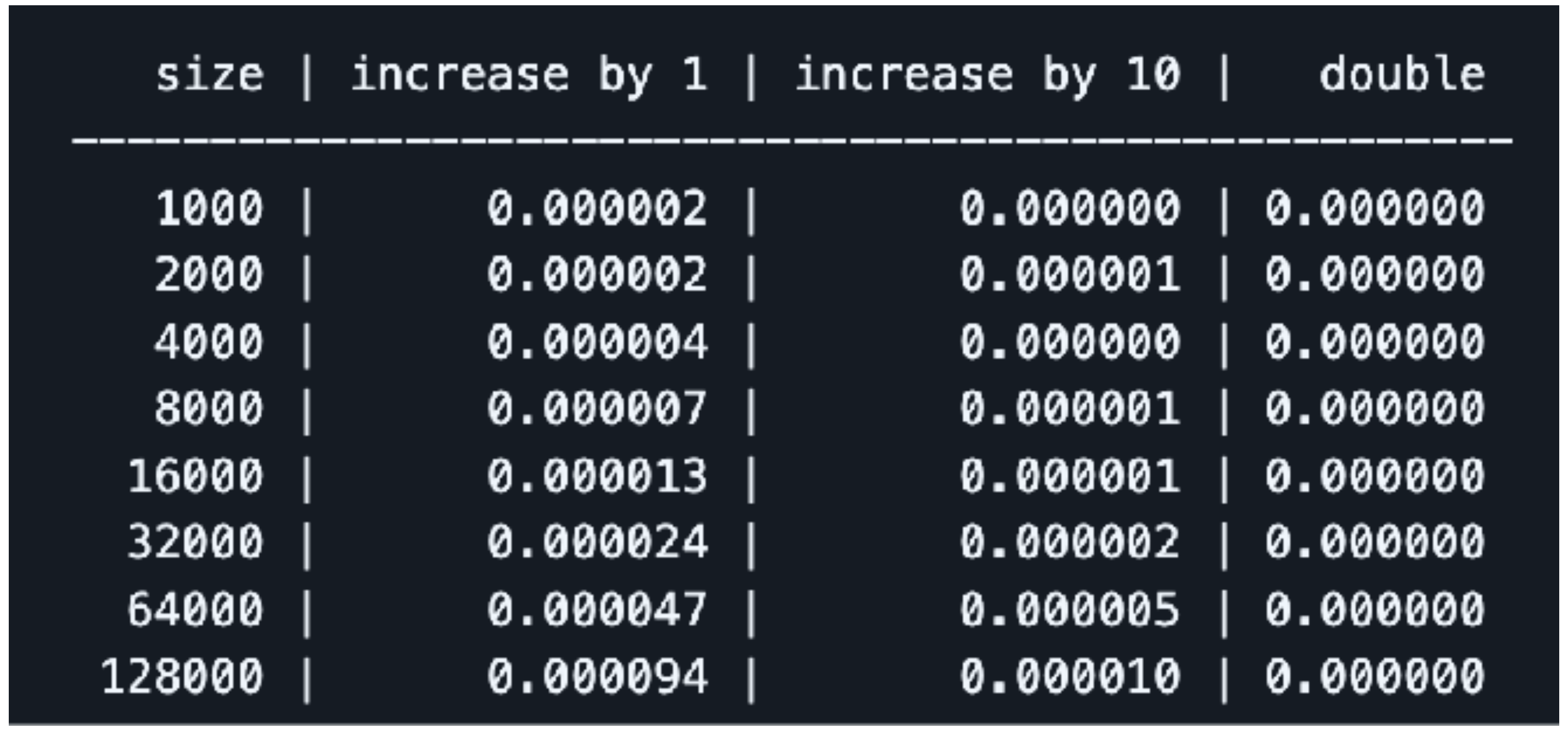
CS62 Lab3: Timing ArrayLists

Basic Data Structures



```
    size | increase by 1 | increase by 10 |   double
-------------------------------------------------------
    1000 |      0.000002 |       0.000000 | 0.000000
    2000 |      0.000002 |       0.000001 | 0.000000
    4000 |      0.000004 |       0.000000 | 0.000000
    8000 |      0.000007 |       0.000001 | 0.000000
   16000 |      0.000013 |       0.000001 | 0.000000
   32000 |      0.000024 |       0.000002 | 0.000000
   64000 |      0.000047 |       0.000005 | 0.000000
  128000 |      0.000094 |       0.000010 | 0.000000
```

Lab 2 agenda

- Quiz
- Lab

Standard Operations of `ArrayList<E>` class

- `ArrayList()`: Constructs an empty ArrayList with an initial capacity of 2 (can vary across implementations, another common initial capacity is 10).
- `ArrayList(int capacity)`: Constructs an empty ArrayList with the specified initial capacity.
- `isEmpty()`: Returns true if the ArrayList contains no elements.
- `size()`: Returns the number of elements in the ArrayList.
- `get(int index)`: Returns the element at the specified index.
- `add(E element)`: Appends the element to the end of the ArrayList.
- `add(int index, E element)`: Inserts the element at the specified index and shifts the element currently at that position (if any) and any subsequent elements to the right (adds one to their indices).
- `E remove()`: Removes and returns the element at the end of the ArrayList.
- `E remove(int index)`: Removes and returns the element at the specified index. Shifts any subsequent elements to the left (subtracts one from their indices).
- `E set(int index, E element)`: Replaces the element at the specified index with the specified element and returns the olde element.
- `clear()`: Removes all elements.

Lab structure

- We’ll answer the question, why do we double the size of the ArrayList? Why not just increase it by like 10 each time? Why is doubling better?
- You’ll need to create your own .java file for this one
- What we’re grading on Gradescope is answers.txt

Doubling analysis (we saw this in lecture)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Insertion Cost	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Copying Cost	0	1	2	0	4	0	0	0	8	0	0	0	0	0	0	0	16
Total Cost	1	2	3	1	5	1	1	1	9	1	1	1	1	1	1	1	17

- As the ArrayList increases, doubling happens *half as often* but *costs twice as much.*
- O(total cost)= $\sum$(“cost of insertions”) + $\sum$(“cost of copying”)
- $\sum$(“cost of insertions”) $= n$.
- $\sum$(“cost of copying”) = $1 + 2 + 2^2 + \ldots + 2^{\log_2 n - 1} \leq 2n$.
- O(total cost) $\leq 3n$, therefore amortized cost is $\leq \dfrac{3n}{n} = 3 = O^+(1)$, but “lumpy”.

Amortized analysis for n add() operations when increasing ArrayList by 1.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Insertion Cost	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Copying Cost	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Total Cost	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

- $\sum$(“cost of insertions”) $= n.$
- $\sum$(“cost of copying”) = $0 + 1 + 2 + 3 + \ldots + n - 1 = n(n-1)/2.$
- O(total cost) $= n + n(n-1)/2 = n(n+1)/2$, therefore amortized cost is $(n+1)/2$ or $O^{+}(n)$.
- Same idea when increasing ArrayList size by a constant (like 10).
  - This is why increasing the capacity by 1 is the slowest, and 10 the second slowest, and doubling the fastest in this lab.