Correcting Sorted Sequences in a Single Hop Radio Network

Marcin Kik

Wrocław University of Technology

Poland

FCT 2009

Model of computation

Radio network:

- n stations $s_0, \ldots, s_{n-1}$ communicating by radio messages

Model of computation

Radio network:

- n stations $s_0, \ldots, s_{n-1}$ communicating by radio messages
- single-hop (each station within the range of any other station)

Model of computation

Radio network:

- n stations $s_0, \ldots, s_{n-1}$ communicating by radio messages
- single-hop (each station within the range of any other station)
- synchronized (time is divided into *slots*)

Model of computation

Radio network:

- n stations $s_0, \ldots, s_{n-1}$ communicating by radio messages
- single-hop (each station within the range of any other station)
- synchronized (time is divided into *slots*)
- single *channel* (in a single slot at most one message)

Model of computation

Radio network:

- n stations $s_0, \ldots, s_{n-1}$ communicating by radio messages
- single-hop (each station within the range of any other station)
- synchronized (time is divided into *slots*)
- single *channel* (in a single slot at most one message)
- single message contains either a single key or a $\lceil \lg_2 n \rceil$-bit integer

Model of computation

Radio network:

- n stations $s_0, \ldots, s_{n-1}$ communicating by radio messages
- single-hop (each station within the range of any other station)
- synchronized (time is divided into *slots*)
- single *channel* (in a single slot at most one message)
- single message contains either a single key or a $\lceil \lg_2 n \rceil$-bit integer
- broadcasting/listening in a single time slot requires a unit of energetic cost

Model of computation

Radio network:

- n stations $s_0, \ldots, s_{n-1}$ communicating by radio messages
- single-hop (each station within the range of any other station)
- synchronized (time is divided into *slots*)
- single *channel* (in a single slot at most one message)
- single message contains either a single key or a $\lceil \lg_2 n \rceil$-bit integer
- broadcasting/listening in a single time slot requires a unit of energetic cost
- memory of single station limited (constant number of variables storing either keys or $\lceil \lg_2 n \rceil$-bit integers).

Complexity measures of algorithms

Time: the number of time slots used by the algorithm

Complexity measures of algorithms

Time: the number of time slots used by the algorithm

Energetic cost of the algorithm: the maximal energy dissipated by a single station.

Statement of the problem

- *Informally:* We want to sort a sequence that is *almost* sorted.

Statement of the problem

- *Informally:* We want to sort a sequence that is *almost* sorted.
- A *k-disturbed sequence* – a sequence obtained from a sorted sequence by changing at most k keys.

Statement of the problem

- *Informally:* We want to sort a sequence that is *almost* sorted.
- A *k-disturbed sequence* – a sequence obtained from a sorted sequence by changing at most k keys.
- Each station stores one *old key* and one *new key* (either equal to the old one or *changed*).

Statement of the problem

- *Informally:* We want to sort a sequence that is *almost* sorted.
- A *k-disturbed sequence* – a sequence obtained from a sorted sequence by changing at most k keys.
- Each station stores one *old key* and one *new key* (either equal to the old one or *changed*).
- The sequence of the *old keys is sorted* (i.e. each station knows the position of its old key in the sorted sequence).

Statement of the problem

- *Informally:* We want to sort a sequence that is *almost* sorted.
- A k*-disturbed sequence* – a sequence obtained from a sorted sequence by changing at most k keys.
- Each station stores one *old key* and one *new key* (either equal to the old one or *changed*).
- The sequence of the *old keys is sorted* (i.e. each station knows the position of its old key in the sorted sequence).
- We want to sort the sequence of the *new keys* (i.e. each station has to learn the index of its new key in the sorted sequence of the new keys).

Sorting algorithms for this model

Let n be the number of keys (stations).

Sorting algorithms for this model

Let n be the number of keys (stations).

- Singh, Prasanna (PERCOM 2003) sorting based on energetically balanced implementation of selection algorithm.
  - Time: $O(n \log n)$
  - Energy: $O(\log n)$

Sorting algorithms for this model

Let n be the number of keys (stations).

- Singh, Prasanna (PERCOM 2003) sorting based on energetically balanced implementation of selection algorithm.
  - Time: $O(n \log n)$
  - Energy: $O(\log n)$
- (SOFSEM 2006) simple sorting based on (moderately) balanced merging
  - Time: $O(n \log n)$
  - Energy: $O(\log^2 n)$
  - low constants under O

Our results

Let n be the number of stations and let k be the number of actually changed keys.

Our results

Let n be the number of stations and let k be the number of actually changed keys.
We can sort a k-disturbed sequence:

- in time: $4n + k \cdot (\lceil \lg k \rceil^2 + \lceil \lg(n-k+1) \rceil + 6\lceil \lg k \rceil) - 2$

Our results

Let n be the number of stations and let k be the number of actually changed keys.
We can sort a k-disturbed sequence:

- in time: $4n + k \cdot (\lceil \lg k \rceil^2 + \lceil \lg(n - k + 1) \rceil + 6\lceil \lg k \rceil) - 2$
  For example: If k is $o(n/\log n)$ then time is $o(n \log n)$

Our results

Let n be the number of stations and let k be the number of actually changed keys.
We can sort a k-disturbed sequence:

- in time: $4n + k \cdot (\lceil \lg k \rceil^2 + \lceil \lg(n-k+1) \rceil + 6\lceil \lg k \rceil) - 2$
  For example: If k is $o(n/\log n)$ then time is $o(n \log n)$
- with energetic cost: $3 \cdot \lceil \frac{(\lceil \lg k \rceil + 1)(\lceil \lg k \rceil + 2)}{2\lfloor n/k \rfloor} \rceil + 4 \cdot \lceil \frac{\lceil \lg k \rceil}{\lfloor n/k \rfloor} \rceil + 10$

Our results

Let n be the number of stations and let k be the number of actually changed keys.
We can sort a k-disturbed sequence:

- in time: $4n + k \cdot (\lceil \lg k \rceil^2 + \lceil \lg(n-k+1) \rceil + 6\lceil \lg k \rceil) - 2$
  For example: If k is $o(n/\log n)$ then time is $o(n \log n)$
- with energetic cost: $3 \cdot \lceil \frac{(\lceil \lg k \rceil + 1)(\lceil \lg k \rceil + 2)}{2\lfloor n/k \rfloor} \rceil + 4 \cdot \lceil \frac{\lceil \lg k \rceil}{\lfloor n/k \rfloor} \rceil + 10$
  For example: If k is $O(n/\log^2 n)$ then energetic cost is $O(1)$

Our results

Let n be the number of stations and let k be the number of actually changed keys.
We can sort a k-disturbed sequence:

- in time: $4n + k \cdot (\lceil \lg k \rceil^2 + \lceil \lg(n - k + 1) \rceil + 6\lceil \lg k \rceil) - 2$
  For example: If k is $o(n/\log n)$ then time is $o(n \log n)$
- with energetic cost: $3 \cdot \lceil \frac{(\lceil \lg k \rceil + 1)(\lceil \lg k \rceil + 2)}{2\lfloor n/k \rfloor} \rceil + 4 \cdot \lceil \frac{\lceil \lg k \rceil}{\lfloor n/k \rfloor} \rceil + 10$
  For example: If k is $O(n/\log^2 n)$ then energetic cost is $O(1)$
- if $\frac{(\lceil \lg k \rceil + 1)(\lceil \lg k \rceil + 2)}{2} + \lceil \lg k \rceil \leq \lfloor n/k \rfloor$ then the energetic cost is bounded by 14.

Our results

Let n be the number of stations and let k be the number of actually changed keys.
We can sort a k-disturbed sequence:

- in time: $4n + k \cdot (\lceil \lg k \rceil^2 + \lceil \lg(n-k+1) \rceil + 6\lceil \lg k \rceil) - 2$
  For example: If k is $o(n/\log n)$ then time is $o(n \log n)$
- with energetic cost: $3 \cdot \lceil \frac{(\lceil \lg k \rceil + 1)(\lceil \lg k \rceil + 2)}{2\lfloor n/k \rfloor} \rceil + 4 \cdot \lceil \frac{\lceil \lg k \rceil}{\lfloor n/k \rfloor} \rceil + 10$
  For example: If k is $O(n/\log^2 n)$ then energetic cost is $O(1)$
- if $\frac{(\lceil \lg k \rceil + 1)(\lceil \lg k \rceil + 2)}{2} + \lceil \lg k \rceil \leq \lfloor n/k \rfloor$ then the energetic cost is bounded by 14.

k is not fixed nor limited. The algorithm adapts itself to arbitrary $k \leq n$.

outline of the algorithm

1. `split-and-count`:

outline of the algorithm

1. `split-and-count`:
   - Each station with unchanged key (a-*key*) learns its position in the (sorted) sequence of the unchanged keys (a-*sequence*).

outline of the algorithm

1. `split-and-count`:
   - Each station with unchanged key (a-*key*) learns its position in the (sorted) sequence of the unchanged keys (a-*sequence*).
   - Each station with changed key (b-*key*) learns its position in the (unsorted) sequence of the changed keys (b-*sequence*).

outline of the algorithm

1. `split-and-count`:
   - Each station with unchanged key (a-*key*) learns its position in the (sorted) sequence of the unchanged keys (a-*sequence*).
   - Each station with changed key (b-*key*) learns its position in the (unsorted) sequence of the changed keys (b-*sequence*).
   - Each station learns k – the number of changed keys.

outline of the algorithm

1. `split-and-count`:
   - Each station with unchanged key (a-*key*) learns its position in the (sorted) sequence of the unchanged keys (a-*sequence*).
   - Each station with changed key (b-*key*) learns its position in the (unsorted) sequence of the changed keys (b-*sequence*).
   - Each station learns k – the number of changed keys.

   If k is large fraction of n then apply sorting algorithm and stop, else continue.

outline of the algorithm

1. `split-and-count`:
   - Each station with unchanged key (a-*key*) learns its position in the (sorted) sequence of the unchanged keys (a-*sequence*).
   - Each station with changed key (b-*key*) learns its position in the (unsorted) sequence of the changed keys (b-*sequence*).
   - Each station learns k – the number of changed keys.

   If k is large fraction of n then apply sorting algorithm and stop, else continue.
2. `assign-workers`: Each changed key is assigned $\lfloor n/k \rfloor$ stations, that will balance among themselves the energetic cost of the following procedures.

outline of the algorithm

1. `split-and-count`:
   - Each station with unchanged key (a-*key*) learns its position in the (sorted) sequence of the unchanged keys (a-*sequence*).
   - Each station with changed key (b-*key*) learns its position in the (unsorted) sequence of the changed keys (b-*sequence*).
   - Each station learns k – the number of changed keys.

   If k is large fraction of n then apply sorting algorithm and stop, else continue.

2. `assign-workers`: Each changed key is assigned $\lfloor n/k \rfloor$ stations, that will balance among themselves the energetic cost of the following procedures.

3. `sort`: Sorting the b-sequence

outline of the algorithm

1. `split-and-count`:
   - Each station with unchanged key (a-*key*) learns its position in the (sorted) sequence of the unchanged keys (a-*sequence*).
   - Each station with changed key (b-*key*) learns its position in the (unsorted) sequence of the changed keys (b-*sequence*).
   - Each station learns k – the number of changed keys.

   If k is large fraction of n then apply sorting algorithm and stop, else continue.
2. `assign-workers`: Each changed key is assigned $\lfloor n/k \rfloor$ stations, that will balance among themselves the energetic cost of the following procedures.
3. `sort`: Sorting the b-sequence
4. `final-merge`: Merging the b-sequence with the a-sequence.

split-and-count

- Each station compares its *new* key to its *old* key.

split-and-count

- Each station compares its *new* key to its *old* key.
- For $0 \leq t \leq n-2$, in time slot t, the station s_t sends to s_{t+1} the number of changes in $s_0, \ldots, s_t$.

