Sorting Algorithms

Sorting

- Sort data in order
  - Numbers in ascending/descending order
  - Strings alphabetically
  - Dates chronologically
  - etc
- Total order
  - Ascending $x_0 \leq x_1 \leq x_2 \leq x_3 \leq .... \leq x_{n-1}$
  - Descending $x_0 \geq x_1 \geq x_2 \geq x_3 \geq .... \geq x_{n-1}$

Total order

$$x_0 \leq x_1 \leq x_2 \leq x_3 \leq \ldots \leq x_{n-1}$$

- Is a binary relation ≤ that satisfies
  - Antisymmetry: if both $v \leq w$ and $w \leq v$, then $v = w$.
  - Transitivity: if both $v \leq w$ and $w \leq x$, then $v \leq x$.
  - Totality: either $v \leq w$ or $w \leq v$ or both.

Useful sorting abstractions

Less. Is item v less than w ?

```
private static boolean less(Comparable v, Comparable w)
{  return v.compareTo(w) < 0;  }
```

Exchange. Swap item in array a[] at index i with the one at index j.

```
private static void exch(Comparable[] a, int i, int j)
{
   Comparable swap = a[i];
   a[i] = a[j];
   a[j] = swap;
}
```

- public interface Comparable <T> -This interface imposes a total ordering on the objects of each class that implements it.
- public int compareTo (Item x)
- Implemented by String, Integer, Double, Short, Calendar, Year, etc

Performance Analysis

› Cost models
  – Running time
  – Memory cost

› Methods to measure/express
  – Tilde notation, T(n) – counting number of executions of certain operations as a function of input size n

› Order of growth classification
  – Big Theta Θ(n) – asymptotic order of growth
  – Big Oh O(n) - upper bound
  – Big Omega Ω (n) – lower bound

Performance Analysis

- Time complexity
  - Worst Case Analysis – usually done
    - Upper bound on running time of an algorithm
    - Must know the case that causes the maximum number of operations to be performed, eg in linear search, if the element is not in the array
  - Average – not easy to do in practice
    - Take all possible inputs and calculate computing time for all of the inputs, and average
    - Must know/predict distribution of cases
  - Best – is it any use if worst case bad?
    - Lower bound on running time of an algorithm
    - Must know the case that causes the minimum number of operations to be performed

