Special sorting algorithms

Raimondas Čiegis

Matematinio modeliavimo katedra, e-mail: rc@vgtu.lt

December 5 d., 2023

In this lecture we consider some special sorting algorithms. They are faster than general algorithms, since an additional information is given on the data.

In this lecture we consider some special sorting algorithms. They are faster than general algorithms, since an additional information is given on the data.

It is clear that even in this case general sorting algorithms, such as QuickSort, can be used. But special algorithms are solving this task much faster.

In this lecture we consider some special sorting algorithms. They are faster than general algorithms, since an additional information is given on the data.

It is clear that even in this case general sorting algorithms, such as QuickSort, can be used. But special algorithms are solving this task much faster.

We know that a complexity of fast general sorting algorithms is defined as $O(N \log N)$.

Our aim is to construct algorithms that have a complexity $\Theta(N)$ even in the worst case. Clearly, such a result can be achieved only for special types of data.

Counting sort algorithm

We are sorting elements with keys k integer numbers not larger than K:

$$1 \leq a_i.key \leq K, \quad i = 1, \ldots, N.$$

Counting sort algorithm

We are sorting elements with keys k integer numbers not larger than K:

$$1 \leq a_i.key \leq K, \quad i = 1, \ldots, N.$$

It is interesting to note that no comparisons are done in this algorithm.

Counting sort algorithm

We are sorting elements with keys k integer numbers not larger than K:

$$1 \leq a_i.key \leq K, \quad i = 1, \ldots, N.$$

It is interesting to note that no comparisons are done in this algorithm.

No comparisons are used and still all elements are sorted!

1a. The array L is initialized, its elements define K single linked lists. Initially all lists are empty.

1a. The array L is initialized, its elements define K single linked lists. Initially all lists are empty.

1b. All elements are transfered into appropriate linked lists according their key values:

$$\textbf{for}\ \ j = 1, \ldots, N$$
$$L[a_j.key].\mathrm{append}(a_j)$$

1a. The array L is initialized, its elements define K single linked lists. Initially all lists are empty.

1b. All elements are transfered into appropriate linked lists according their key values:

for j= 1, . . . , N
  $L[a_j.key]$.append(a_j)

The complexity of this part of the algorithm is equal to $\Theta(N)$.

2. The separate lists are joined into one sorted linked list S

```
for k= 1, ..., K
  S.extend(L[k])
```

2. The separate lists are joined into one sorted linked list S

for k= 1, . . . , K
  S.extend($L[k]$)

Let's analyse the complexity of this step.

2. The separate lists are joined into one sorted linked list S

```
for k= 1, ..., K
    S.extend(L[k])
```

Let's analyse the complexity of this step.

The complexity of puting elements of list $L[k]$ into the sorted list S is equal to $\Theta(|L[k]| + 1)$.

The complexity of saving elements of all K lists is equal to $\Theta(N + K)$.

2. The separate lists are joined into one sorted linked list S

for k= 1, . . . , K
  S.extend($L[k]$)

Let's analyse the complexity of this step.

The complexity of puting elements of list $L[k]$ into the sorted list S is equal to $\Theta(|L[k]| + 1)$.

The complexity of saving elements of all K lists is equal to $\Theta(N + K)$.

The same linear complexity estimate is valid for the full CountingSort algorithm.

If the bound K don't depend on N

or

it can grow, but the following estimate

$$K \leq cN$$

is valid with small c, e.g. $c = 2$,

then the complexity of CountingSort algorithm is linear

$$\Theta(N).$$

What to do if K is growing much faster, e.g. $K = N^3$?

Radix sorting algorithm

For simplicity of presentation we use the decimal numeral system.

Radix sorting algorithm

For simplicity of presentation we use the decimal numeral system.

Other base values b also can be used, e.g. binary $b = 2$ or hexadecimal numbers $b = 16$.

Radix sorting algorithm

For simplicity of presentation we use the decimal numeral system.

Other base values b also can be used, e.g. binary $b = 2$ or hexadecimal numbers $b = 16$.

Let's assume that elements of the set A are integer numbers

$$0 \leqslant a_i < 10^n ,$$

but clearly not all numbers from this interval are necessary included in A.

RadixSort algorithm is a modification of the CountingSort algorithm.

It is a non-comparative sorting algorithm.

RadixSort algorithm is a modification of the CountingSort algorithm.

It is a non-comparative sorting algorithm.

Radix sort avoids comparison by creating and distributing elements into buckets according to their radix (base).

RadixSort algorithm is a modification of the CountingSort algorithm.

It is a non-comparative sorting algorithm.

Radix sort avoids comparison by creating and distributing elements into buckets according to their radix (base).

For elements with more than one significant digit, this bucketing process is repeated for each digit, while preserving the ordering of the prior step, until all digits have been considered.

First we distribute all elements into ten (or b) sub-sets (buckets) according the last (least significant) digit.

First we distribute all elements into ten (or b) sub-sets (buckets) according the last (least significant) digit.

These subsets are combined into one set which is sorted with respect to the least significant digits.

First we distribute all elements into ten (or b) sub-sets (buckets) according the last (least significant) digit.

These subsets are combined into one set which is sorted with respect to the least significant digits.

The obtained set is again distributed and sorted for the next digit.

First we distribute all elements into ten (or b) sub-sets (buckets) according the last (least significant) digit.

These subsets are combined into one set which is sorted with respect to the least significant digits.

The obtained set is again distributed and sorted for the next digit.

This bucketing process is repeated for each digit, until all n digits have been considered.

Note, that the ordering of the prior step is always preserved.

Let's sort the following set of integer numbers ($n = 2$):

$A = (73, 29, 92, 14, 74, 45, 54, 18, 3, 97, 9, 61, 11, 63, 35, 37).$

Let's sort the following set of integer numbers ($n = 2$):

$A = (73, 29, 92, 14, 74, 45, 54, 18, 3, 97, 9, 61, 11, 63, 35, 37)$.

Starting from the rightmost (last) digit, sort the numbers based on that digit:

$$
\begin{aligned}
0 &: \\
1 &: \ 61,\ 11\,, \\
2 &: \ 92 \\
3 &: \ 73,\ 3,\ 63\,, \\
4 &: \ 14,\ 74,\ 54\,, \\
5 &: \ 45,\ 35\,, \\
6 &: \\
7 &: \ 97,\ 37\,, \\
8 &: \ 18\,, \\
9 &: \ 29,\ 9\,.
\end{aligned}
$$

The sub-sets are combined in-order:

$$A = (61, 11, 92, 73, 3, 63, 14, 74, 54, 45, 35, 97, 37, 18, 29, 9).$$

The sub-sets are combined in-order:

$$A = (61,\ 11,\ 92,\ 73,\ 3,\ 63,\ 14,\ 74,\ 54,\ 45,\ 35,\ 97,\ 37,\ 18,\ 29,\ 9).$$

Sorting by the next left digit we get the sub-sets (buckets)

$$\begin{aligned}
0 &: \quad 03,\ 09\,, \\
1 &: \quad 11,\ 14,\ 18\,, \\
2 &: \quad 29\,, \\
3 &: \quad 35,\ 37\,, \\
4 &: \quad 45\,, \\
5 &: \quad 54\,, \\
6 &: \quad 61,\ 63\,, \\
7 &: \quad 73,\ 74\,, \\
8 &: \\
9 &: \quad 92,\ 97\,.
\end{aligned}$$

Combining all ten sub-sets the sorted set is obtained

$$A = (3, 9, 11, 14, 18, 29, 35, 37, 45, 54, 61, 63, 73, 74, 92, 97).$$

Combining all ten sub-sets the sorted set is obtained

$$A = (3,\ 9,\ 11,\ 14,\ 18,\ 29,\ 35,\ 37,\ 45,\ 54,\ 61,\ 63,\ 73,\ 74,\ 92,\ 97).$$

We claim that Radix algorithm is sorting correctly any set of integer numbers.

Combining all ten sub-sets the sorted set is obtained

$$A = (3, 9, 11, 14, 18, 29, 35, 37, 45, 54, 61, 63, 73, 74, 92, 97).$$

We claim that Radix algorithm is sorting correctly any set of integer numbers.

It is sufficient to consider a case of two digits numbers.

