Sorting Algorithms

Sorting Algorithms

- Sorting algorithms
  - Overview
  - Examples
- STL sort
  - standard operator
  - function pointer
  - function object

Overview

No sorting algorithm is perfect!

Performance depends on:
- Input datastructure
- Input order
- Input size

Performance can be judged by:
- Number of comparisons (average, best, worst)
- Number of swaps
- Space requirement
- Stability (order of items with the same key)

A task specific algorithm has to be chosen,
or a combination of multiple algorithms

Overview

Common algorithms

Name	Time (best - worst)	Stability
Bubble Sort	$O(n^2)$	Stable
Selection Sort	$O(n)$ - $O(n^2)$	Stable
Insertion Sort	$O(n)$ - $O(n^2)$	Stable
Heap Sort	$O(n*\log(n))$	Instable
Merge Sort	$O(n*\log(n))$	Stable
Quick Sort	$O(n*\log(n))$ - $O(n^2)$	Instable
Tim Sort	< $O(n*\log(n))$ - $O(n*\log(n))$	Stable

Example

Why would I choose a less efficient algorithm?

Overhead due to number of swaps, extra space or other!

Example Quicksort:

- Choosing a fixed index of the partition as pivot element causes bad performance on nearly sorted lists
→ intelligent choice of pivot causes overhead

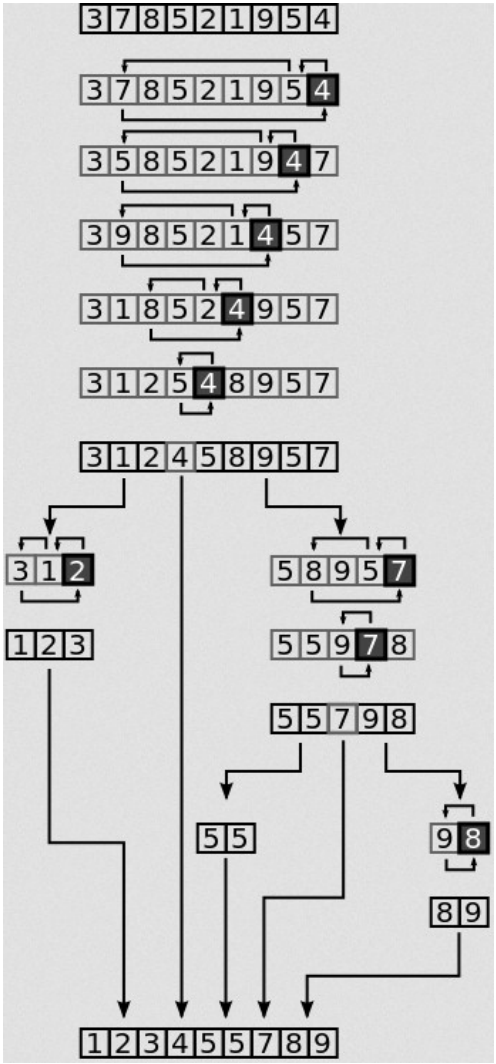
- Recursive call of quicksort requires O(n) extra space
→ can be optimized to O(log(n)) but causes overhead

Example Selection Sort:

- A minimum number of swaps will be done for nearly sorted lists, but a huge overhead of comparisons

