Linköping University
Department of Science and Technology/ITN
Aida Nordman

TND004: Data Structures

Lab 1

Goals

- To use recursion versus iteration.
- To use divide-and-conquer algorithm design technique.
- To analyse the time and space complexity of algorithms.
- To compare the efficiency of different algorithms that solve the same problem.

The stable partition problem

Consider the following boolean function declaration

```
bool p(int i);
```

and a sequence `S` of values (e.g. integers). The stable partition of sequence `S`, using `p`,

```
stable_partition(S, p);
```

is a function that rearranges the elements in `S`, in such a way that all the elements for which `p` returns `true` precede all those for which it returns `false`, and the relative order of elements within each group is preserved.

For example, assume `S=<3, 4, 1, 2, 5, 6, 7, 8, 9>` and consider the boolean function `even` to test whether an integer is even.

```
bool even(int i)
{
    return i % 2 == 0;
}
```

Then, `stable_partition(S, even)` rearranges the items in `S` as follows.

`S=<4, 2, 6, 8, 3, 1, 5, 7, 9>`

Note that `<2, 4, 6, 8, 3, 1, 5, 7, 9>` is not a stable partition of `S` because both `2` and `4` are even numbers and they appear in swapped order when compared with the original sequence.

The stable partition problem is used to solve many practical problems. Due to its relevance, the C++ standard library contains an implementation of it, `std::stable_partition`.

In this lab, you are not going to use `std::stable_partition`. Instead, you are going to make your own implantation, using two different algorithms that solve the stable partition problem.

- The first algorithm is iterative (i.e. recursion should not be used) and executes in $O(n)$ time[1], for any sequence with $n > 0$ items.

[1] In other words, a linear algorithm.

- The second algorithm uses a divide-and-conquer strategy.

This lab consists of three exercises.

- **Exercise1**: to create and implement an iterative linear algorithm[2] that solves the stable partition problem of a sequence. Note that this exercise is part of the preparation tasks to be done before the **HA** lab session on week 13.
- **Exercise 2**: to implement the divide-and-conquer algorithm described here. This exercise must be ready to present on the **RE** lab session of week 14.
- **Exercise 3**: to analyze the running time and space usage of both algorithms. You must deliver written solutions for this exercise to your lab assistant not later than the end of week 14.

For deadlines and how to deliver your solutions, please read this section.

Preparation

A list of tasks that you need to do before the **HA** lab session on week 13 is given below.

1. Review lecture 1 and lecture 2.
2. Download the zipped folder with the files needed for this lab. It is possible to compile and create an executable from the given source file `Exerc1.cpp`, though not all functions are fully implemented. Note that by now the sequences displayed are the input sequences.
3. Design an **iterative linear time algorithm** that creates a stable partition of a given sequence and then implement it in the function `stable_partition_iterative`.
4. Test whether your code compiles and runs with the `main` given in `Exerc1.cpp`. Compare the output obtained with the expected output which is available in the file `exerc1_out.txt`. You can disregard by now the output for the divide-and-conquer algorithm implementation.

   When you have successfully completed steps 3 and 4 then you have also done **exercise 1** in this lab.
5. Read and understand the divide-and-conquer algorithm to stable partition a sequence, described below.
6. Simulate manually in a piece of paper the execution of the algorithm for the following sequences:
   - `S=<3, 3>`
   - `S=<1, 2, 3, 4, 5, 6, 7, 8, 9>`

In the beginning of the **HA** lab session on week 13, the lab assistant will give feedback on your iterative algorithm to solve the stable partition problem.

[2] If a sequence has $n > 0$ items then a linear time algorithm executes $O(n)$ steps, in the worst-case.

A divide-and-conquer algorithm for the stable partition

Divide-and-conquer is a common technique to design algorithms. Divide-and-conquer algorithms consist of two parts.

- **Divide**: divide the problem into two or more smaller sub-problems, with exception for the base cases. Each of these smaller sub-problems are then solved recursively.
- **Conquer**: The solutions of the smaller sub-problems are then put together to create a solution for the original problem.

A divide-and-conquer algorithm to solve the stable partition problem is described below. The figures in the appendix help to illustrate the algorithm.

1. **Divide**: divide the sequence $S = < v_1, \dots, v_n >$, with $n > 1$, in two halves: $S_L = < v_1, \dots, v_{mid-1} >$ and $S_R = < v_{mid}, \dots, v_n >$. Then, apply stable partition recursively to S_L and S_R. Empty sequences or one-item sequences form the base cases.
2. **Conquer**: use the C++ `std::rotate` to place the S_R-block of items with property `p` (e.g. to be an even number) just after the S_L-block of items with the same property.