The proof for general n-digits case can be done by using the mathematical induction method.

Let's consider two digit numbers X and Y:

$$X = 10\,a + b, \quad Y = 10\,c + d, \quad 0 \leqslant a, b, c, d \leqslant 9.$$

Let's consider two digit numbers X and Y:

$$X = 10\,a + b, \quad Y = 10\,c + d, \quad 0 \leqslant a, b, c, d \leqslant 9.$$

Inequality $X < Y$ is valid, if

$$(a < c) \quad \text{or} \quad (a = c)\,\&\,(b < d).$$

Let's consider two digit numbers X and Y:

$$X = 10\,a + b, \quad Y = 10\,c + d, \quad 0 \leqslant a, b, c, d \leqslant 9.$$

Inequality $X < Y$ is valid, if

$$(a < c) \quad \text{or} \quad (a = c)\,\&\,(b < d).$$

If $a < c$, then at the second step of Radix sort algorithm X is distributed into a bucket with smaller order number than a bucket to which Y is distributed.

Let's consider two digit numbers X and Y:

$$X = 10\,a + b, \quad Y = 10\,c + d, \quad 0 \leqslant a, b, c, d \leqslant 9.$$

Inequality $X < Y$ is valid, if

$$(a < c) \quad \text{or} \quad (a = c)\,\&\,(b < d).$$

If $a < c$, then at the second step of Radix sort algorithm X is distributed into a bucket with smaller order number than a bucket to which Y is distributed.

If $(a = c)\,\&\,(b < d)$, then at the first step of Radix sort algorithm X is distributed into a bucket with a smaller order number than Y.

After the second step both elements will be distributed into the same bucket, but X will be distributed before Y.

Let's consider two digit numbers X and Y:

$$X = 10\,a + b, \quad Y = 10\,c + d, \quad 0 \leqslant a, b, c, d \leqslant 9.$$

Inequality $X < Y$ is valid, if

$$(a < c) \quad \text{or} \quad (a = c)\,\&\,(b < d).$$

If $a < c$, then at the second step of Radix sort algorithm X is distributed into a bucket with smaller order number than a bucket to which Y is distributed.

If $(a = c)\,\&\,(b < d)$, then at the first step of Radix sort algorithm X is distributed into a bucket with a smaller order number than Y.

After the second step both elements will be distributed into the same bucket, but X will be distributed before Y.

Thus in both cases Radix sort correctly these numbers.

Complexity of the Radix sort algorithm

Let's count basic operations when N integer numbers are sorted and they are written in b base format.

We assume that the following bound is valid for the values of these numbers (in the decimal numeral system)

$$1 \le k \le K.$$

Complexity of the Radix sort algorithm

Let's count basic operations when N integer numbers are sorted and they are written in b base format.

We assume that the following bound is valid for the values of these numbers (in the decimal numeral system)

$$1 \leq k \leq K.$$

It follows from the complexity analysis of CountSort algorithm that

for one step of Radix sort algorithm $\Theta(N + b)$ operations are done.

The number of steps is equal to $n = \log_b K$, thus total cost of Radix sort algorithm is given by

$$\Theta\big((N + b) \log_b K\big).$$

It is easy to compute that the optimal base value is $b = N$, then

$$\Theta\big(N \log_N K\big).$$

It is easy to compute that the optimal base value is $b = N$, then

$$\Theta\big(N \log_N K\big).$$

Let us reconsider the previous example of $K = N^3$.

Then $\log_N K = 3$ and the complexity of Radix sort algorithm is linear again

$$\Theta(N).$$

External sorting

External sorting is required when the data being sorted do not fit into the main memory of a computing device (RAM)

and

instead they must reside in the slower external memory, usually a disk drive.

External merge sort typically uses a hybrid sort-merge strategy.

External merge sort typically uses a hybrid sort-merge strategy.

In the sorting phase, chunks of data small enough to fit in main memory are read, sorted, and written out to a temporary file.

External merge sort typically uses a hybrid sort-merge strategy.

In the sorting phase, chunks of data small enough to fit in main memory are read, sorted, and written out to a temporary file.

In the merge phase, the sorted subfiles are combined into a single larger file.