Linked List

Quicksort: 24 insertions

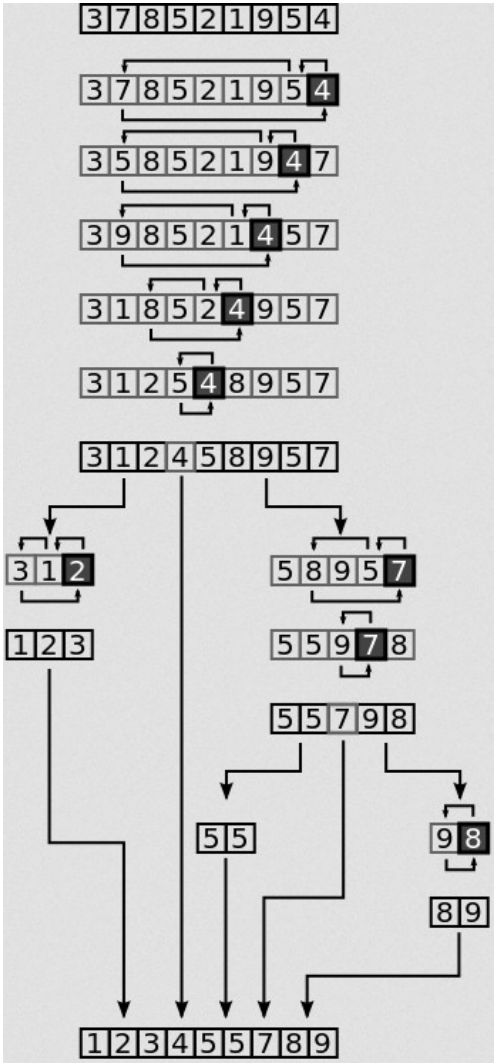


Insertion Sort: 6 insertions

3 7 8 **5** 2 1 9 5 4
3 7 **5** 8 2 1 9 5 4
3 5 7 8 **2** 1 9 5 4
2 3 5 7 8 **1** 9 5 4
1 2 3 5 7 8 9 **5** 4
1 2 3 5 5 7 8 9 **4**
1 2 3 4 5 5 7 8 9

Array

Quicksort: 15 swaps



Insertion Sort: 19 swaps

3 7 8 **5** 2 1 9 5 4
3 7 **5** 8 2 1 9 5 4
3 5 7 8 **2** 1 9 5 4
2 3 5 7 8 **1** 9 5 4
1 2 3 5 7 8 9 **5** 4
1 2 3 5 5 7 8 9 **4**
1 2 3 4 5 5 7 8 9

std::sort

Can be used to quickly implement a sorting operator using user defined keys and constraints.

The standard operator is the *less than* operator, which can be globally redefined:

```
#include <vector>
#include <algorithm>

class Box
{
  int width, heigth, depth;
};

inline bool operator<(const Box& a, const Box& b)
{
  return a.width < b.width;
}

int main()
{
  std::vector<Box> boxes = fillVectorWithRandomBoxes();
  std::sort(boxes.begin(),boxes.end());
  // boxes are now ordered by their width
  return 0;
}
```

The same can be done within a class to define a member specific *less than* operator

std::sort

For more a more complex behavior, a comparison function can be supplied by a function pointer:

```
bool GreaterWidth(const Box& a, const Box& b)
{
  if(a.width == b.width)
    return a.heigth < b.heigth;
  return a.width > b.width;
}

int main()
{
  std::vector<Box> boxes = fillVectorWithRandomBoxes();
  std::sort(boxes.begin(),boxes.end(), GreaterWidth);
  // boxes are now ordered by their width
  // and height
  return 0;
}
```

std::sort

It may be desireable to maintain a state within the comparison function. Use a Function Object:

```
struct GreaterWidth : public std::binary_function<Box,Box,bool>
{
  int maxWidth = 0;
  // overload the () operator
  inline bool operator()(const Box& a, const Box& b)
  {
    if(a.width > maxWidth)
      maxWidth = a.width;
    if(b.width > maxWidth)
      maxWidth = b.width;

    if(a.width == b.width)
      return a.heigth < b.heigth;
    return a.width > b.width;
  }
};

int main()
{
  std::vector<Box> boxes = fillVectorWithRandomBoxes();
  std::vector<Box> moreBoxes = fillVectorWithRandomBoxes();
  GreaterWidth compare;
  std::sort(boxes.begin(),boxes.end(), compare);
  std::sort(moreBoxes.begin(),moreBoxes.end(), compare);
  // boxes are now ordered by their width
  // and height
  // compare.maxWidth contains the maximum width of both vectors
  return 0;
}
```

std::*sort*

Other algorithms are based on different sorting strategies:

- stable_sort: maintains the original order of equal elements
- partial_sort: orders a subrange of the input set
- qsort: executes a quicksort

Thank you for your attention!

Further information on www.dkfz.de

dkfz. GERMAN CANCER RESEARCH CENTER IN THE HELMHOLTZ ASSOCIATION

50 Years – Research for A Life Without Cancer