Exercise 2 of this lab consists in understanding and then implementing the algorithm described above. You should add your code for this exercise to the function `stable_partition` in the file `Exerc1.cpp`. Then, compile and run the program. Compare the output obtained with the expected output which is available in the file `exerc1_out.txt`.

Algorithms analysis: time and space requirements

Exercise 3 requires that you analyze the running time and space usage of both algorithms you have implemented. Use Big-Oh notation and **motivate clearly** your answers.

1. Analyze the time and space complexity of the iterative algorithm.
2. Analyze the time and space complexity of the divide-and-conquer algorithm.
3. Compare both algorithms. In which situations would you prefer to use the iterative algorithm (if any)? In which situations would you prefer to use the divide-and-conquer algorithm (if any)?

Remember to write your answers in paper, preferably computer typed, and do not forget to indicate the name plus LiU login of each group member. You must deliver your written answers to the lab assistant, when you present the lab on week 14.

Presenting solutions and deadlines

You must demonstrate your solution orally during your **RE** lab session on **week 14**. Note that this lab has a strict deadline. Failing to present the lab on week 14 implies that you cannot be awarded 3p for the labs in the course.

Necessary requirements for approving your lab are given below.

- Use of global variables is not allowed, but global constants are accepted.
- Readable and well-indented code. Note that complicated functions and over-repeated code make code quite unreadable and prone to bugs.
- Compiler warnings they may affect badly the program execution are not accepted, though the code may pass the given tests. See course website for how to set Visual Studio compiler warning level.
- The iterative algorithm to solve stable partition problem must execute in linear time.
- The `std::stable_partition` algorithm cannot be used to solve the exercises in this lab.
- Leave your written solutions for exercise 3 to the lab assistant not later than the end of week 14, with indication of the name plus LiU login of each group member. You'll need to discuss your answers with the lab assistant who in turn will give you feedback. Hand written answers that we cannot understand are simply ignored.

If you have any specific question about the exercises, then send us an e-mail or drop in during office hours. Be short and concrete, otherwise you won't get a quick answer. You can write your e-mail in Swedish. Add the course code to the e-mail's subject, i.e. "**`TND004:`** …".

Recall that presenting your solution to this lab exercises can only be done in the **RE** session of week 14.

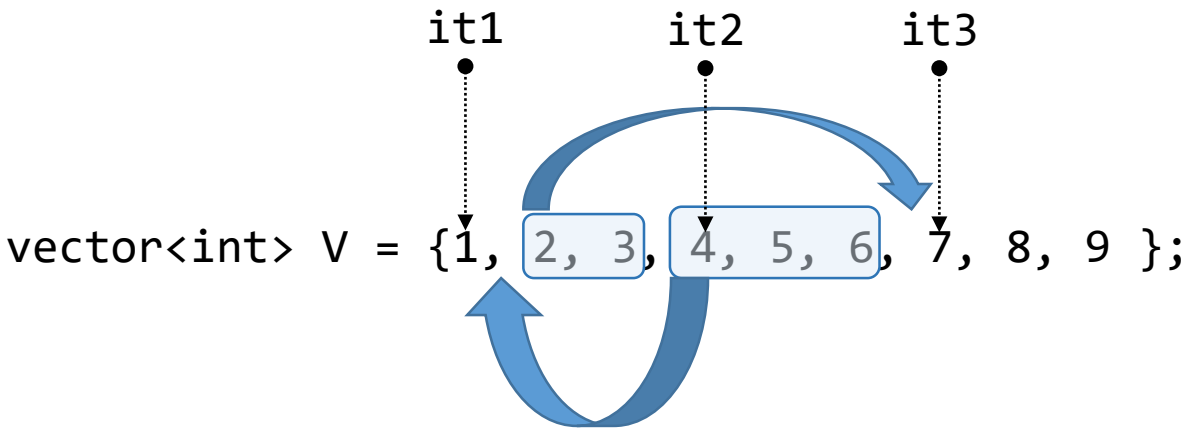
Appendix

In the next pages you can find some figures.

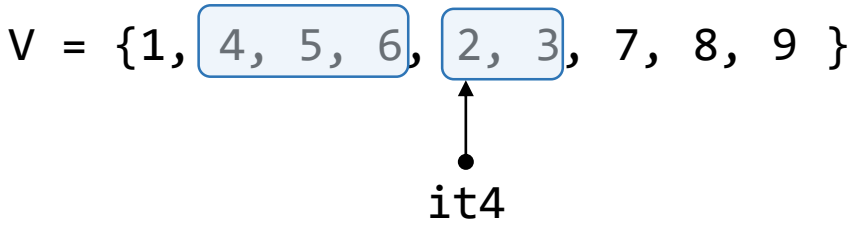
- The first figure illustrates how C++ `std::rotate` function works. Please, consult the online library documentation for more details. Recall that `std::rotate` is used in the implementation of the divide-and-conquer algorithm.
- The remaining figures illustrates how the divide-and-conquer algorithm for the stable partition problem of a sequence `S` works.

