Sorting and Searching

CMPT 125
Mo Chen
SFU Computing Science
22/1/2020

Lecture 9

Today:

- Introduction to sorting
- Selection sort

Sorting

Goal: Place a collection of items in order from smallest to largest.

- Order depends on the type of the items
- For numbers, it's by value

`a <= b`

8	10	5	-5

$\longrightarrow$

-5	5	8	10

- For strings, it's alphabetical order

"Deb"	"Alice"	"Charlie"	"Bob"

$\longrightarrow$

"Alice"	"Bob"	"Charlie"	"Deb"

`strcmp(a, b) <= 0`

Why is Sorting Useful?

- A core algorithm in computer science, studied in depth for many many years.
  - sometimes important in its own right.
  - sometimes applied as a step in a larger algorithm.
- Our purpose isn’t to learn to implement a sort
  - (but you probably will write one or two sorts in Lab)
  - many excellent implementations are out there
  - every programming language has a library sort
- Our purpose is to understand the techniques that are used and the analysis to evaluate them

Simple Sorting

- As an example of algorithm analysis let’s look at two simple sorting algorithms
  - Selection Sort
  - Insertion Sort
- Predict the running time for each sorting algorithm by counting the operations
  - most expensive (i.e., frequent) operations are the comparison and movement of objects
  - compare sorting algorithms using Big-*O*

Selection Sort

Main idea: Repeatedly find the smallest item, and move it into position using a swap

- Start by finding the smallest element. Say its position is at index `minpos`
- Exchange `A[0]` $\leftrightarrow$ `A[minpos]`
- Now think of the array as in two parts: a sorted part (`A[0..0]`) and an unsorted part (`A[1..n-1]`).
- Find the smallest element of the unsorted part
- Exchange it with `A[1]`
- The sorted part is now `A[0..1]`, unsorted part is now `A[2..n-1]`.
- Find the next min, . . .

Selection Sort Demo

Sort this array using Selection Sort:

How many comparisons for an array of length N?

82	50	$\boxed{12}$	68	17	95	35	23	find smallest unsorted item in	7 comparisons
12	50	82	68	$\boxed{17}$	95	35	23	find smallest unsorted item in	6 comparisons
12	17	82	68	50	95	35	$\boxed{23}$	find smallest unsorted item in	5 comparisons
12	17	23	68	50	95	$\boxed{35}$	82	find smallest unsorted item in	4 comparisons
12	17	23	35	$\boxed{50}$	95	68	82	find smallest unsorted item in	3 comparisons
12	17	23	35	50	95	$\boxed{68}$	82	find smallest unsorted item in	2 comparisons
12	17	23	35	50	68	95	$\boxed{82}$	find smallest unsorted item in	1 comparison
12	17	23	35	50	68	82	95	find smallest unsorted item???	

Selection Sort in C

```
void SelectionSort(int arr[], int len) {
```

- Repeat for all `i` from `0` to `len-2`:
  - Determine `minpos`, the index of the smallest element of `arr[i..len-1]`
  - Algorithm: linear scan
  - Swap min element into position
  - `arr[minpos]` ↔ `arr[i]`

```
}
```

Selection Sort in C

```
void SelectionSort(int arr[], int len) {
```

- Repeat for all `i` from `0` to `len-2`:

```
minpos = i;
for (int j = i+1; j < len; j++) {
        if (arr[j] < arr[minpos]) {
                minpos = j;
    }
}
```

- Swap min element into position
- `arr[minpos] ↔ arr[i]`

```
}
```

Selection Sort in C

```
void SelectionSort(int arr[], int len) {
```

- Repeat for all `i` from `0` to `len-2`:

```
minpos = i;
for (int j = i+1; j < len; j++) {
    if (arr[j] < arr[minpos]) {
        minpos = j;
    }
}
int tmp = arr[i];
arr[i] = arr[minpos];
arr[minpos] = tmp;
```

```
}
```

Selection Sort in C

For now, in pseudocode.

(Needs to be a valid logical expression)

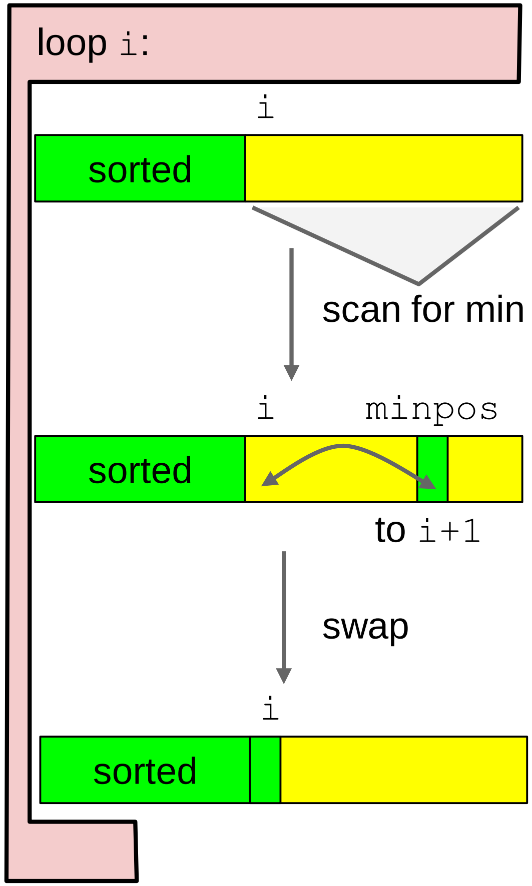
```
void SelectionSort(int arr[], int len) {
    for (int i = 0; i < len-1; i++) {
        assert(arr[0..i-1] sorted, with i smallest items)
        minpos = i;
        for (int j = i+1; j < len; j++) {
            if (arr[j] < arr[minpos]) {
                minpos = j;
            }
        }
        int tmp = arr[i];
        arr[i] = arr[minpos];
        arr[minpos] = tmp;
    }
}
```

A Note About Assertions

Assertions: part comment / part math

Two benefits:

1. Can reason about the algorithm
    - Can visualize the progress of an algorithm
        - E.g., first `i` elements of `arr[]` always hold the smallest `i` elements of `arr[]` in sorted order
    - If you can prove the assertion holds throughout the algorithm, then you prove the algorithm is correct
        - called a *loop invariant*



A Note About Assertions

Two benefits (continued)

2. Stronger debugging
   - C’s `assert(condition);` will check your assertions, and halt your program if an assertion fails.
   - a failed assertion means your program has a bug.
   - parameter to `assert( … )` should produce no side-effects!

Q. What would be a good C version of the assertion used in Selection Sort?

Analysis of Selection Sort

How many steps for an input of length N?

- Comparisons:
  - N - 1 to find first min,
  - then N - 2 for the second min,
  - then N - 3,
  - $\ldots$,
  - then 3,
  - then 2,
  - then 1.
- Swaps:
  - N - 1 swaps
- Total running time: $O(N^2)$

Total Comparisons

$$= \sum_{i=1}^{N-1} i$$

$$= \frac{N(N-1)}{2}$$

Independent of the nature of the input (the initial ordering of the numbers)