`split-and-count`

- Each station compares its *new* key to its *old* key.
- For $0 \leq t \leq n-2$, in time slot t, the station s_t sends to s_{t+1} the number of changes in $s_0, \ldots, s_t$.
- In time slot $n-1$ the station s_{n-1} sends the global number of changes k to the remaining stations.

split-and-count

- Each station compares its *new* key to its *old* key.
- For $0 \leq t \leq n-2$, in time slot t, the station s_t sends to s_{t+1} the number of changes in $s_0, \ldots, s_t$.
- In time slot $n-1$ the station s_{n-1} sends the global number of changes k to the remaining stations.

Time: n

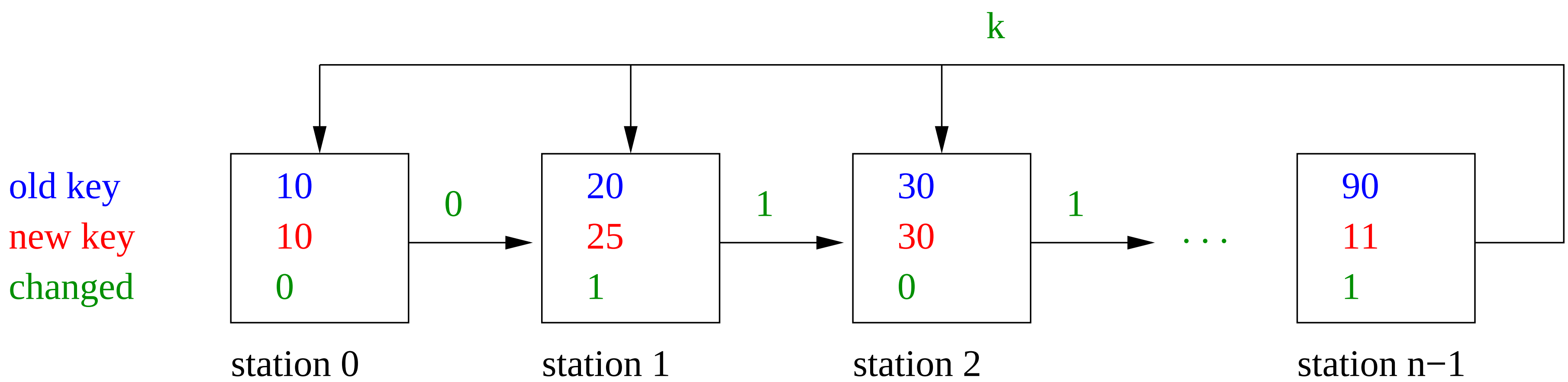
`split-and-count`

- Each station compares its *new* key to its *old* key.
- For $0 \leq t \leq n-2$, in time slot t, the station s_t sends to s_{t+1} the number of changes in $s_0, \ldots, s_t$.
- In time slot $n-1$ the station s_{n-1} sends the global number of changes k to the remaining stations.

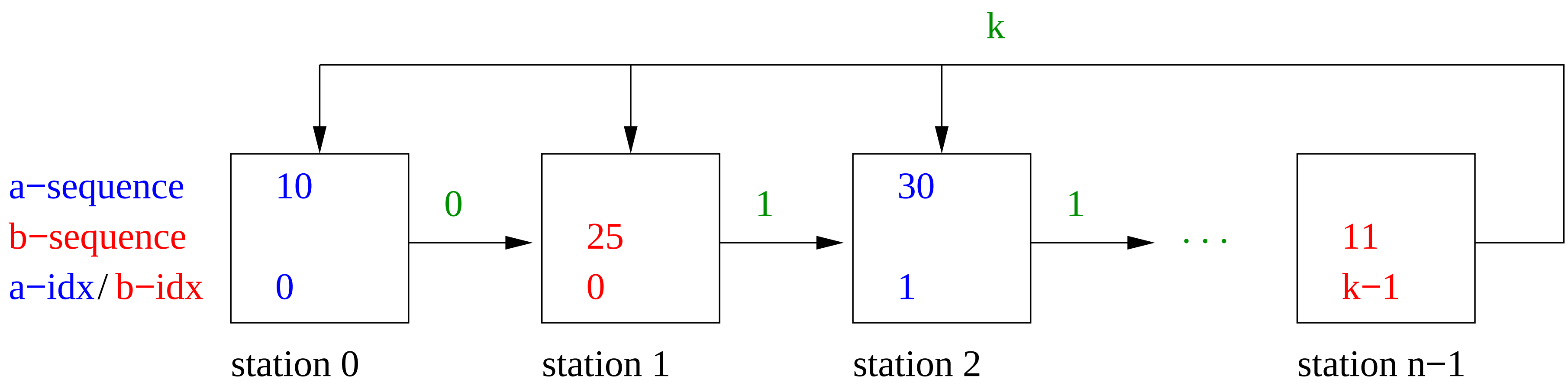
Time: n

Energetic cost: 3

split-and-count



split-and-count



`split-and-count`

After `split-and-count`:

`split-and-count`

After `split-and-count`:

- Each station knows k.

`split-and-count`

After `split-and-count`:

- Each station knows k.
- Each station with *unchanged* key (*owner of this* a*-key*) knows its position in the (sorted) a-sequence.