Lycka till !

std::rotate




```
vector<int>::iterator it1 = begin(V) + 1;
vector<int>::iterator it2 = begin(V) + 3;
vector<int>::iterator it3 = begin(V) + 6;

auto it4 = std::rotate(it1, it2, it3);
```

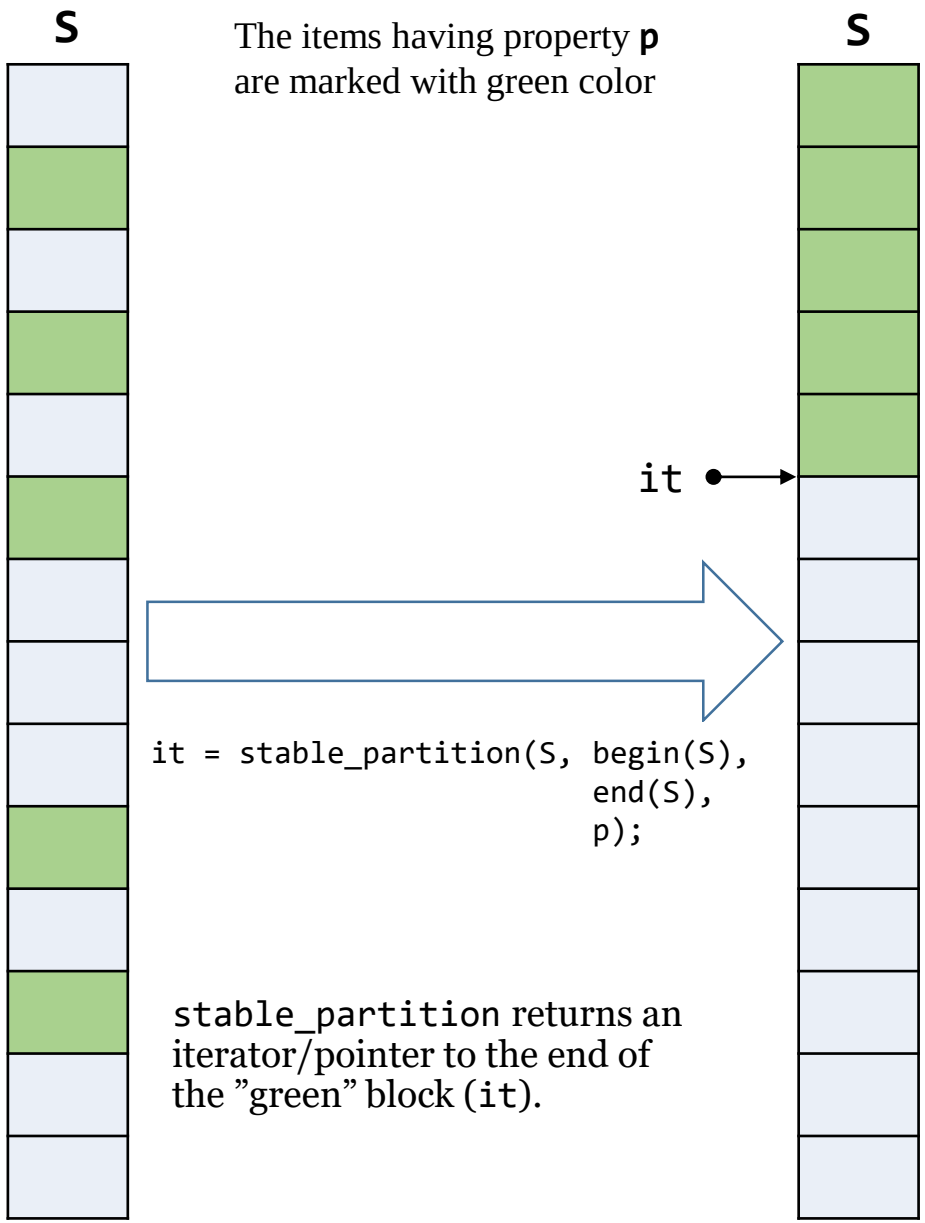


Note: if `it2` and `it3` point to the same item in the sequence then `std::rotate` does not modify `V`

