Tampereen yliopisto

STL algorithms

COMP.CS.300 Data structures and algorithms 1

Matti Rintala (matti.rintala@tuni.fi)

STL algorithms

- **Lots** of ready-made algorithms
- Have a look at the algorithm list (like cppreference.com)
- Algorithms take iterators as params
- Algorithms **never** add/remove elements! (invalidation)
- (There are also parallel versions of algorithms)

Tampereen yliopisto

STL algorithms - examples

COMP.CS.300 Data structures and algorithms 1

Matti Rintala (matti.rintala@tuni.fi)

Selection-Sort(*A*)

```
1 for next_elem := 1 to A.length-1 do
2       smallest := next_elem
3       for place := next_elem+1 to A.length do
4             if A[place] < A[smallest] then
5                   smallest := place
6       A[next_elem] ⇄ A[smallest]
```

(move the border between parts)
(possible location of smallest = first elem)

(if an even smaller elem is found...)
(...take not of its location)
(swap the smallest to the beginning)

Mergesort(*A*, *left*, *right*)

```
1 if left < right then                  (nothing to be done in trivial case)
2     mid := ⌊(left + right) / 2⌋       (calculate midpoint)
3     Mergesort(A, left, mid )          (sort left half)
4     Mergesort(A, mid+1, right)        (sort right half)
5     Merge(A, left, mid, right)        (merge sorted halfs together)
```

Insertion-Sort(*A*)

```
1 for next_elem := 2 to A.length do
2       key := A[next_elem]
3       place := next_elem − 1
4       while place > 0 and A[place] > key do
5             A[place + 1] := A[place]
6             place := place − 1
7       A[place + 1] := key
```

(move the border between parts)
(the next element to be handled)

(find the right place for next element)
(make space for the element)

(put the element into the correct place)

Tampereen yliopisto

Tuning STL algorithms and containers

COMP.CS.300 Tietorakenteet ja algoritmit 1

Matti Rintala (matti.rintala@tuni.fi)

Tuning STL

- Many algorithms takes as a parameter a lambda/function, which the algorithm uses (=calls)
- For example sort & comparing elems, find_if what kind of elem to look for...
- The behaviour of some containers can also be tuned
- For example map and order of search keys

Quicksort(*A*, *left*, *right*)

```
1 if left < right then                    (nothing to be done in trivial case)
2     pivot := Partition(A, left, right) (partition to small & large, pivot marks split)
3     Quicksort(A, left, pivot−1 )       (sort smaller than pivot)
4     Quicksort(A, pivot+1, right)       (sort larger than pivot)
```