CSCI 262
Data Structures

3

Selection Sort

Introduction to Analysis of Algorithms

CS@Mines

Getting things in order

SORTING

CS@Mines

Sorting

- Input: a list of elements, e.g. integers
- Output: a list of the input elements in sorted order

Why do we study this problem?

- Teaching example
  - Algorithm design
  - Algorithm analysis
- Sorting is also useful for all sorts of applications!

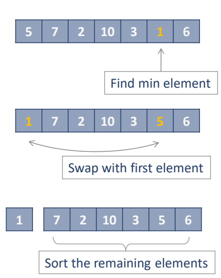
CS@Mines 3

Selection Sort

- Input: a list of elements, e.g. integers
- Output: a list of the input elements in sorted order
- A simple solution:
  - Find the minimum element in the list
  - Swap it with the first element in the list
  - Sort the sublist following the first element
- This sorting algorithm is named **selection sort**.

CS@Mines 4

Selection Sort Illustrated



CS@Mines 5

Selection Sort Code

```
// for vectors of int
void selection_sort(vector<int> &vec) {
    int n = vec.size();
    for (int left = 0; left < n; left++) {
        int right = left;
        for (int j = left + 1; j < n; j++) {
            if (vec[j] < vec[right]) right = j;
        }
        swap(vec[left], vec[right]);
    }
}
```

CS@Mines 6

Swap Code

```
// exchange two int values
void swap(int &a, int &b) {
    int tmp = a;
    a = b;
    b = tmp;
}
```

This function is actually already available in the standard library; #include <algorithm>

CS@Mines

How much?

MEASURING WORK

CS@Mines

How Much Work in Selection Sort?

- Difficult to actually count CPU cycles –
  - Differs by CPU
  - Differs by compiler
  - Lots of noise factors: caching, context switching, etc.
- Simplification: just measure comparisons and swaps
  - Ignore loop counter updates, etc.
  - We'll see later why we can get away with this
- Let's count (only somewhat carefully):
  - Use a vector of size 10
  - Later, generalize to size *n*

CS@Mines

Analyzing Selection Sort

```
void selection_sort(vector<int> &vec) {
    int n = vec.size();
    for (int left = 0; left < n; left++) {
        int right = left;
        for (int j = left + 1; j < n; j++) {
            if (vec[j] < vec[right]) right = j;
        }
        swap(vec[left], vec[right]);
    }
}
```

CS@Mines 10

Analyzing Selection Sort: 1st Loop

On first loop:

- Compare min element with each of 9 elements: cost = 9
- Do 1 swap: cost = 1

Total cost: 10

CS@Mines

Analyzing Selection Sort: 2nd Loop

On second loop:

- Compare min element with each of 8 elements: cost = 8
- Do 1 swap: cost = 1

Total cost: 9

CS@Mines

Analyzing Selection Sort: Last Loop

In the end, we have a list of size 1 left and don't have to do any work!

So work is 10 + 9 + 8 + … + 1 + 0

= 55

n

Now let's generalize to vectors of size *n*

First loop: n – 1 comparisons, 1 swap: cost = n

Second loop: cost = n – 1

…

Cost: n + (n – 1) + (n – 2) + … + 1 + 0

This is a famous sum!

Arithmetic Series

Memorize this!

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

That is,

$$0 + 1 + 2 + \cdots + n = \frac{n^2 + n}{2}.$$

How to Solve $\sum_{i=0}^{n} i$

Write the sum twice, once forwards and once backwards; then sum the two:

	0	+	1	+	…	+	n-1	+	n
+	n	+	n-1	+	…	+	1	+	0
=	n	+	n	+	…	+	n	+	n

How many n's are there in the sum? Answer: n+1.

Since we took twice the summation, we have to divide by 2,
Thus we have
n(n+1)/2.

Can also prove easily using induction, geometry, …

Visual Analysis

Counting based on code can be a pain;

Sometimes, a visual approach is simpler:

Original list	5	7	2	10	3	1	6	
Elements "touched" in first loop iteration	5	7	2	10	3	1	6	n
Elements "touched" in second loop iteration	1	7	2	10	3	5	6	n - 1
Elements "touched" in third loop iteration	1	2	7	10	3	5	6	n - 2
…								
Last iteration	1	2	3	5	6	7	10	1

Cost: O(1 + 2 + … + n)

ALGORITHMIC ANALYSIS

"Big O"

Big O notation:

O(n) measures *asymptotic complexity* of algorithm

Don't worry about the fancy language for now – this will be explained in CSCI 406!