Refresher

- Refer to semester 1 slides Lecture 4 for more details on performance analysis

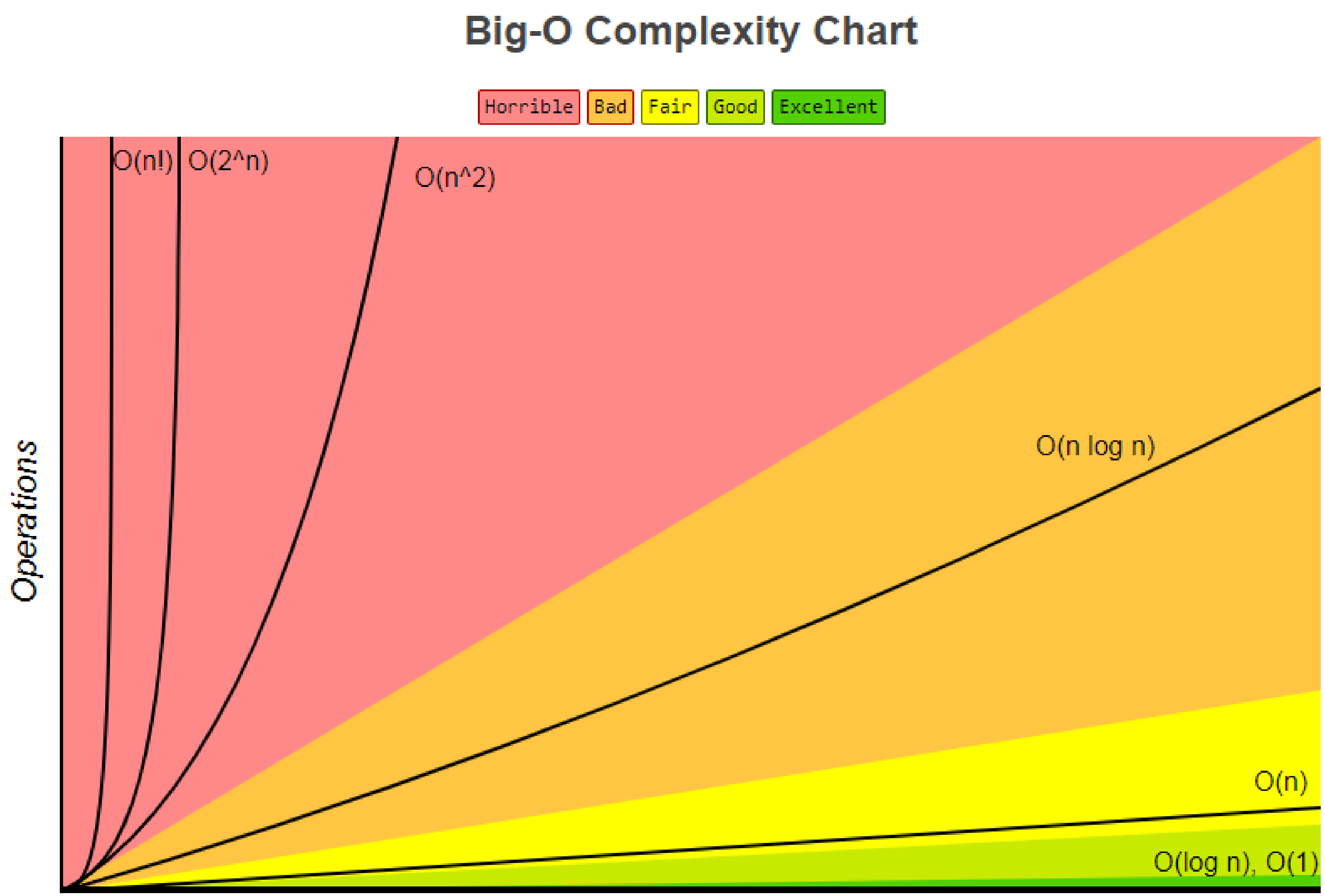
Common order-of-growth classifications

order of growth	name	typical code framework	description	example	$T(2N)$ / $T(N)$
1	**constant**	`a = b + c;`	statement	add two numbers	1
$\log N$	**logarithmic**	`while (N > 1)` `{ N = N / 2; ... }`	divide in half	binary search	~ 1
N	**linear**	`for (int i = 0; i < N; i++)` `{ ... }`	loop	find the maximum	2
$N \log N$	**linearithmic**	[see mergesort lecture]	divide and conquer	mergesort	~ 2

N^2	**quadratic**	`for (int i = 0; i < N; i++)`
		`for (int j = 0; j < N; j++)`
		`{ ... }`
			double loop	check all pairs	4

N^3	**cubic**	`for (int i = 0; i < N; i++)`
		`for (int j = 0; j < N; j++)`
		`for (int k = 0; k < N; k++)`
		`{ ... }`
			triple loop	check all triples	8

2^N	**exponential**	[see combinatorial search lecture]	exhaustive search	check all subsets	$T(N)$



http://bigocheatsheet.com/

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

http://bigocheatsheet.com/

Running time estimates:

- Laptop executes 10^8 compares/second.
- Supercomputer executes 10^{12} compares/second.

	insertion sort (N^2)			mergesort ($N \log N$)		
computer	thousand	million	billion	thousand	million	billion
home	instant	2.8 hours	317 years	instant	1 second	18 min
super	instant	1 second	1 week	instant	instant	instant

Visualisation of sorting algorithm performance

- https://www.youtube.com/watch?v=ZZuD6iUe3Pc
- Tested for different types of input

Why do we need so many then?

- No Free Lunch Theorem
- Different applications/different behaviour based on input
- Examples
  - Merge sort – useful for linked lists
  - Heapsort – sorting arrays, predictable, very little extra RAM
  - Quicksort – excellent average-case behaviour
  - Insertion sort – good if your list is already almost sorted
  - Bubble sort – if small enough data set, it is the simplest to implement

- Also, a handy way to learn different algorithm design strategies on the same example!

Stability of Sorting Algorithms

› Stable if two objects with equal keys appear in the same order in sorted output as they appear in the input array to be sorted

› Do we care?

– NO: When equal elements are indistinguishable, such as with integers, or more generally, any data where the entire element is the key

– NO: If all keys are different.

– YES: if duplicate keys and want to maintain original order by eg secondary keyWhen equal elements are indistinguishable, such as with integers, or more generally, any data where the entire element is the key, stability is not an issue. Stability is also not an issue if all keys are different.

Stability of Sorting Algorithms

sorted by time	sorted by location (not stable)	sorted by location (stable)
Chicago 09:00:00	Chicago 09:25:52	Chicago 09:00:00
Phoenix 09:00:03	Chicago 09:03:13	Chicago 09:00:59
Houston 09:00:13	Chicago 09:21:05	Chicago 09:03:13
Chicago 09:00:59	Chicago 09:19:46	Chicago 09:19:32
Houston 09:01:10	Chicago 09:19:32	Chicago 09:19:46
Chicago 09:03:13	Chicago 09:00:00	Chicago 09:21:05
Seattle 09:10:11	Chicago 09:35:21	Chicago 09:25:52
Seattle 09:10:25	Chicago 09:00:59	Chicago 09:35:21
Phoenix 09:14:25	Houston 09:01:10	Houston 09:00:13
Chicago 09:19:32	Houston 09:00:13	Houston 09:01:10
Chicago 09:19:46	Phoenix 09:37:44	Phoenix 09:00:03
Chicago 09:21:05	Phoenix 09:00:03	Phoenix 09:14:25
Seattle 09:22:43	Phoenix 09:14:25	Phoenix 09:37:44
Seattle 09:22:54	Seattle 09:10:25	Seattle 09:10:11
Chicago 09:25:52	Seattle 09:36:14	Seattle 09:10:25
Chicago 09:35:21	Seattle 09:22:43	Seattle 09:22:43
Seattle 09:36:14	Seattle 09:10:11	Seattle 09:22:54
Phoenix 09:37:44	Seattle 09:22:54	Seattle 09:36:14

no longer sorted by time

still sorted by time

Stability when sorting on a second key

› Stable sorting algorithms: Insertion sort, bubble sort, merge sort

Memory requirements/In-place algorithms

- Transforms input without additional auxiliary data structure, eg array
- A small amount of extra storage space is allowed for auxiliary variables
- The input is usually overwritten by the output as the algorithm executes
- In-place algorithm updates input sequence only through replacement or swapping of elements

- Affects space complexity of an algorithm
- Selection, insertion, shell, quick

Checkpoint – is the material clear enough?

Which of these O(n) has the highest complexity (worst worst performance)?

- A – $O(\log n)$
- B – $O(n \log(n))$
- C – $O(n^2)$
- D – $O(2^n)$

https://responseware.turningtechnologies.eu/responseware/polling or use Turning Point App

Session id: