Merging and Merge-sort in a Single Hop Radio Network*

Marcin Kik
kik@im.pwr.wroc.pl

Institute of Mathematics and Computer Science,
Wrocław University of Technology
ul. Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland

Abstract. We present two merging algorithms on a single-channel single-hop radio network without collision detection. The simpler of these algorithms merges two sorted sequences of length n in time $4n$ with energetic cost for each station $\approx \lg n$. The energetic cost of broadcasting is constant. This yields the merge-sort for n elements in time $2n \lg n$, where the energetic cost for each station is $\frac{1}{2} \lg^2 n + \frac{7}{2} \lg n$ (the energetic cost of broadcasting is only $2 \lg n$), which seems to be suitable for practical applications due to its simplicity and low constants. We also present algorithm for merging in time $O(n \lg^* n)$ with energetic cost $O(\lg^* n)$.

1 Introduction

A *radio network* consists of processing units (called *stations*) which communicate with each other by broadcasting radio messages. There are two important complexity measures of the radio network algorithms: *time* and *energy consumption*. Most of energy is consumed by broadcasting and listening to messages. The stations are often powered by batteries. If a single station fails due to battery exhaustion, then the whole task performed by the network may also fail. Therefore we want to implement algorithms in such a way that the maximal energy used by a single station is minimized. There are many problems concerning self-organization of the network (such as leader election and initialization [8], [5], [6]) that are nontrivial even in the single-hop networks. We may also need to process or organize data distributed among the stations (for example some measurements made by the stations). Some of the typical examples of such problems are finding minimum, maximum, median [10], average value [7], or sorting [11].

We consider a network of n numbered stations $s_1, \ldots, s_n$ communicating through a single radio channel. Each station s_i knows the value n, its own number i and stores a single key in its variable $key[s_i]$. We want to sort the keys within the network. (The keys are *sorted* if, for each pair of stations, the station with a lower number holds the lower key.) All stations are synchronized. Time is divided into *slots*. Within a single time slot a single message can be broadcast. We consider *single-hop* network: Message broadcast by any station can be received by any other station. A single message contains

* Partially supported by KBN, grant number 3T11C 011 26 in year 2004 and by the European Union within the 6th Framework Programme under contract 001907 (DELIS).

$O(\max\{B, \lg n\})$ bits, where B is the number of bits of a single key. (Typically $B = \Theta(\lg n)$.) Broadcasting and listening in a single time slot requires a unit of *energetic cost*. Each station has limited memory. It can contain a constant number of words of $O(\max\{B, \lg n\})$ bits each. By *energetic cost of the algorithm* we mean the maximal energy dissipated by a single station. We do not assume the existence of the "wake up mechanism" with a low power paging channel, as described in [10], [11]. Each station predicts its next time slot for listening or broadcasting using only its internal clock and state.

There exists an algorithm [9] that sorts n elements in time $O(n)$ with energetic cost of broadcasting $O(1)$. However the energetic cost of listening in this algorithm is $\Theta(n)$. A comparator network can also be transformed into algorithm for single-hop networks: each comparator is simulated in two consecutive time slots, when two endpoints of the comparator exchange their values. The time of such algorithm (in single channel) is two times the number of comparators, and the energetic cost is not greater than two times the depth of the network. Thus the AKS sorting network [1] can be transformed into (impractical) sorting algorithm with time $O(n \lg n)$ and energetic cost $O(\lg n)$ and the Batcher networks [2] can be transformed into sorting algorithms with time $O(n \lg^2 n)$ and energetic cost $O(\lg^2 n)$. However, in radio network, a single message can be listened by many stations and the messages may contain other information besides the keys. Singh and Prasanna [10], [11] proposed algorithm sorting in time $O(n \lg n)$ with energetic cost $O(\lg n)$ by implementing quick-sort and selecting the median as the partitioning element in each recursive call with energetically balanced implementation of asymptotically optimal selection algorithm [3]. It is sophisticated and the constants involved are large (although not as large as in the AKS network) (see simulation results in [11]).

1.1 Result

We present two merging procedures. The first one merges two sequences of length n in time $O(n)$ with energetic cost of listening $O(\lg n)$ and of broadcasting $O(1)$. It can be used for implementation of sorting in time $O(n \lg n)$ and energetic cost of listening $O(\lg^2 n)$ and of broadcasting $O(\lg n)$ based on the classical merge-sort algorithm (see [4]). Although the asymptotic energetic cost of listening for sorting is worse than that obtained by Singh and Prasanna, it seems to be more suitable for practical implementations due to the low constants and simplicity. The energetic cost of broadcasting in merging is only $O(1)$ and in merge-sort is $O(\lg n)$. This is important since in practice broadcasting requires more energy than listening. The second presented merging algorithm works in time $O(n \lg^* n)$ with energetic cost of listening and broadcasting $O(\lg^* n)$. To the knowledge of the author it is not known whether there exists merging algorithm with asymptotically lower energetic cost or whether there is any non-constant lower bound for energetic cost of merging. This algorithm can also be used for merge-sorting in time in time $O(n \lg n \lg^* n)$ with energetic cost $O(\lg n \lg^* n)$. Implementations of the simulations of these algorithms can be found at [12].

Theorem 1. *There exist algorithms that merge two sorted sequences of length m on a single hop radio network without collision detection:*

– *in time* $4m$ *with energetic cost of listening* $\lceil \lg(m+1) \rceil + 1$ *and of broadcasting* 2.
– *in time* $O(m \lg^* m)$ *with energetic cost of listening and broadcasting* $O(\lg^* m)$

2 Merging

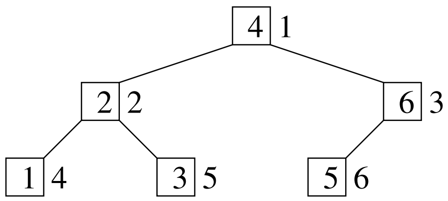

Fig. 1. Tree T_6. Right to the nodes are their *preorder* indexes.

For simplicity of description we assume that all the keys are pairwise distinct. Let T_m denote a balanced binary tree consisting of the nodes $1, \ldots, m$: If $m = 2^k - 1$, for some integer $k > 0$, then T_m is a complete binary tree. If $m = 2^k - 1 - l$, for some positive integer $l < 2^{k-1}$, then the l rightmost leaves are missing. The nodes are placed in T_m in the *inorder* order (i.e. for each node x the nodes in its left subtree are less than x and the nodes in its right subtree are greater than x). By $l(m, x)$ (respectively $r(m, x)$), for $1 \leq x \leq m$, we denote the left (respectively right) child of node x in T_m. (A non-existing child is represented by NIL.) By $p(m, x)$ we denote the index of node x in T_m in *preorder* ordering. (I.e. the *preorder* index of the root is 1, then the nodes on the second level are indexed from left to right, then on the third level, and so on.) We also assume that $p(m, NIL) = NIL$. An example of T_m for $m = 6$ is given in Figure 1. Note that the height (number of levels) of T_m is $\min\{k : 2^k - 1 \geq m\} = \lceil \lg(m+1) \rceil$ (where "lg" denotes "$\log_2$"). For $m \geq 1$, we define a sequence $h(m, 0), h(m, 1), \ldots$ as follows:

$$h(m, i) = \begin{cases} m & \text{if } i = 0 \\ \lceil \lg(h(m, i-1) + 1) \rceil & \text{if } i \geq 1 \end{cases} \tag{1}$$

Let $l^*(m) = \min\{i : h(m, i) \leq 2\}$. Note that $l^*(m) = O(\lg^* m)$. (Note also, that $l^*(m) \leq 4$, for $m \leq 2^{127} - 1$.) The functions $l(m, x)$, $r(m, x)$, $p(m, x)$, $h(m, i)$ and $l^*(m)$ can be computed internally by each station.

