Big-O classes

- Examples:
  - O(1) constant: cost is independent of n : Fixed cost!
    - Retrieve/insert in regular arrays, hashmap operations
  - O(log n) logarithmic: cost grows by 1, when n doubles : *almost constant*
    - Traversing a binary tree, some divide-conquer algorithms
  - O(n) linear: cost grows *linearly* with n :
    - Find a value in array, do something to all elements in an array, adding in the middle of ArrayList
  - O(n log n) log linear: cost grows a bit more than linear: *Slow growth!*
    - Good sorting algorithms (merge, quick, heap sort). Complex divide-conquer algorithms

Big-O classes

- Examples continued:
  - $O(n^2)$ quadratic: costs x 4 when n doubles: *limits problem size*
    - Do something to all elements in a 2d array. Nested loops
  - $O(n^c)$, c>2 polynomial: *limits problem size even more*
    - Do something to all elements in a 3d array. Many nested loops
  - $O(2^n)$ exponential: costs doubles when n increases by 1: *severely limits problem size*
    - Route finding, e.g. travelling salesman problem
  - Super-exponential: e.g.O(n!) *don't even think about it…*

Manageable Problem Sizes

- How big can the data be if
  - one step takes around a microsecond
  - algorithm is O(….)

	1 min	1 h	1 day	1 week	1 year
O(n)	10^7	10^9	10^{11}	10^{12}	10^{13}
O(n log n)	10^6	10^8	10^9	10^{10}	10^{12}
$O(n^2)$	10^4	10^5	10^5	10^6	10^7
$O(n^3)$	10^2	10^3	10^3	10^4	10^4
$O(2^n)$	25	31	36	39	44

half million seconds in a year

What is a “step”?

- Any important actions that are at the center of the algorithm
  - comparing data
  - moving data
  - anything you consider to be “expensive”
  - Doesn’t depend on size of data

```
public E remove(int index){
    if (index < 0 || index >= count) throw new ....Exception();
    E ans = data[index];
    for (int i=index+1; i< count; i++)
        data[i-1]=data[i];                    ← Key Step
    count--;
    data[count] = null;
    return ans;
  }
```

What's a step: Pragmatics

- Count the most expensive actions?
  - Adding 2 numbers is cheap
  - Raising to a power is not so cheap
  - Comparing 2 strings *may* be expensive
  - Reading a line from a file *may* be very expensive
  - Waiting for input from a user or another program may take forever…
- Remember the Big (O) picture
  - Multiplication is more costly than addition
  - BUT if addition is done in an $O(n^2)$ algorithm and multiplication in an $O(n \log(n))$ algorithm, then for even a relatively small number *n*, the extra cost of multiplication is negligible.
- Sometimes we need to know about how the underlying operations are implemented in the computer to choose well (NWEN241/342).

Costs of Standard Collection classes

- ArrayList: O(1): clear, add, set, remove from end:
  O(n): add, remove, contains, Collections.reverse, .shuffle
  O(n log(n)) Collections.sort,

- ArrayDeque: O(1): clear, push, pop, offer, poll, add/remove First/Last:
  O(n): contains, remove(obj)
- PriorityQueue: O(log(n)): offer, poll

- HashSet: O(1): add, remove, contains
- TreeSet: O(log(n)): add, remove, contains

- HashMap: O(1): clear, containsKey, put, get
  But depends on the cost of hashCode