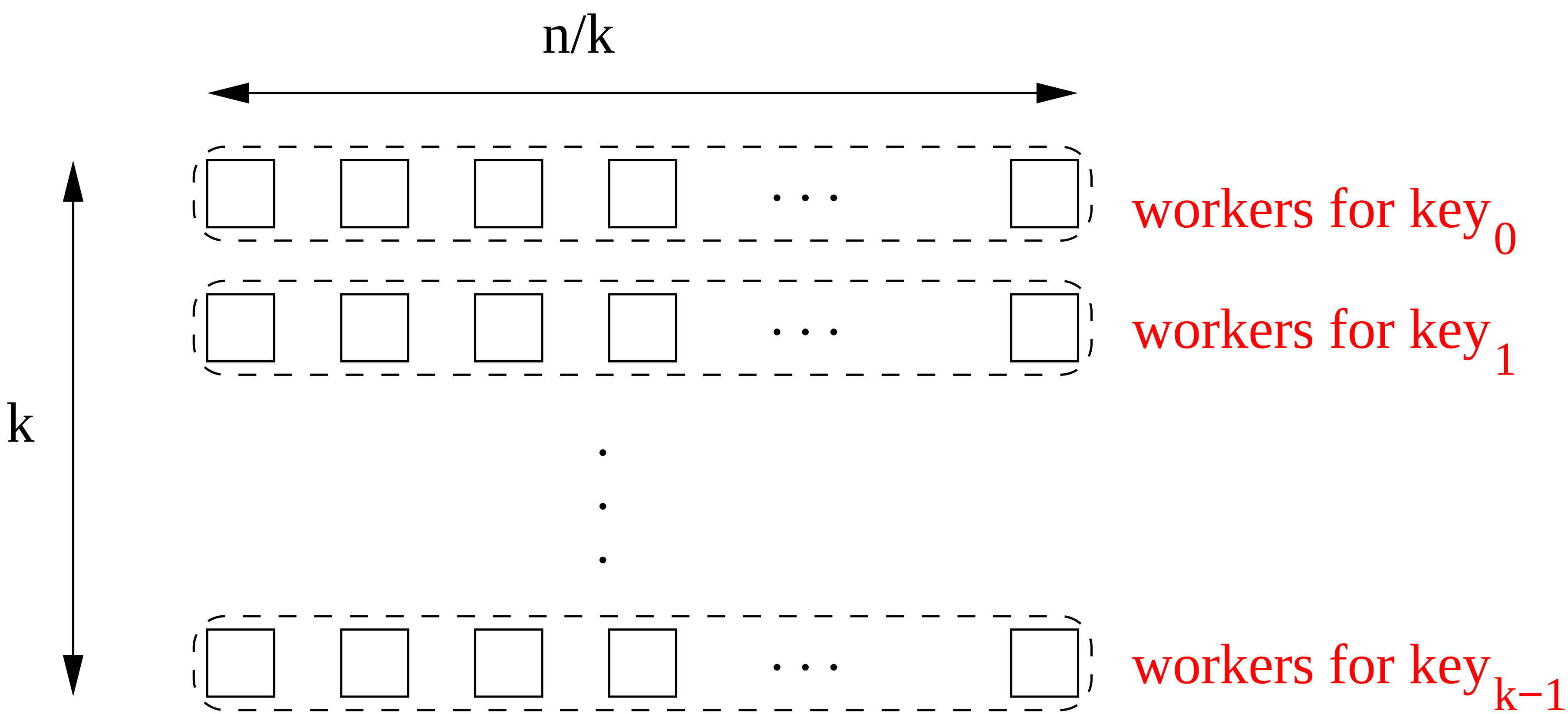
`split-and-count`

After `split-and-count`:

- Each station knows k.
- Each station with *unchanged* key (*owner of this a-key*) knows its position in the (sorted) a-sequence.
- Each station with *changed* key (*owner of this b-key*) knows its position in the (unsorted) b-sequence.

assign-workers

Let key_i denote the ith b-key. All stations know k. They are arranged in the following matrix:

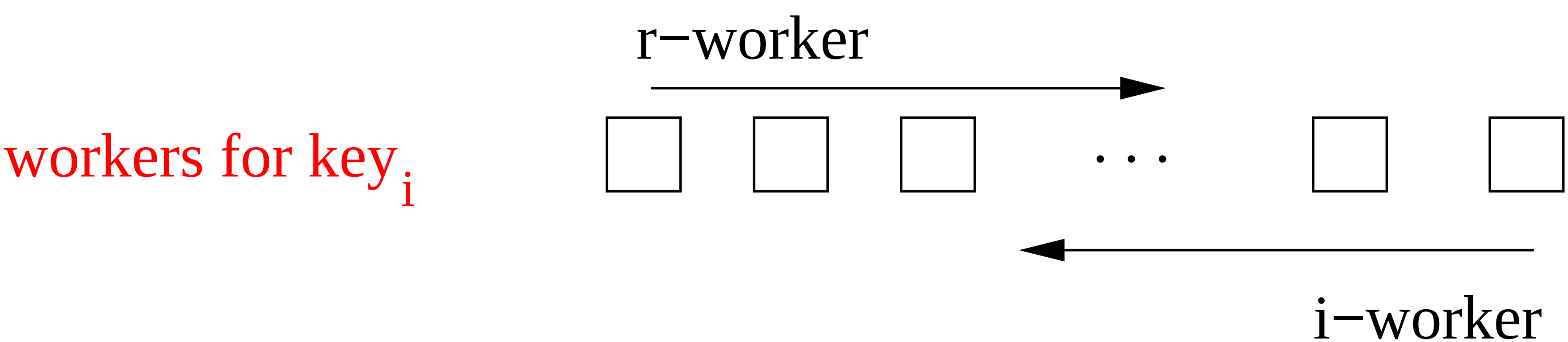


assign-workers

- For $0 \leq t \leq k-1$, in time slot t, the owner of key_t sends key_t to the workers for key_t.

assign-workers

- For $0 \leq t \leq k-1$, in time slot t, the owner of key_t sends key_t to the workers for key_t.
- For $0 \leq i \leq k-1$,
  - the *first* worker for key_i becomes the current *r-worker* (*rank-worker*).
  - the *last* worker for key_i becomes the current *i-worker* (*index-worker*).



assign-workers

Time: k

assign-workers

Time: k

Energetic cost: 2

sort

The b-sequence is sorted by a (balanced) merge-sort:

begin

$m \leftarrow 1$;

while $m < k$ **do**

merge all pairs of subsequences of length m;

$m \leftarrow 2 \cdot m$;

end

merging

To merge two sorted sequences, each key from one sequence has to learn its rank in the other sequence. Then it can compute its index in the merged sequence.

procedure `merge`(seq_1, seq_2)
begin
  `rank`(seq_1, seq_2);
  `rank`(seq_2, seq_1);
end

ranking

Ranking seq_1 in seq_2:

ranking

Ranking seq_1 in seq_2:

- For each key of seq_i, its current i-worker knows its index in the *sorted* seq_i.

ranking

Ranking seq_1 in seq_2:

- For each key of seq_i, its current i-worker knows its index in the *sorted* seq_i.
- The sorted sequence seq_2, permuted by a special permutation (bso), is transmitted by the i-workers of seq_2. (Each i-worker transmits once.)