We want to merge two sorted sequences of keys stored in stations $\langle a_1, \ldots, a_m \rangle$ and $\langle b_1, \ldots, b_m \rangle$ into a single sorted sequence of length $2m$ stored in $\langle a_1, \ldots, a_m, b_1, \ldots b_m \rangle$. Procedure Rank (see Algorithm 1) computes the rank of each element of the first sequence in the second sequence. (By the *rank* of a_i in $\langle b_1, \ldots, b_m \rangle$ we mean the number of elements b_j with $key[b_j] < key[a_i]$.) The result of Rank for each a_i is in $rank[a_i]$. Note that it is a parallel implementation of the classical bisection algorithm, where each station a_i predicts when its next bisecting key will be broadcast by some b_j. The bisecting keys are broadcast in appropriate order, since in *preorder* each key is preceded by all the keys from the higher levels of T_m. The time of `Rank` is m slots. The

procedure Rank($\langle a_1, \ldots, a_m \rangle$, $\langle b_1, \ldots, b_m \rangle$)
Each station a_i does: $timer[a_i] \leftarrow 1$; $rank[a_i] \leftarrow 0$;
for *time slot* $d \leftarrow 1$ **to** m **do**
  let x be such that $p(m, x) = d$; (* d is preorder index of x *)
  station b_x broadcasts $\langle k \rangle$, where $k = key[b_x]$;
  each station a_j with $timer[a_j] = d$ listens and does:
  if $key[a_j] < k$ **then**
    $timer[a_j] \leftarrow p(m, l(m, x))$; (* preorder index of left child of x *)
  else
    $timer[a_j] \leftarrow p(m, r(m, x))$; (* preorder index of right child of x *)
    $rank[a_j] \leftarrow x$;

Algorithm 1: Procedure Rank

energetic cost of broadcasting is 1. (Each b_i broadcasts once.) The energetic cost of listening is $\lceil \lg(m+1) \rceil$, since each a_i listens at most once at each level of T_m. Rank can be used for merging two sorted sequences as in the procedure Merge (Algorithm 2). The time of Merge($\langle a_1, \ldots, a_m \rangle$, $\langle b_1, \ldots, b_m \rangle$) is $4m$. The energetic cost of listen-

procedure Merge($\langle a_1, \ldots, a_m \rangle$, $\langle b_1, \ldots, b_m \rangle$)
Rank($\langle a_1, \ldots, a_m \rangle$,$\langle b_1, \ldots, b_m \rangle$);
Rank($\langle b_1, \ldots, b_m \rangle$,$\langle a_1, \ldots, a_m \rangle$);
All stations a_i and b_i do: $idx[a_i] \leftarrow i + rank[a_i]$; $idx[b_i] \leftarrow i + rank[b_i]$;
(* for $1 \leq i \leq m$ let $c_i = a_i$ and $c_{m+i} = b_i$ *)
for *time slot* $t \leftarrow 1$ **to** $2m$ **do**
  station c_i with $idx[c_i] = t$ broadcasts $\langle k \rangle$, where $k = key[c_i]$;
  station c_t listens and does: $new[c_t] \leftarrow k$;
Each station c_i does: $key[c_i] \leftarrow new[c_i]$;

Algorithm 2: Procedure Merge.

ing is $\lceil \lg(m+1) \rceil + 1$. (Each station listens at most $\lceil \lg(m+1) \rceil$ times in one of the Rank procedures and once in the "for" loop.) The energetic cost of broadcasting is 2: Each station broadcasts at most twice (in one of the Rank procedures and in the "for" loop). Thus the total energetic cost is $\lceil \lg(m+1) \rceil + 3$. (This could be compared to the time $\approx 2 \cdot m \lg m$ and energetic cost $\approx 2 \lg m$ of merging procedures obtained by the transformation of Batcher merging comparator networks [2].) Note that the algorithm is correct: The key $key[a_i]$ is preceded by $idx[a_i] - 1 = i - 1 + rank[a_i]$ keys in the sorted sequence of keys from both sequences. (The same holds for each $key[b_i]$.) Since the keys are pairwise distinct, no two elements c_i have the same $idx[c_i]$ and there are no transmission collisions in the "for" loop.

