire must carry $f(a_i)$ when $f(a_i)$ is on the same input wire of C, as the claim holds a for single comparator.

$\square$

Zero On Principle

Zero-One Principle

If a comparison network sorts all 2^n binary strings of length n then it correctly sorts all sequences.

Proof.

- Let the output $b_1, b_2, \ldots, b_n$ of $a_1, a_2, \ldots, a_n$ be incorrect.
- Let $a_i < a_k$ be the pair which appearing in incorrect orders in the output.
- Let us define:

$$f(x) = \begin{cases} 0, \text{ if } x \leq a_i \\ 1, \text{ if } x > a_i \end{cases}$$

$\square$

Zero On Principle

Proof(contd.)

- By our lemma, for the input $\langle f(a_1), \ldots, f(a_n)\rangle$ the circuit will output $\langle(f(b_1), \ldots, f(b_n))\rangle$ if it outputs $b_1, b_2, \ldots b_1$ for input $a_1, a_2, \ldots, a_n$.
- By assumption the output sequence is of the form:

$$000\ldots ???f(a_k)???\ldots ???f(a_i)???\ldots 111$$

- The above sequence by definition of $f(.)$ is of the form:

$$000\ldots ???1???\ldots ???0???\ldots 111$$

- So, the sorting network cannot sort a binary string correctly.

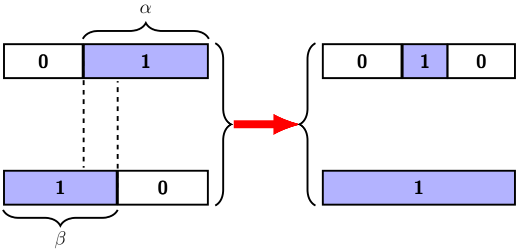
□

Half Cleaner

- A half cleaner compares inputs b_i and $b_{i+n/2}$ and creates $L(b)$ and $R(b)$.
- Let us consider zero-one bitonic sequences: $0^i1^j0^k$ or $1^i0^j1^k$, for $i \geq 0$, $j \geq 0$, and $k \geq 0$.
- If the input is of the form that all 1s occur completely to the left or completely to the right of $n/2$-partition, then the claim is obviously true.
- So, assume that it straddles the $n/2$-partition boundary.

Half Cleaner

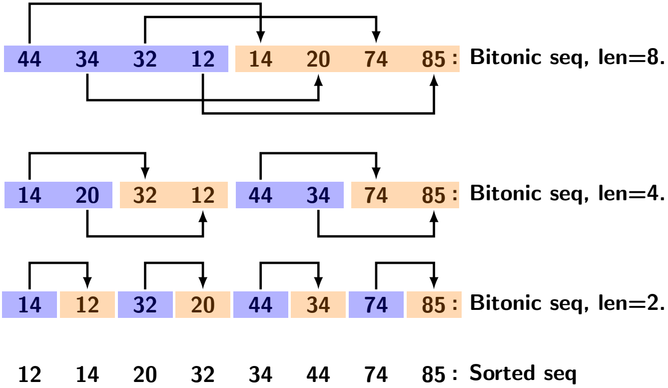
- Let 1s occur from $n/2 - \alpha$ to $n/2 + \beta$, which implies two possibilities:
  - $n/2 - \alpha \geq \beta$ or
  - $n/2 - \alpha < \beta$.
- If $n/2 - \alpha \geq \beta$ then $R(b)$ will consist of all 1s.
- If $n/2 - \alpha < \beta$ then $L(b)$ will consist of all 0s.



Bitonic Merge Algorithm

- Assume length of the sequence is a power of 2.
- If sequence is of length 2^0 no-operation.
- Otherwise perform following steps repeatedly until $n = 2$:
  - Split list of n elements into two list of size $n/2$.
  - Compare and exchange each item of first list with corresponding item of second list.

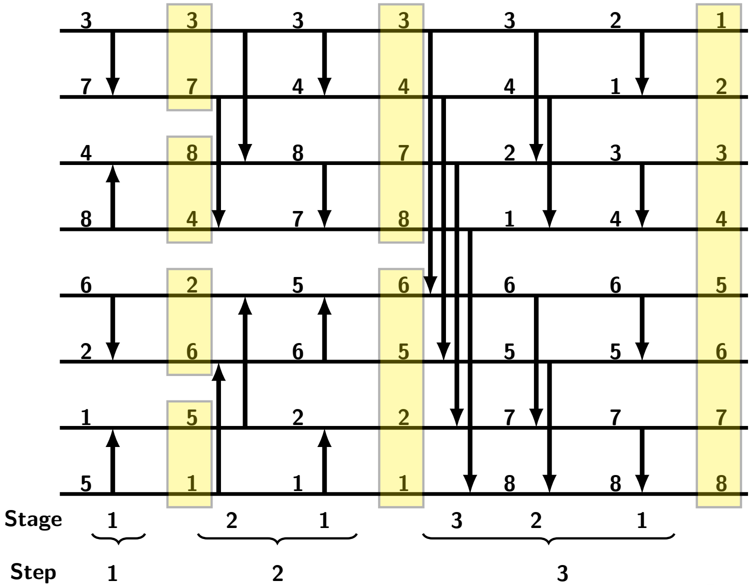
Bitonic Merge Example



Batcher's Bitonic Sort

- Every pair of elements are compared, and alternate pairs are sorted in respectively in ascending and descending manner.
- So, in next step we get bitonic sequence of length 4.
- Now every adjacent pair of 4-element bitonic sequences are merged to produce 8-element bitonic sequences.

Batcher's Bitonic Sort



Complexity of Bitonic Sort

- To form a sorted sequence of length n from two sorted sequences of length $n/2$ are necessary, and requires $\log n$ comparisons.
- So, the recurrence relation is:

$$T(n) = T(n/2) + O(\log n)$$

Solving the recurrence with base case of $T(2) = 1$, we have

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

Since each merging step requires $n/2$ comparisons the total time is O($n \log^2 n$) time.

Summary

- Most common sorting algorithms: insertion sort, bubble sort, merge sort, quick sort, etc., are covered earlier in ESC 101A.
- Heap sort was covered in binary heaps.
- Here we learnt about lower bound of sorting algorithms,
- Odd even merge sort which is essentially derived from idea of buble sort.
- For comparison based sorting, sorting of 0s and 1s is as difficult as sorting of other numbers.
- Bitonic merge and bitonic sorting are also covered.