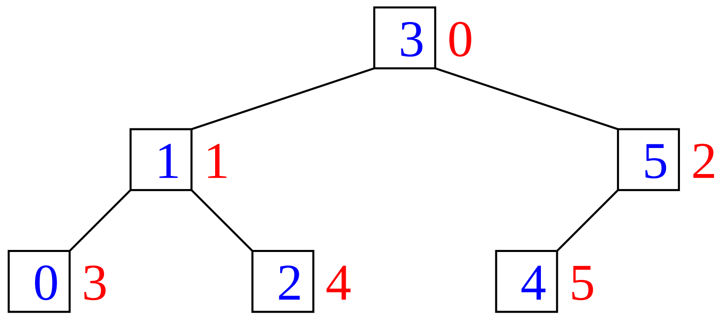
ranking

Ranking seq_1 in seq_2:

- For each key of seq_i, its current i-worker knows its index in the *sorted* seq_i.
- The sorted sequence seq_2, permuted by a special permutation (bso), is transmitted by the i-workers of seq_2.
  (Each i-worker transmits once.)
- During these transmissions, for each key of seq_1, some its r-workers are used to compute its rank in seq_2.
  (Each r-worker uses constant energy.)

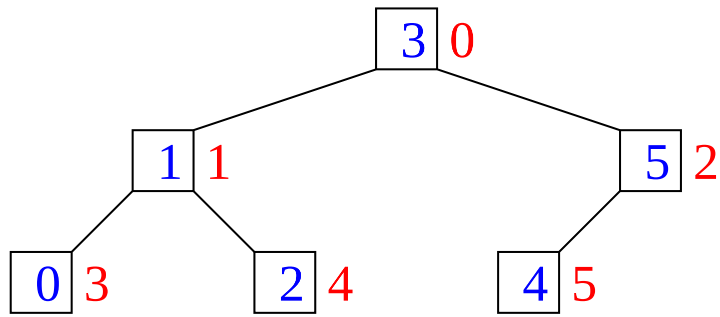
Ranking: Binary tree T_m, bso

Let m be the length of seq_2. The elements of seq_2 are arranged in the following tree:



Ranking: Binary tree T_m, bso

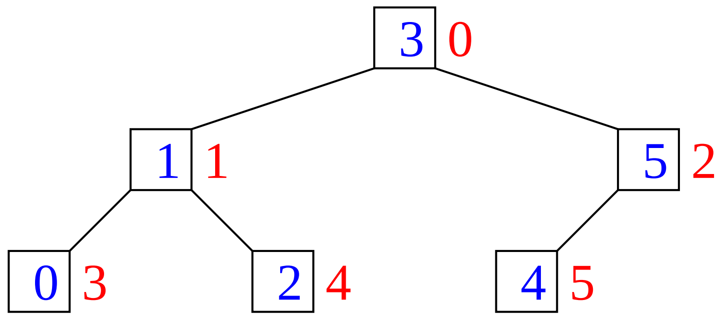
Let m be the length of seq_2. The elements of seq_2 are arranged in the following tree:



- x-*indexing* – in-order (indexes of seq_2)

Ranking: Binary tree T_m, bso

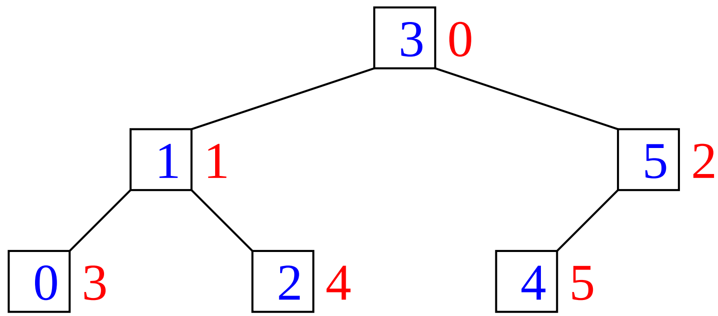
Let m be the length of seq_2. The elements of seq_2 are arranged in the following tree:



- x-*indexing* – in-order (indexes of seq_2)
- y-*indexing* – heap-order (transmissions order)

Ranking: Binary tree T_m, bso

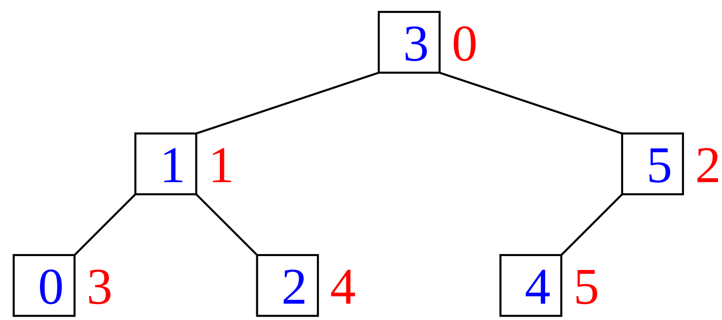
Let m be the length of seq_2. The elements of seq_2 are arranged in the following tree:



- x-*indexing* – in-order (indexes of seq_2)
- y-*indexing* – heap-order (transmissions order)
- bso_m – an ("easily computable") permutation:
  for a node with x-index x, $y = bso_m(x)$ is its y-index.