2.1 Reducing the asymptotic energetic cost of merging to $O(\lg^* m)$

We reduce the asymptotic energetic cost of listening. Instead of computing the ranks of each a_i in $\langle b_1, \ldots, b_m \rangle$, we first compute the ranks of some stations b_j (*b-splitters*) in $\langle a_1, \ldots, a_m \rangle$. The b-splitters split the sequence $\langle b_1, \ldots, b_m \rangle$ into blocks (b-*blocks*) of size $h(m, 1)$. The energetic cost of computing the rank of each b-splitter is balanced among all stations in its b-block. Then the stations $a_1, \ldots, a_m$ are grouped so that the stations of each group are ranked in separate b-block. Then we split $\langle a_1, \ldots, a_m \rangle$ into a-blocks of size $h(m, 2)$ which compute the rank of their a-splitters (it is enough, to find the rank of a-splitter in its corresponding b-block) and regroup the stations $b_1, \ldots, b_m$ into separate a-blocks. We iterate this procedure while the sizes of the blocks decrease rapidly. We define an auxiliary procedure Regroup (see Algorithm 3). Let $g(m, i) = \lceil \frac{m}{h(m,i)} \rceil$ and $\alpha(m, i, j, k) = (j - 1) \cdot h(m, i) + k$. By $c_{i,j,k}$ and $d_{i-1,j,k}$ we denote the stations from $\{a_1, \ldots, a_m\}$ and $\{b_1, \ldots, b_m\}$ as follows. For $1 \leq k \leq h(m, i)$:

$$c_{i,j,k} = \begin{cases} a_{\alpha(m,i,j,k)} & \text{if } \alpha(m, i, j, k) \leq m, \\ b_{\alpha(m,i,j,k)-m} & \text{if } \alpha(m, i, j, k) > m. \end{cases}$$

For $1 \leq k \leq h(m, i - 1)$: for $\alpha(m, i - 1, j, k) \leq m$, let $d_{i-1,j,k} = b_{\alpha(m,i-1,j,k)}$ and, for $\alpha(m, i - 1, j, k) > m$, $d_{i,j,k}$ does not exist (it is treated as if $key[d_{i,j,k}] = +\infty$).

For $1 \leq j \leq g(m, i)$, $c_{i,j,1}$ is jth a-*splitter* and, for $k > 1$, $c_{i,j,k}$ is a *slave* of $c_{i,j,1}$. For parameter $i > 1$, we assume that the stations $a_1, \ldots a_m$ are grouped between the b-splitters $d_{i-1,1,1}, \ldots, d_{i-1,g(m,i-1),1}$ as follows. For any a_l and $j = group[a_l]$:

- If $1 \leq j \leq g(m, i - 1) - 1$ then $key[d_{i-1,j,1}] < key[a_l] < key[d_{i-1,j+1,1}]$.
- If $j = 0$ then $key[a_l] < key[b_1]$. (Note that $b_1 = d_{i-1,1,1}$)
- If $j = g(m, i - 1)$, then $key[a_l] > key[d_{i-1,g(m,i-1),1}]$.

Note that, for parameter $i = 1$, we do not have any assumptions. In this case $g(m, i - 1) = 1$ and each a_l has $group[a_l] = 1$. The task of Regroup is grouping of the stations $b_1, \ldots, b_m$ between the splitters $c_{i,1,1}, \ldots, c_{i,g(m,i),1}$.

We divide the code into fragments (*phases*) and analyze each phase separately. Each station has a *clock* variable t, that is increased after each time slot. In Phase 1 the rank of each splitter $c_{i,j,1}$ in $\langle b_1, \ldots, b_m \rangle$ is computed. Each splitter $d_{i-1,j',1}$ together with its slaves forms a binary tree $T_{h(m,i-1)}$. These trees are scanned level by level: first all the nodes of all the trees at level 1 (i.e. roots), then all the nodes of all the trees at level 2, and so on. The number of levels (the height of $T_{h(m,i-1)}$) is $h(m, i)$. To compute the rank of $c_{i,j,1}$ we have to consider only the tree corresponding to $group[c_{i,j,1}]$. At level l each station listens at most once and corrects its $rank'$ and $timer$. (The new value of $timer$ is either NIL or *preorder* index of some $b_{i'}$ on the next level.) Between the levels l and $l + 1$, after all stations $b_{i'}$ on level l in all the trees have broadcast their messages, the collected informations and the task of further computation is transferred from each $c_{i,j,l}$ to the next slave $c_{i,j,l+1}$. The time slot of this transfer is known in advance, since the size of each level is known. The time of Phase 1 is $O(m)$ since the number of all stations $c_{i,j,k}$ and $d_{i-1,j',k'}$ is $O(m)$ and in each time slot a different one of them broadcasts. The energetic cost is $O(1)$, since each $c_{i,j,k}$ listens once and broadcasts once and each $d_{i-1,j',k'}$ broadcasts once. After Phase 1 each $c_{i,j,h(m,i)}$ stores in $rank'[c_{i,j,h(m,i)}]$ the rank of $c_{i,j,1}$ in $\langle b_1, \ldots, b_m \rangle$. (The value $rank'[c_{i,j,1}]$ is deliberately initiated to 0 at the beginning of Phase 1: If $i > 1$ and $group[c_{i,j,1}] \geq 1$ then $key[c_{i,j,1}]$ is compared to at least one lesser key, since $key[d_{i-1,group[c_{i,j,1}],1}] < key[c_{i,j,1}]$. If $i = 1$ or $group[c_{i,j,1}] = 0$, this ensures that we do not start with too large

