Lecture 15: Elementary Sorting

Administrativia

Review your grades!
Autolab and Gradescope
What’s next?
1 programming assignment, 2 problem sets, 1 final exam

Elementary Sorting

Algorithms

Total Order

Every pair of items must be comparable according to a **total order**, that satisfies:

- **Antisymmetry**: if \( k_1 \leq k_2 \) and \( k_2 \leq k_1 \) then \( k_1 = k_2 \)
- **Transitivity**: if \( k_1 \leq k_2 \) and \( k_2 \leq k_3 \) then \( k_1 \leq k_3 \)
- **Totality**: \( k_1 \leq k_2 \) or \( k_2 \leq k_1 \)

Any comparison rule that satisfies total order will never lead to a contradiction

Selection Sort

Array is divided into **sorted** (left) and **unsorted** (right) parts, and several scans are run from left to right sorted part grows over time

At each iteration \( i \), find index \( \text{min} \) of the smallest element in the unsorted part

swap \( A[i] \) and \( A[\text{min}] \)

```c
void selection(ul_int *A, ul_int n) {
    ul_int i, j, min, temp;
    // grows the left part (sorted)
    for (i = 0 ; i < (n-1) ; i ++) {
        min = i;
        // find min in unsorted part
        for (j = i+1 ; j < n ; j ++) {
            if (A[j] < A[min]) {
                min = j;
            }
        }
        // swap A[i] and A[min]
        swap(A, i, min);
    }
}
```

Running time insensitive to input quadratic time in all cases

Minimal data movement

Selection Sort

Insertion Sort

Array is divided into **sorted** (left) and **unsorted** (right) parts, and several scans are run from left to right sorted part grows over time

At each iteration \( i \), swap \( A[i] \) with each larger entry to its left

inserts \( A[i] \) (from unsorted) into the sorted part
### Insertion Sort

**Worst case**
- array reversed (no duplicates)

**Best case**
- array already sorted

### Partially Sorted Arrays

“array is **partially sorted** if the number of pairs that are out-of-order is $O(n)$”

For partially sorted arrays Insertion Sort runs in **Linear Time**

### Sorting based on Comparisons

**Basic operation:** compare two items

**What is a lower bound for the cost of sorting?**

### Decision Tree (sorting x, y, z)

**Number of leaves?**
- number of all possible permutations

**Height of the decision tree?**
- worst-case number of comparisons

### Decision Tree (sorting x, y, z)

### What is the height?

Consider sorting $n$ distinct items

- number of leaves at least $n!$ (# permutations)
- number of leaves at most $2^h$ (perfect binary tree)

$2^h \geq n!$  

$h \geq \log n!$  

$h \geq n \log n$  

by Stirling’s formula
<table>
<thead>
<tr>
<th>Cost of Sorting</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is a <strong>lower bound</strong> for the cost of sorting algorithms?</td>
</tr>
<tr>
<td>$\Omega(n \log n)$</td>
</tr>
<tr>
<td>Cost of sorting algorithms considered <strong>optimal</strong>:</td>
</tr>
<tr>
<td>$\Theta(n \log n)$</td>
</tr>
</tbody>
</table>