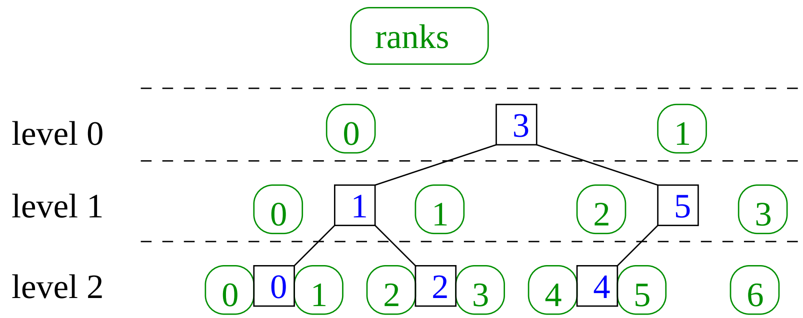
Ranking: Binary tree T_m, bso

Let m be the length of seq_2. The elements of seq_2 are arranged in the following tree:

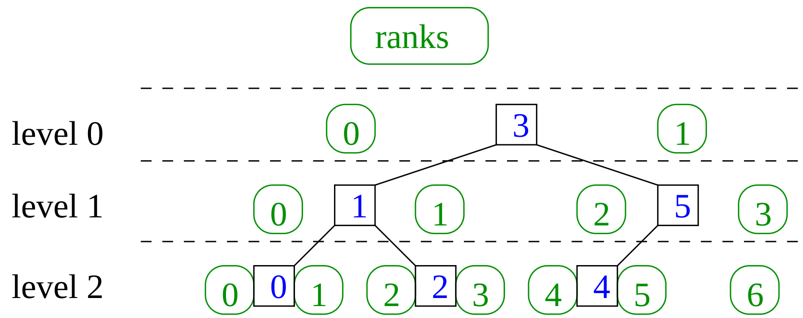


- x-*indexing* – in-order (indexes of seq_2)
- y-*indexing* – heap-order (transmissions order)
- bso_m – an ("easily computable") permutation:
  for a node with x-index x, $y = bso_m(x)$ is its y-index.
- the ith element of the sorted seq_2 is transmitted as the tth, where $t = bso_m(i)$.

Ranking: Layers of T_m

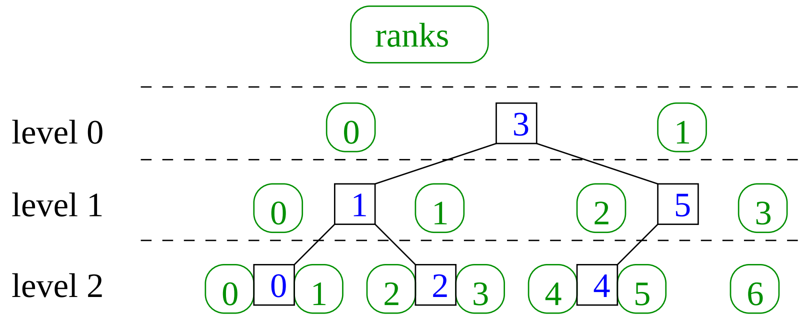


Ranking: Layers of T_m



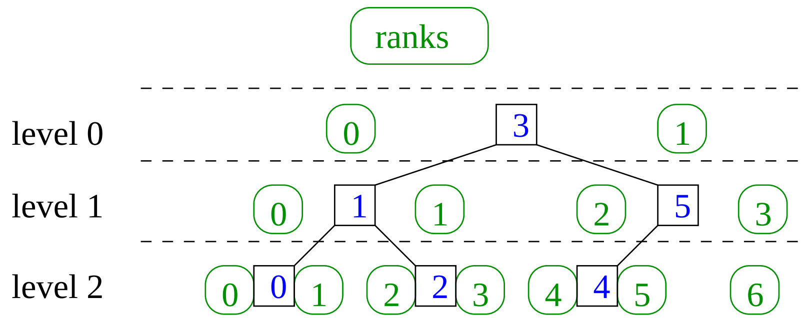
- seq_2 is transmitted level by level.

Ranking: Layers of T_m



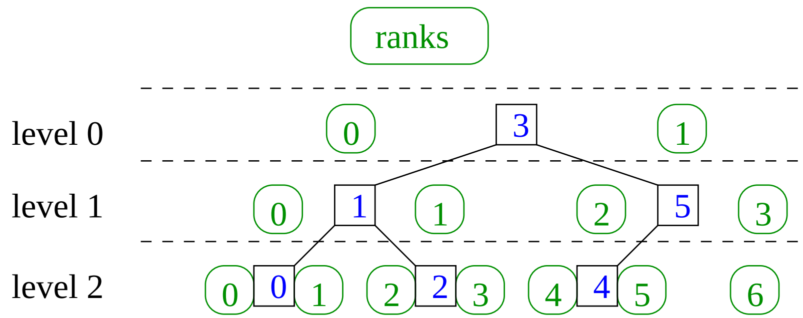
- seq_2 is transmitted level by level.
- For each key of seq_1, its current r-worker:

Ranking: Layers of T_m



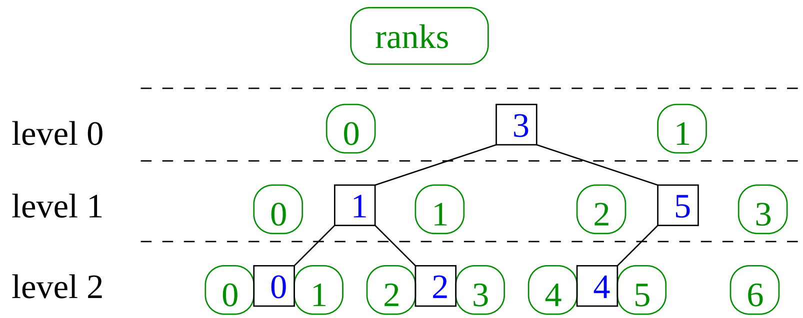
- seq_2 is transmitted level by level.
- For each key of seq_1, its current r-worker:
  - knows its rank in the previously transmitted levels,

Ranking: Layers of T_m



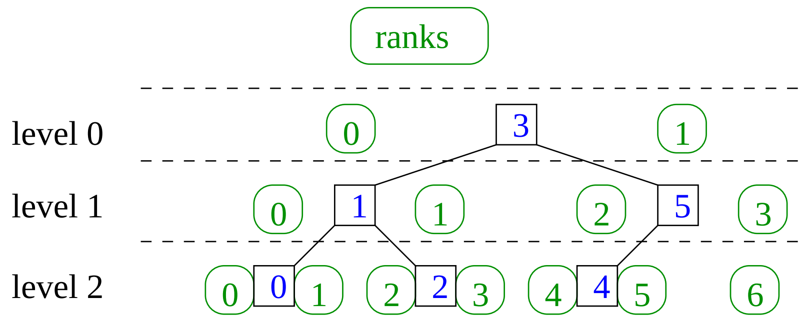
- seq_2 is transmitted level by level.
- For each key of seq_1, its current r-worker:
  - knows its rank in the previously transmitted levels,
  - listens *only once* in the next level to compute its rank in the subsequence of seq_2 enhanced by this level,