procedure Regroup(i, $\langle a_1, \ldots, a_m \rangle$, $\langle b_1, \ldots, b_m \rangle$)
(* Phase 1 *)
Each station $c_{i,j,1}$ does: **begin**
  $group'[c_{i,j,1}] \leftarrow group[c_{i,j,1}]$; $key'[c_{i,j,1}] \leftarrow key[c_{i,j,1}]$; $timer[c_{i,j,1}] \leftarrow 1$;
  $rank'[c_{i,j,1}] \leftarrow 0$;
end
for $l \leftarrow 1$ **to** $h(m,i)$ **do**
  (* l denotes level in $T_{h(m,i-1)}$ *)
  for $v \leftarrow 2^{l-1}$ **to** $\min\{2^l - 1, h(m,i-1)\}$ **do**
    (* v – preorder index on level l *)
    let x be such that $p(h(m,i-1),x) = v$;
    for $g \leftarrow 1$ **to** $g(m,i-1)$ **do**
      $d_{i-1,g,x}$ (if exists) broadcasts $\langle k' \rangle$, where $k' = key[d_{i-1,g,x}]$;
      Each $c_{i,j,l}$ with $group'[c_{i,j,l}] = g$ and $timer[c_{i,j,l}] = v$ listens and does:
      if *there was no message or* $key'[c_{i,j,l}] < k'$ **then**
        $timer[c_{i,j,l}] \leftarrow p(h(m,i-1), l(h(m,i-1),x))$;
      else
        $timer[c_{i,j,l}] \leftarrow p(h(m,i-1), r(h(m,i-1),x))$;
        $rank'[c_{i,j,l}] \leftarrow \alpha(m,i-1,g,x)$; (* index of $d_{i-1,g,x}$ *)
      all stations increase clock t;
  if $l < h(m,i)$ **then**
    (* not last level – TRANSFER TO THE NEXT SLAVES *)
    for $j \leftarrow 1$ *to* $g(m,i)$ **do**
      $c_{i,j,l}$ broadcasts $\langle t', r', g, k' \rangle$ where $t' = timer[c_{i,j,l}]$, $r' = rank'[c_{i,j,l}]$,
      $g = group'[c_{i,j,l}]$, and $k' = key'[c_{i,j,l}]$;
      $c_{i,j,l+1}$ listens and does: **begin**
        $timer[c_{i,j,l+1}] \leftarrow t'$; $rank'[c_{i,j,l+1}] \leftarrow r'$; $group'[c_{i,j,l+1}] \leftarrow g$;
        $key'[c_{i,j,l+1}] \leftarrow k'$;
      end
      all stations increase clock t;
(* Phase 2 *)
Each station $c_{i,j,1}$ does: $winner[c_{i,j,1}] \leftarrow TRUE$;
for $j \leftarrow 1$ *to* $g(m,i)$ **do**
  $c_{i,j,h(m,i)}$ broadcasts $\langle r' \rangle$ where $r' = rank'[c_{i,j,h(m,i)}]$;
  $c_{i,j,1}$ and (if $j > 1$) $c_{i,j-1,1}$ listen;
  $c_{i,j,1}$ does $rank[c_{i,j,1}] \leftarrow r'$;
  $c_{i,j-1,1}$ (if exists) does: **if** $rank[c_{i,j-1,1}] = r'$ **then** $winner[c_{i,j-1,1}] \leftarrow FALSE$;
  all stations increase clock t;
(* Phase 3 *)
Each station b_l does: **if** $l = 1$ **then** $group[b_l] \leftarrow 0$ **else** $group[b_l] \leftarrow NIL$;
for $l \leftarrow 1$ **to** m **do**
  if *exists* $c_{i,j,1}$ *with* $winner[c_{i,j,1}] = TRUE$ *and* $rank[c_{i,j,1}] = l - 1$ **then**
    $c_{i,j,1}$ broadcasts $\langle j \rangle$;
  b_l listens and does: **if** *there was a message* **then** b_l does $group[b_l] \leftarrow j$;
  all stations increase clock t;
(* Phase 4 *)
for $l \leftarrow 1$ **to** $m - 1$ **do**
  b_l broadcasts $\langle g \rangle$ where $g = group[b_l]$;
  if $group[b_{l+1}] = NIL$ **then** b_{l+1} listens and does: $group[b_{l+1}] \leftarrow g$;
  all stations increase clock t;