We sort N data elements and they are written in external file F.

Let's assume that only M elements fit into main memory.

- Chunks of size M are read from F and sorted by using some fast sorting algorithm (e.g. Quicksort).

  These chunks are written in turn to temporary files $F1, F2$.

We sort N data elements and they are written in external file F.

Let's assume that only M elements fit into main memory.

- Chunks of size M are read from F and sorted by using some fast sorting algorithm (e.g. Quicksort).

  These chunks are written in turn to temporary files $F1, F2$.
- In the merge phase, the sorted chunks of M length from files $F1, F2$ are combined into single chunks of $2M$ length and are written in turn to temporary files $F3, F4$

We sort N data elements and they are written in external file F.

Let's assume that only M elements fit into main memory.

- Chunks of size M are read from F and sorted by using some fast sorting algorithm (e.g. Quicksort).

  These chunks are written in turn to temporary files $F1, F2$.
- In the merge phase, the sorted chunks of M length from files $F1, F2$ are combined into single chunks of $2M$ length and are written in turn to temporary files $F3, F4$
- This merge procedure is repeated till one sorted file of length N is obtained.

Example

We have a set of data saved in file F, the length of it is equal to $N = 29$:

$$(4, 5, 2, 8, 4, 1, 7, 9, 2, 3, 0, 3, 8, 6, 2, 4, 9, 3, 9, 5, 0, \\ 4, 6, 2, 5, 3, 5, 1, 0).$$

Example

We have a set of data saved in file F, the length of it is equal to $N = 29$:

$$(4,\ 5,\ 2,\ 8,\ 4,\ 1,\ 7,\ 9,\ 2,\ 3,\ 0,\ 3,\ 8,\ 6,\ 2, 4,\ 9,\ 3,\ 9,\ 5,\ 0,\\ 4,\ 6,\ 2,\ 5,\ 3,\ 5,\ 1,\ 0).$$

Let us assume that $M = 3$, then the first sorting step is implemented in the following way:

$$M = 3:$$
$$\mathrm{F1} = (\,2,\ 4,\ 5 \mid 2,\ 7,\ 9 \mid 2,\ 6,\ 8 \mid 0,\ 5,\ 9 \mid 3,\ 5,\ 5\,)$$
$$\mathrm{F2} = (\,1,\ 4,\ 8 \mid 0,\ 3,\ 3 \mid 3,\ 4,\ 9 \mid 2,\ 4,\ 6 \mid 0,\ 1\,)$$

Then the merging steps are implemented

$$M = 6:$$

$$\text{F3} = (\,1,\ 2,\ 4,\ 4,\ 5,\ 8 \mid 2,\ 3,\ 4,\ 6,\ 8,\ 9 \mid 0,\ 1,\ 3,\ 5,\ 5\,)$$

$$\text{F4} = (\,0,\ 2,\ 3,\ 3,\ 7,\ 9 \mid 0,\ 2,\ 4,\ 5,\ 6,\ 9\,)$$

$$M = 12:$$

$$\text{F1} = (\,0,\ 1,\ 2,\ 2,\ 3,\ 3,\ 4,\ 4,\ 5,\ 7,\ 8,\ 9 \mid 0,\ 1,\ 3,\ 5,\ 5\,)$$

$$\text{F2} = (\,0,\ 2,\ 2,\ 3,\ 4,\ 4,\ 5,\ 6,\ 6,\ 8,\ 9,\ 9\,)$$

$$M = 24:$$

$$\text{F3} =$$

$$\text{F4} =$$

$$M = 29:$$

$$\text{F} =$$

Let us estimate the ammount of data transfered from external memory to internal memory and back.

Let us estimate the ammount of data transfered from external memory to internal memory and back.

These transfers between internal and external memory make the main part of running time.

Let us estimate the ammount of data transfered from external memory to internal memory and back.

These transfers between internal and external memory make the main part of running time.

For simplicity of analysis we assume that $N = 2^k M$.

At each stage N/M packets are transfered between internal and external memory.

The number of stages is $(k+1)$ thus the total number of transfered packets is equal to

$$\frac{N}{M} \log \left(\frac{N}{M} \right).$$