Ranking: Layers of T_m

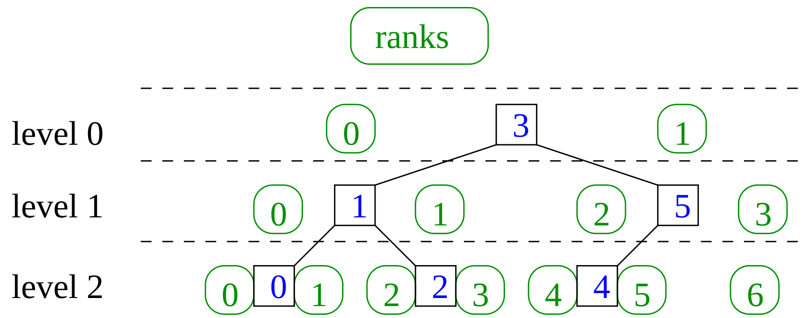


- seq_2 is transmitted level by level.
- For each key of seq_1, its current r-worker:
  - knows its rank in the previously transmitted levels,
  - listens *only once* in the next level to compute its rank in the subsequence of seq_2 enhanced by this level,
- between the the levels, the current r-workers of seq_1 transfer the ranks to the next r-workers.

Ranking: Layers of T_m



Ranking: Layers of T_m



- After the last level, each r-worker of seq_1 sends the rank to the i-worker which computes the index of its key in the sequence merged from seq_1 and seq_2. (Procedure `send-ranks-to-indexes`.)

`final-merge`

Phase 1: Computing output index for each b-key:

- `rank`(b-sequence, a-sequence)   (also blanced!)

`final-merge`

Phase 1: Computing output index for each b-key:

- `rank`(b-sequence, a-sequence) (also blanced!)
- Each owner of b-key *overhears* in `send-ranks-to-indexes` the rank of this b-key in the a-sequence.

`final-merge`

Phase 1: Computing output index for each b-key:

- `rank`(b-sequence, a-sequence)   (also blanced!)
- Each owner of b-key *overhears* in `send-ranks-to-indexes` the rank of this b-key in the a-sequence.
- Each owner of b-key is informed by its i-worker about the final index of this key in the *sorted sequence of all keys*. (Thus, it can also compute its index in the sorted b-sequence.)

`final-merge`

Phase 1: Computing output index for each b-key:

- `rank`(b-sequence, a-sequence) (also blanced!)
- Each owner of b-key *overhears* in `send-ranks-to-indexes` the rank of this b-key in the a-sequence.
- Each owner of b-key is informed by its i-worker about the final index of this key in the *sorted sequence of all keys*. (Thus, it can also compute its index in the sorted b-sequence.)

Now each owner of b-key knows:

- its index in the sorted output
- its index in the sorted b-sequence
- its rank in the a-sequence

final-merge

Phase 2: Computing output index for each a-key:

- In the sorted b-sequence, each b-key informs its predecessor about its rank in the a-sequence. (Each *last* b-key with given rank becomes aware of this fact.)

final-merge

Phase 2: Computing output index for each a-key:

- In the sorted b-sequence, each b-key informs its predecessor about its rank in the a-sequence. (Each *last* b-key with given rank becomes aware of this fact.)
- For $0 \leq t \leq n - k - 1$, in time slot t, the last b-key from the sorted b-sequence with the rank t (if exists) informs the tth a-key from a-sequence about its index in the sorted b-sequence. (a *displacement* of this a-key).

final-merge

Phase 2: Computing output index for each a-key:

- In the sorted b-sequence, each b-key informs its predecessor about its rank in the a-sequence. (Each *last* b-key with given rank becomes aware of this fact.)
- For $0 \leq t \leq n-k-1$, in time slot t, the last b-key from the sorted b-sequence with the rank t (if exists) informs the tth a-key from a-sequence about its index in the sorted b-sequence. (a *displacement* of this a-key).
- For $0 \leq t \leq n-k-2$, in time slot t, the $(t+1)$st a-key that did not receive its displacement from the b-sequence, receives the displacement from its predecessor in a-sequence.

final-merge

Phase 2: Computing output index for each a-key:

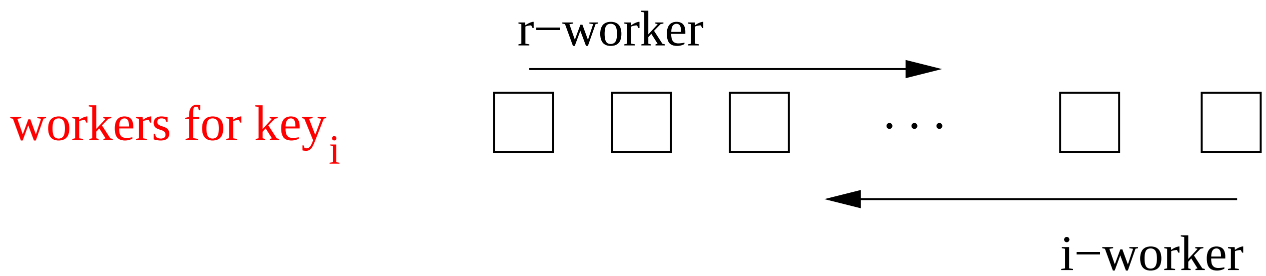
- In the sorted b-sequence, each b-key informs its predecessor about its rank in the a-sequence. (Each *last* b-key with given rank becomes aware of this fact.)
- For $0 \leq t \leq n-k-1$, in time slot t, the last b-key from the sorted b-sequence with the rank t (if exists) informs the tth a-key from a-sequence about its index in the sorted b-sequence. (a *displacement* of this a-key).
- For $0 \leq t \leq n-k-2$, in time slot t, the $(t+1)$st a-key that did not receive its displacement from the b-sequence, receives the displacement from its predecessor in a-sequence.
- Each a-key adds its displacement to its index in a-sequence to get its index in the *sorted sequence of all keys*.