Algorithm 3: Procedure Regroup.

$rank'[c_{i,j,1}]$.) In Phase 2 each splitter $c_{i,j,1}$ learns its rank and computes Boolean value $winner[c_{i,j,1}]$. A splitter $c_{i,j,1}$ is a *winner* if it is the last splitter with given $rank$. Time of Phase 2 is $g(m,i)$ and energetic cost is $O(1)$. In Phase 3 each winner $c_{i,j,1}$ informs its successor b' in $\langle b_1, \dots b_m\rangle$ about its block number j (i.e. new group number for b'). The uninformed stations b_l with $l > 1$ end up with $group[b_l] = NIL$. (b_1 ends up with $group[b_1] = 0$ or higher.) The time of Phase 3 is m and energetic cost is $O(1)$. In Phase 4 each b_l with $group[b_l] = NIL$ learns its proper group number from its predecessor. After Phase 4 each station b_l with $group[b_l] = j$, knows that it is ranked somewhere between $c_{i,j,1}$ and $c_{i,j+1,1}$. The time of Phase 4 is $m-1$ and energetic cost is $O(1)$. The time of Regroup is $O(m)$ and the energetic cost is $O(1)$, since the time of each phase is $O(m)$ and energetic cost of each phase is $O(1)$.

```
procedure Rank'(⟨a_1, ..., a_m⟩, ⟨b_1, ..., b_m⟩)
if m ≥ 2 then
    Each a_i does: group[a_i] ← 1;
    for i ← 1 to ⌈l*(m)/2⌉ + 1 do
        Regroup(2i − 1, ⟨a_1, ..., a_m⟩, ⟨b_1, ..., b_m⟩);
        Regroup(2i, ⟨b_1, ..., b_m⟩, ⟨a_1, ..., a_m⟩);
    (* RANK EACH b_j IN ⟨a_1, ..., a_m⟩ *)
    Each station b_j does: rank[b_j] ← 0;
    for i ← 1 to m do
        a_i broadcasts ⟨k⟩, where k = key[a_i];
        each b_j with group[b_j] = ⌈i/2⌉ listens and does:
        if k' < key[b_j] then rank[b_j] ← i;
        all stations increase clock t;
    DO SYMMETRICAL RANKING OF EACH a_j IN ⟨b_1, ..., b_m⟩
else a_1 and b_1 simply compare-exchange their keys
```

Algorithm 4: Procedure Rank'.

We apply Regroup in the procedure Rank' (Algorithm 4) that ranks two sorted sequences of length m in each other in time $O(m \lg^* m)$ with energetic cost $O(\lg^* m)$. Note that in the last iteration of the first "for" loop we have $h(m, 2i-1) = h(m, 2i) = 2$. Thus we only need to rank each element in a block of size 2 of the other sequence. The number of iterations of the first "for" loop is $O(\lg^* m)$, and hence the time of it is $O(m \lg^* m)$ and the energetic cost is $O(\lg^* m)$. The time of the remaining part is $O(m)$ and energetic cost is $O(1)$. By replacing both invocations of Rank in Merge by a single Rank'($\langle a_1, \dots, a_m\rangle$, $\langle b_1, \dots, b_m\rangle$), we obtain an algorithm merging in time $O(m \lg^* m)$ with energetic cost (of both listening and broadcasting) $O(\lg^* m)$.

