Laboratorio di Algoritmi e Strutture Dati (Laboratory of Algorithms and Data Structures)

Guido Fiorino
guido.fiorino@unimib.it

Divide and conquer on arrays
(Last Modified: 20-03-2017)

Maximum in a sequence

Given a sequence $\langle A_1, \ldots, A_N \rangle$ find the maximum.
The maximum is in the first or in the second half of the sequence. If the sequence has one element we immediately have the answer.
We can proceed recursively following a divide-and-conquer approach:

- Base case: if $N = 1$, then A_1 is the maximum;
- Recursion:
  1. recursively compute the maximum M_1 of the sequence $\langle A_1, \ldots, A_{\frac{N}{2}} \rangle$;
  2. recursively compute the maximum M_2 of the sequence $\langle A_{\frac{N}{2}+1}, \ldots, A_N \rangle$;
  3. the maximum of $\langle A_1, \ldots, A_N \rangle$ is the maximum between M_1 and M_2.

Exercise (method `maximum`)

write a method `int maximum(int[] a, int l, int r)` *that finds the maximum in the array* `a` *spanning from* `l` *to* `r`.

Average of a sequence

Given a sequence $\langle A_1, \ldots, A_N \rangle$ find the average.
If we know the average of $\langle A_1, \ldots, A_{\frac{N}{2}} \rangle$ and $\langle A_{\frac{N}{2}+1}, \ldots, A_N \rangle$ we can compute the average of the whole sequence by appropriately combining the two values. If the sequence has one element, then we immediately have the answer.
We can proceed recursively following a divide-and-conquer approach:

- Base case: if $N = 1$, then A_1 is the average of the sequence;
- Recursion:
  1. recursively compute the average of the sequence $\langle A_1, \ldots, A_{\frac{N}{2}} \rangle$, let Av_1 be the value;
  2. recursively compute the average of the sequence $\langle A_{\frac{N}{2}+1}, \ldots, A_N \rangle$, let Av_2 be the value;
  3. the average of $\langle A_1, \ldots, A_N \rangle$ is the $\frac{Av_1 * \frac{N}{2} + Av_2 * (N - \frac{N}{2})}{N}$

Exercise (method `avg`)

Write a method `double avg(int[] a, int l, int r)` *that finds the average in the array* `a` *spanning from* `l` *to* `r`.

Palindromes

A sequence $\langle A_1, \ldots, A_N \rangle$ that reads the same backward as forward is palindrome.
This definition can be stated recursively as follows:

> a sequence $\langle A_1, \ldots, A_N \rangle$ is palindrome if $A_1 = A_N$ and $\langle A_2, \ldots, A_{N-1} \rangle$ is palindrome.

What about the base case? We have two base cases: the empty sequence and the sequence with one element.

Definition (palindrome)

A sequence $\langle A_1, \ldots, A_N \rangle$ is palindrome iff

- the sequence is empty (equivalently $N = 0$);
- the sequence contains one element (equivalently $N = 1$);
- $A_1 = A_N$ and the sequence $\langle A_2, \ldots, A_{N-1} \rangle$ is palindrome.

Palindromes

Exercise (method `isPalindrome`)

Write a method `boolean isPalindrome(int[] X, int L, int R)` *that returns* `true` *if the array a that spans between l and r is palindrome,* `false` *otherwise.*

Reverse of an array

The reverse of a sequence $\langle A_1, \ldots, A_N \rangle$, denoted as $rev(\langle A_1, \ldots, A_N \rangle)$, is the sequence $\langle A_N, \ldots, A_1 \rangle$.
The definition emphasizes that function *rev* returns a particular permutation of a given sequence. Thus function rev maps sequences into sequences.
Function rev has an easy recursive definition:

$$rev(\langle A_1, \ldots, A_N \rangle) = \begin{cases} \langle A_1 \rangle & \text{if } N = 1 \\ \langle\, rev(\langle A_2, \ldots, A_n \rangle)\,, A_1 \,\rangle & \text{otherwise} \end{cases}$$

Another recursive definition is based on reversing separately the two halves of the array. The results must be combined carefully:

$$rev(\langle A_1, \ldots, A_N \rangle) = \begin{cases} \langle A_1 \rangle & \text{if } N = 1 \\ \langle\, rev(\langle A_{\frac{n}{2}+1}, \ldots, A_n \rangle), rev(\langle A_1, \ldots, A_{\frac{n}{2}} \rangle) \,\rangle & \text{otherwise} \end{cases}$$

Exercise (method rev)

Write a method `int[] rev(int[]a,int i, int n)` *that returns the reverse of the array* `a` *that spans from* `i` *to* `n`.

MergeSort

Given a sequence $\langle A_1, \ldots, A_N \rangle$, proceed recursively following the divide-and-conquer approach:

- Base case: if $N = 1$, then, the sequence is sorted, return $\langle A_1 \rangle$;
- Recursion:
  1. recursively sort $\langle A_1, \ldots, A_{\frac{N}{2}} \rangle$;
  2. recursively sort $\langle A_{\frac{N}{2}+1}, \ldots, A_N \rangle$;
  3. return the result of merging the two sorted subsequences obtained in the previous steps.

The implementation of MergeSort depends on the implementation of Phase 3.

There are different techniques to perform the merge. The aim is to save time.

Listing all permutations of a sequence

There are $N!$ permutations of a given a sequence $\langle A_1, \ldots, A_N \rangle$, where all the elements are distinct, that is for every A_i, A_j of the sequence, $A_i \neq A_j$ if $i \neq j$.

We can describe the set of all permutations of $\langle A_1, \ldots, A_N \rangle$ by a recursive definition that follows a divide-and-conquer approach:

> if we have all the permutations of $\langle A_2, \ldots, A_N \rangle$, we are able to build all the permutations of $\langle A_1, \ldots, A_N \rangle$ by inserting A_1 in every possible position inside every given permutation of $\langle A_1, \ldots, A_N \rangle$.

Listing all permutations of a sequence

In detail:

- Base case: if $N = 1$, then $\langle A_1 \rangle$ is the only permutation;
- Recursion:
  1. recursively compute the set S of permutations of the sequence $\langle A_2, \ldots A_N \rangle$;
  2. for every sequence $\langle A_{f(2)}, \ldots, A_{f(N)} \rangle$ in S, where $f : \{2, \ldots, n\} \to \{2, \ldots, n\}$, build

     the N new subsequences
     $$\begin{cases} \langle A_1, A_{f(1)}, \ldots, A_{f(N)} \rangle, \\ \langle A_{f(1)}, A_1, A_{f(2)}, \ldots, A_{f(N)} \rangle, \\ \vdots \\ \langle A_{f(1)}, \ldots, A_{f(N)}, A_1 \rangle. \end{cases}$$

     This is the set of all the permutations of $\langle A_1, \ldots, A_N \rangle$.

Listing all permutations of a sequence

Exercise

Use the description given in the previous page to write a Java method that returns or prints the permutations of a given sequence. You are free to choose the signature of the method. As an example:

```
int[][] listPerms(int[] seq)
```