What is important:

- In Big O, lower order terms and constants don't matter
- Only interested in how functions *grow* with size of n

CS@Mines

Simplifying

Typically use the simplest term in expression:

- E.g., lower order polynomials can be ignored because they are completely *dominated* by higher order polynomials
  - $O(n)$ not $O(n + c)$
  - $O(n^2)$ not $O(n^2 + n + c)$
- Ignore constants
  - $O(n)$ not $O(3n)$
  - $O(n)$ not $O(n/2)$

Dominance relations (here a > b means a dominates b):

$n! > 3^n > 2^n > n^3 > n^2 > n \log n > n > \log n > 1$

CS@Mines

Practice

Simplify the following:

$O(n^3 + 4)$

$O(12n^2 – n + 1)$

$O(2^n + n^2)$

$O(n + n^2 + n \log n)$

CS@Mines

Technicalities (for the curious)

- Defined:

  $f(n) = O(g(n))$ means $f(n) \le c\, g(n)$ for some c as $n \to \infty$

  More formally, $f(n) = O(g(n))$ if there exists c, n_0 such that $f(n) \le c\, g(n)$ for all $n > n_0$.

- The *asymptotic complexity* of *f* is upper bounded by *g*

CS@Mines

Proof Sketch (for the curious)

(Just for show: not on any exams or homework!)

Prove: $3n^2 + n/4 + 1 = O(n^2)$

Find c, n_0 such that $(3n^2 + n/4 + 1) \le c\, n^2$ for all $n \ge n_0$

Choose $c = 4$, $n_0 = 2$

Inductive proof:

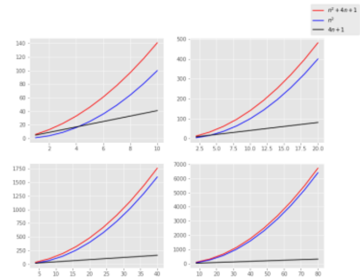
- Base case: $3(2^2) + 2/4 + 1 = 13½ \le 16 = 4(2^2)$
- Induction:
  - Suppose $3n^2 + n/4 + 1 \le 4 n^2$
  - Show for n + 1:

$$3 (n + 1)^2 + (n + 1)/4 + 1 \le 4 (n + 1)^2$$
$$3 (n^2 + 2n + 1) + n/4 + 1/4 + 1 \le 4 (n^2 + 2n + 1)$$
$$3n^2 + 6n + 3 + n/4 + 1/4 + 1 \le 4n^2 + 8n + 4$$
$$(3n^2 + n/4 + 1) + 6n + 3¼ \le (4n^2) + 8n + 4$$

□

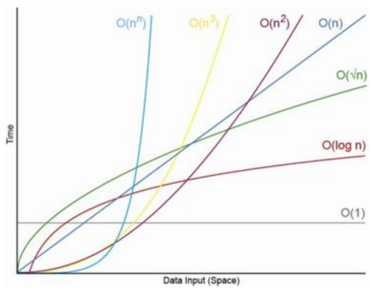
CS@Mines

Asymptotic Complexity Comparison



CS@Mines

Big-O Comparisons



CS@Mines

Why We Care 1

Comparison of different orders of functions as size of input n:

n	10	100	1000	10^6	10^9
log(n)	1	2	3	6	9
n	10	100	1000	1000000	1000000000
n log(n)	10	200	3000	6×10^6	9×10^9
n^2	100	10^4	10^6	10^{12}	10^{18}
2^n	1024	$\sim 10^{30}$	$\sim 10^{300}$	Forget it!	

CS@Mines

Why We Care 2

Assuming 2×10^{10} operations/second
(approximately the FP performance of a typical CPU c. 2011)

n	10	50	100	10^6	10^9	10^{12}
log(n)	< 1 ns	< 1 ns	< 1 ns	1 ns	1 ns	2 ns
n	< 1 ns	< 1 ns	< 1 ns	50 µs	50 ms	50 s
n log(n)	< 1 ns	< 1 ns	1 ns	300 ms	450 ms	10 min
n^2	<1 ns	125 ns	500 ns	50 s	1.6 years	1.6 million years
2^n	50 ns	16 hours	1.5 trillion years			

↑ Datasets of size 10^6 and above are commonplace!

↑ # of unique URLs seen by Google indexer c. 2010

CS@Mines

Up Next

- Friday, August 31
  - Lab 2
  - APT 1 due
  - Project 1 assigned

CS@Mines