3 Merge-sort

For simplicity, we assume that $n = 2^k$ for some positive integer k. The stations $c_1, \dots, c_n$ contain initially unsorted sequence of keys $\langle key[c_1], \dots, key[c_n]\rangle$. Merge-Sort (Algorithm 5) sorts the sequence stored in the network. Assume that we apply the first of the

procedure Merge-Sort($\langle c_1, \ldots, c_n \rangle$)
if $m > 1$ **then**
  Merge-Sort($\left\langle c_1, \ldots, c_{n/2} \right\rangle$)
  Merge-Sort($\left\langle c_{n/2+1}, \ldots, c_n \right\rangle$)
  Merge($\left\langle c_1, \ldots, c_{n/2} \right\rangle$,$\left\langle c_{n/2+1}, \ldots, c_n \right\rangle$)

Algorithm 5: Procedure Merge-Sort.

described merging algorithms. The time for merging two sequences of length $n/2$ is $4n/2 = 2n$. On the next level of recursion we have to merge two pairs of sequences of length $n/4$ in time $2 \cdot 4n/4 = 2n$. And so on. The number of levels is $\lg n$, thus the total sorting time is $2n \lg n$. The energetic cost is $\sum_{l=0}^{k-1} (\lceil \lg(2^l + 1) \rceil + 3) = \frac{1}{2} \lg^2 n + \frac{7}{2} \lg n$. For example, for $n = 2^{13} = 8192$, the bounds on time and energetic cost are 212992 and 130, respectively. If we apply the second merging algorithm, then the time of Merge-sort is $O(n \lg n \lg^* n)$ and the energetic cost of listening and broadcasting is $O(\lg n \lg^* n)$.

Remark: The presented algorithms can be parallelized and accelerated $\Omega(k)$ times if we use k channels instead of one, where k is $O(\sqrt{n})$.

Acknowledgment. I would like to thank Mirosław Kutyłowski for helpful comments.

References

1. M. Ajtai, J. Komlós and E. Szemerédi. Sorting in $c \log n$ parallel steps. Combinatorica, Vol. 3, pages 1–19, 1983.
2. K. E. Batcher. Sorting networks and their applications. Proceedings of 32nd AFIPS, pages 307–314, 1968.
3. M. Blum., R. W. Floyd, V. Pratt, R. L. Rivest, and Robert E. Tarjan. Time bounds for selection. Journal of Computer System Sciences, 7(4):448-461, 1973.
4. Th. H. Cormen, Ch. E. Leiserson, R. L. Rivest. Introduction to Algorithms. 1994.
5. T. Jurdziński, M. Kutyłowski, J. Zatopiański. Efficient algorithms for leader election in radio networks. ACM PODC'2002, 51-57, ACM Press.
6. T. Jurdziński, M. Kutyłowski, J. Zatopiański. Energy-Efficient Size Approximation for Radio Networks with no Collision Detection. COCOON'2002, LNCS 2387, (Springer Verlag, Berlin 2002), 279-289,
7. M. Kutyłowski, D. Letkiewicz. Computing average value in ad hoc networks. MFCS'2003, LNCS 2747, (Springer Verlag, Berlin 2003), 511-520,
8. K. Nakano, S. Olariu. Efficient initialization protocols for radio networks with no collision detection. ICPP 2000. IEEE Computer Society Press: New York, 2000: 263-270.
9. K. Nakano, S. Olariu. Broadcast-efficient protocols for mobile radio networks with few channels. IEEE Transactions on Parallel and Distributed Systems, 10:1276-1289, 1999.
10. M. Singh and V. K. Prasanna. Optimal Energy Balanced Algorithm for Selection in Single Hop Sensor Network. SNPA ICC, May 2003.
11. M. Singh and V. K. Prasanna. Energy-Optimal and Energy-Balanced Sorting in a Single-Hop Sensor Network. PERCOM, March 2003.
12. Compendium of Large-Scale Optimization Problems. (DELIS, Subproject 3). `http://ru1.cti.gr/delis-sp3/`