`final-merge` – complexity

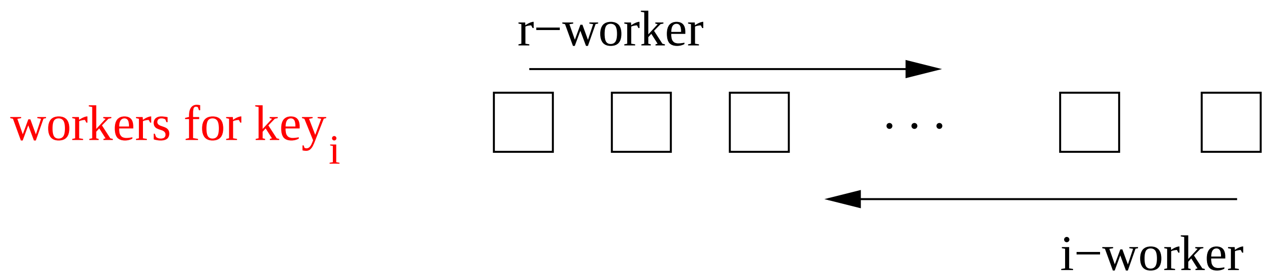
Time: $O(n)$

Energetic cost: $O(1) +$ the cost of `rank`

Balancing the energy in `sort` and `final-merge`

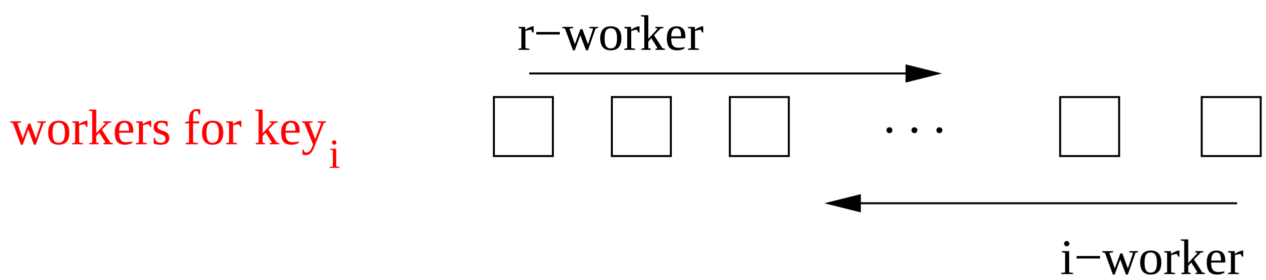


Balancing the energy in `sort` and `final-merge`



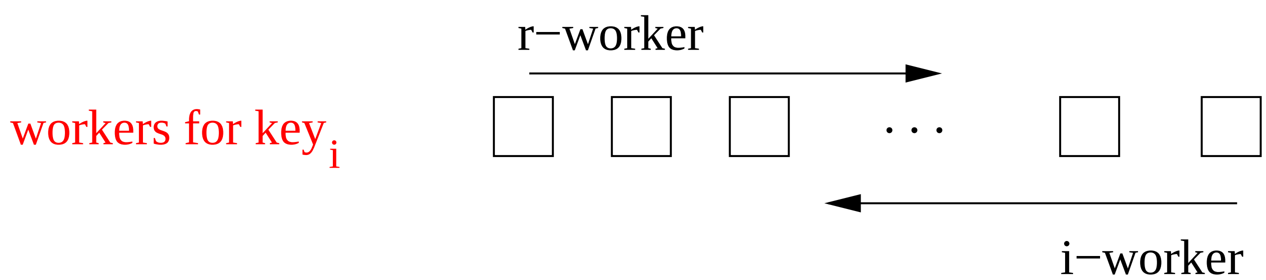
- After each `merge`(seq_1, seq_2) , for each key of seq_1 and seq_2, the task of i-worker is transfered to the previous (modulo $\lfloor n/k \rfloor$) worker.(For each key – at most $\lceil \lg k \rceil$ such transfers.)

Balancing the energy in `sort` and `final-merge`



- After each `merge`(seq_1, seq_2) , for each key of seq_1 and seq_2, the task of i-worker is transfered to the previous (modulo $\lfloor n/k \rfloor$) worker.(For each key – at most $\lceil \lg k \rceil$ such transfers.)
- After each level of `rank`(seq_1, seq_2), for each key of seq_1, the task of r-worker is transfered to the next (modulo $\lfloor n/k \rfloor$) worker.(For each key – at most $\frac{(\lceil \lg k \rceil + 1)(\lceil \lg k \rceil + 2)}{2}$ such transfers.)

Balancing the energy in `sort` and `final-merge`



- After each `merge`(seq_1, seq_2) , for each key of seq_1 and seq_2, the task of i-worker is transfered to the previous (modulo $\lfloor n/k \rfloor$) worker.(For each key – at most $\lceil \lg k \rceil$ such transfers.)
- After each level of `rank`(seq_1, seq_2), for each key of seq_1, the task of r-worker is transfered to the next (modulo $\lfloor n/k \rfloor$) worker.(For each key – at most $\frac{(\lceil \lg k \rceil + 1)(\lceil \lg k \rceil + 2)}{2}$ such transfers.)
- The energetic cost of each transfer is constant.

Final remarks

- Robustness to interferences: How to correct sequences in the model, where each message is received with probability $p < 1$?

  Algorithm for sorting has been proposed on ADHOC-NOW 2008.

Final remarks

- Robustness to interferences: How to correct sequences in the model, where each message is received with probability $p < 1$?

  Algorithm for sorting has been proposed on ADHOC-NOW 2008.

- Simulation in Java available at:

  `http://www.im.pwr.wroc.pl/~kik/CorrectionRN.java`

THE END

THANK YOU!