Lecture 4: Sorting

Steven Skiena

Department of Computer Science
State University of New York
Stony Brook, NY 11794–4400

http://www.cs.sunysb.edu/∼skiena

Applications of Sorting

The key to understanding sorting is seeing how it can be used to solve many important programming tasks:

- *Uniqueness Testing* – How can we test if the elements of a given collection of items S are all distinct?
- *Deleting Duplicates* – How can we remove all but one copy of any repeated elements in S?
- *Median/Selection* – How can we find the kth largest item in set S?
- *Frequency Counting* – Which is the most frequently occurring element in S, i.e., the mode?

- *Reconstructing the Original Order* – How can we restore the original arrangement of a set of items after we permute them for some application?
- *Set Intersection/Union* – How can we intersect or union the elements of two containers?
- *Finding a Target Pair* – How can we test whether there are two integers $x, y \in S$ such that $x + y = z$ for some target z?
- *Efficient Searching* – How can we efficiently test whether element s is in set S?

Basic Sorting Algorithms

What are the time complexities and number of data moves for each algorithm?

- *Selection Sort* – This algorithm splits the input array into sorted and unsorted parts, and with each iteration finds the smallest element remaining in the unsorted region and moves it to the end of the sorted region.
- *Heapsort* – Selection sort with the right data structure!
- *Bubble sort* – Percolate elements forward through adjacent element swaps. No element is guaranteed to be in the proper position until the end.

- *Insertion Sort* – This algorithm also maintains sorted and unsorted regions of the array. In each iteration, the next unsorted element moves up to its appropriate position in the sorted region.
- *Quicksort* – This algorithm reduces the job of sorting one big array into the job of sorting two smaller arrays by performing a *partition* step. The partition separates the array into those elements that are less than the pivot/divider element, and those which are strictly greater than this pivot/divider element.

Multicriteria Sorting

How can we break ties in sorting using multiple criteria, like sorting first on last name breaking ties on first name?

One way is to use a complicated comparison function that combines all the keys to break ties:

```
int suitor_compare(suitor *a, suitor *b)
{
        int result;  /* result of comparison */

        if (a->height < b->height) return(-1);
        if (a->height > b->height) return(1);

        if (a->weight < b->weight) return(-1);
        if (a->weight > b->weight) return(1);

        if ((result=strcmp(a->last,b->last)) != 0)
                    return result;

        return(strcmp(a->first,b->first));
}
```

Stable Sorting

Another way is to use repeated passes through a *stable* sorting function, one which is guaranteed to leave identical keys in the same relative order they were in before the sorting.

If so, we could sort by first name, and then do stable sort by last name and know the final results are sorted by both major and minor keys.

STL provides a stable sorting routine, where keys of equal value are guaranteed to remain in the same relative order.

```
void stable_sort(RandomAccessIterator bg, RandomAccessIterator end)
void stable_sort(RandomAccessIterator bg, RandomAccessIterator end,
          BinaryPredicate op)
```

Sorting Library Functions in C

The `stdlib.h` contains library functions for sorting and searching. For sorting, there is the function `qsort`:

```
#include <stdlib.h>

void qsort(void *base, size_t nel, size_t width,
          int (*compare) (const void *, const void *));
```

It sorts the first `nel` elements of an array (pointed to by `base`), where each element is `width`-bytes long. Thus we can sort arrays of 1-byte characters, 4-byte integers, or 100-byte records, all by changing the value of `width`.

Binary search is also included:

```
bsearch(key, (char *) a, cnt, sizeof(int), intcompare);
```

Comparison Functions

The ultimate desired order is determined by the function `compare`. It takes as arguments pointers to two `width`-byte elements, and returns a negative number if the first belongs before the second in sorted order, a positive number if the second belongs before the first, or zero if they are the same.

Sorting and Searching in C++

The C++ Standard Template Library (STL) includes methods for sorting, searching, and more. Serious C++ users should get familiar with STL.

To sort with STL, we can either use the default comparison (e.g., $\leq$) function defined for the class, or override it with a special-purpose comparison function op:

```
void sort(RandomAccessIterator bg, RandomAccessIterator end)
void sort(RandomAccessIterator bg, RandomAccessIterator end,
          BinaryPredicate op)
```

STL Sorting Applications

Other STL functions implement some of the applications of sorting, including,

- `nth_element` – Return the nth largest item in the container.
- `set_union`, `set_intersection`, `set_difference` – Construct the union, intersection, and set difference of two containers.

  predicate
- `unique` – Remove all consecutive duplicates.

Sorting and Searching in Java

The `java.util.Arrays` class contains various methods for sorting and searching. In particular,

```
static void sort(Object[] a)
static void sort(Object[] a, Comparator c)
```

sorts the specified array of objects into ascending order using either the natural ordering of its elements or a specific comparator c. Stable sorts are also available.

Methods for searching a sorted array for a specified object using either the natural comparison function or a new comparator c are also provided:

```
binarySearch(Object[] a, Object key)
binarySearch(Object[] a, Object key, Comparator c)
```

110401 (Vito's Family)

Find the most central location on a street, to minimize total distance to certain neighbors.

Is the right version of average mean, median, or something else?

110403 (Bridge)

How should n people of variable speeds cross a two-man bridge at night when there is only one flashlight?
Does sorting the people by speed help determine who should be paired up?

110405 (Shoemaker's Problem)

How should a shoemaker order the jobs he does so as to minimize total penalty costs for lateness, when different jobs have different penalties?

Does it help to sort jobs by their length, or by their penalty, or both?

110406 (CDVII)

Compute the total amount of tolls people owe in a world where toll costs vary with time.
How do we pair up entry and exit records?