Divide-and-conquer: stable_partition

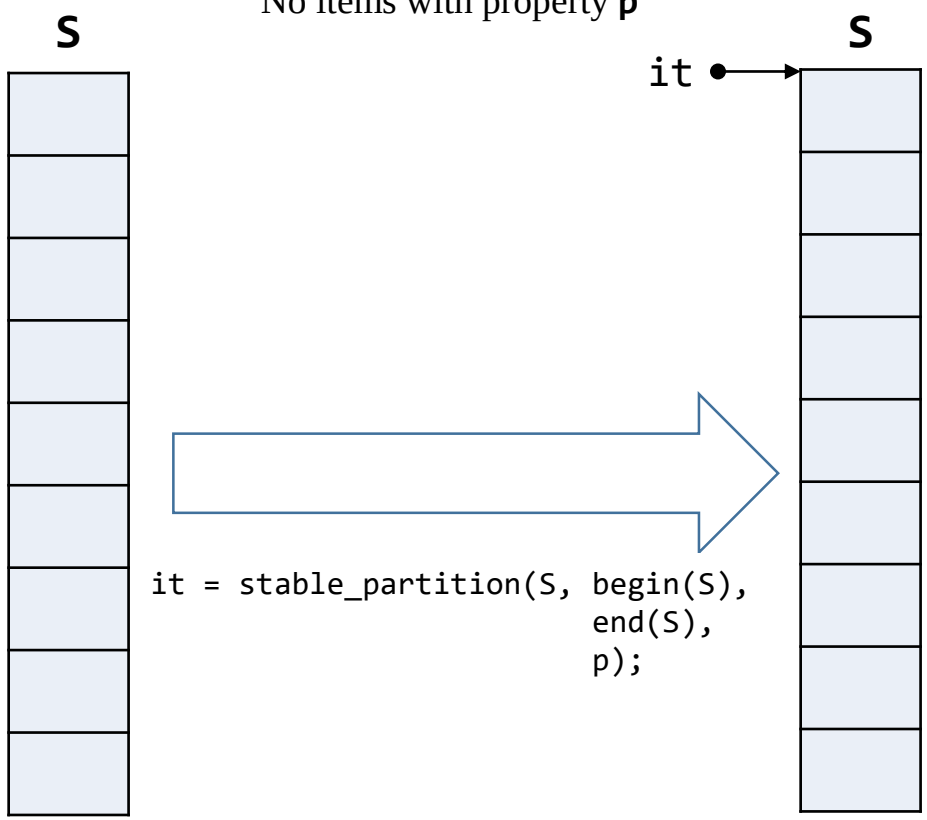
```
void stable_partition(vector<int>& S, Test p)
{
	//call auxiliar function
	stable_partition(S, begin(S), end(S), p);
}


//Divide-and-conquer algorithm: stable-partition the sub-sequence in S starting at first and ending at last-1
//If there are items with property p then return an iterator to the end of the block containing the items with property p
//If there are no items with property p then return first
vector<int>::iterator stable_partition(vector<int>& S,
                                       vector<int>::iterator first, vector<int>::iterator last,
                                       Test p)
{
	//IMPLEMENT
}
```

Divide-and-conquer: `stable_partition`



The items having property **p** are marked with green color

`stable_partition` returns an iterator/pointer to the end of the ”green” block (`it`).



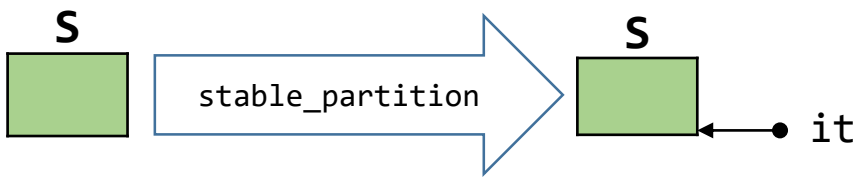
No items with property **p**

`stable_partition` returns an iterator/pointer to the beginning of the sequence (`it`).

Base-case: sequence with one item

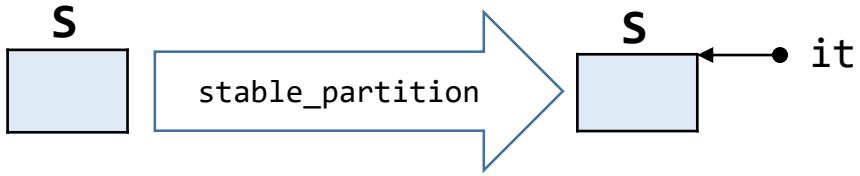
```
it = stable_partition(S, begin(S), end(S), p);
```

Item has property **p** (marked with green color)



`stable_partition` returns an iterator/pointer to the end of the ”green” block (`it`).

Item does not have property **p**



`stable_partition` returns an iterator/pointer to the beginning of the sequence (`it`).

Divide-and-conquer: `stable_partition`