# CPSC 131 – Data Structures

## Analysis of Algorithms

*Professor T. L. Bettens*

*Fall 2020*

# Key terms

- **Experimental analysis**
- **Asymptotic analysis**
- **Worst-case analysis**
- **Big-Oh notation**

- **Constant time operations O(1)**
- **Logarithmic time operations O(log n)**
- **Linear time operations O(n)**
- **Quadratic time operations $O(n^2)$**

# Comparing data structures

- Is a doubly linked list better than a singly linked list?

- What does it men to be better? Under what conditions? What does "good" mean?
  - Running time? Memory used?
  - Usually means a trade-off

- Two approaches to answering this question:
  - Option 1: Experimental analysis
  - Option 2: Asymptotic analysis

# Option 1:  Experimental Analysis

- Implement the two data structures
- Implement a main function that loads both with same data
- Call the same data structure operations on both
  - Insert elements
  - Delete elements …
- Measure time spent by each data structure

# Option 1: Experimental Analysis

- Problems
  - Can do experiments only a limited data set
  - Other factors impact running time:
    - What other programs are running on the computer
  - Does the running time depend on the computer itself?
    - Do the results hold on a different computer?
  - Does the running time depend on the implementation?
    - Do the results hold on a different programming language?

# Option 2: Asymptotic Analysis

- Analysis without actually running any code
- Asymptotic: approaching a value closely
- Key idea:
  - We are interested in running time for large data sets
  - How fast will running time increase as we increase data size?
  - Rate of increase

# Option 2: Asymptotic Analysis

- Most important factor?
  - Number of elements in the data structure
- Represent this by ***n***
- Analysis
  - How does running time increase in terms of n?

# Printing a linked list

- Consider a singly linked list with n elements

```
SinglyLinkedList<int> list;
... // insert n integers
for (const auto & value : list )
{
  cout << value << endl;
}
```

- How many steps did we have to do?
  - Too complex!
- Do the number of steps increase in proportion to **n**?

# Printing a linked list

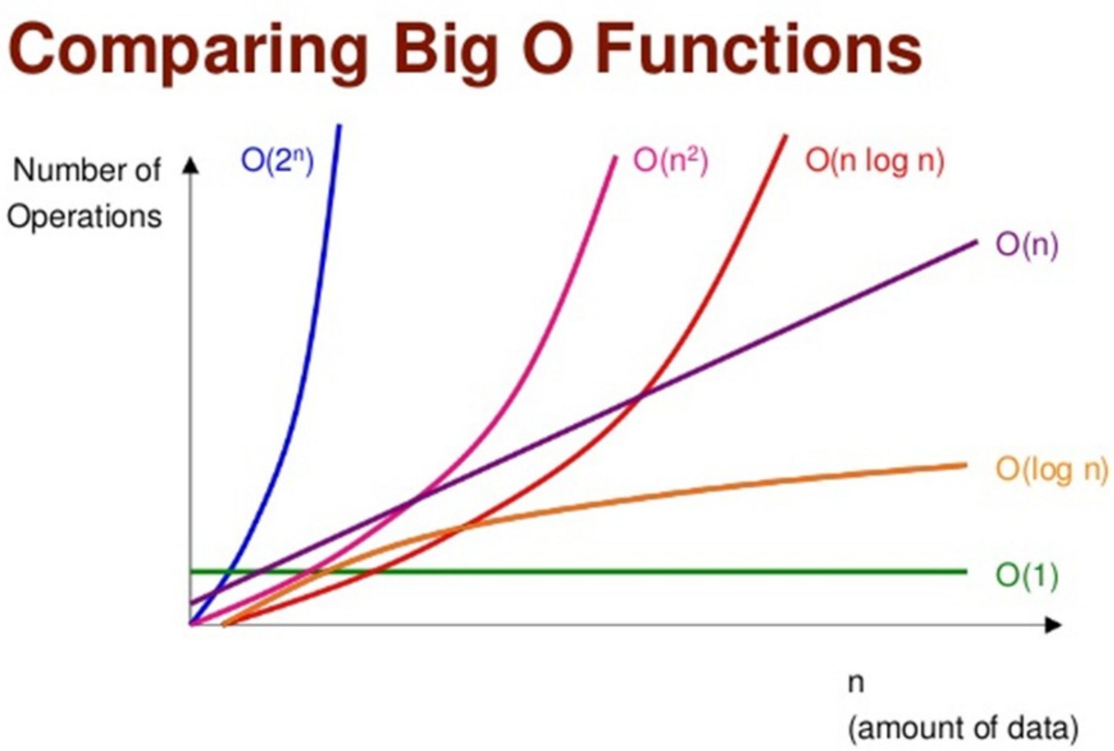
- Yes! number of operations increase in proportion to n
- “Printing all elements in a linked list takes order of n”
- Written as O(n)
- So also commonly spoken as “Oh of n”

# Printing a linked list

- But what about the fact that we had to
  - initialize the loop
  - Printing required cout
  - We also printed endl
  - …
- Don't care about constant factors
  - Focus on the big picture
    - Not details like initialization
- Worst-case analysis

# Common classes of Big-Oh functions

- O(1): Constant time operations
  - Does not depend on n
- O(n) : Linear time operations
  - Proportional to n
- O(log n) Logarithmic time operation
  - Bounded by $\log_2$ of n



# Big O Example: Binary Search

**Search for 75**

Round 1

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |
| lwr | | | | mid | | | | upr |

Round 2

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |
| | | | | | lwr | mid | | upr |

Round 3

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |
| | | | | | | | lwr<br>mid | upr |

Round 4

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |
| | | | | | | upr | lwr | |

**range is empty**

N = 9

75 == 50? (1)

75 == 70? (2)

75 == 80? (3)

(4)

- $\log_2(9) = 3.17 <= 4$
- Any n between 9 and 16 will take at most 4 operations
- Binary search of sorted data: O(log n)

# Comparing

**Data Structure**

- Arrays
- Singly linked lists
- Doubly linked lists
- Binary Trees
- Hash Tables

**Operations**

- Create empty
- Get front element
- Add or remove front element
- Get back element
- Add or remove back element
- Clear data structure
- Get/add/remove $i^{th}$ element

Source: ref2

# Comparing

- This course's goal: the design of "good" data structures and algorithms
- Data structures: systematic way of organizing and accessing data
- Algorithms: Step-by-step procedure for performing a task in a finite time.
- What does "good" mean?
  - Running time—fast
  - Space usage—small
- Usually means a trade-off
  - Faster often requires more memory for extra pointers
  - Smaller often requires more complex algorithms
- Essential point: **Running time increases with input size**

# Amortization

- Financial: Amortization is paying off an amount owed over time by making planned, incremental payments of principal and interest.

- Computer Science: To even out the costs of running an algorithm over many iterations, so that high-cost iterations are much less frequent than low-cost iterations, which lowers the average running time per iteration.

# Amortization Example:
# Increase the Extension Size, Reduce the Number of Copy Operations

| | Extend by 1 | | | Double Each Extension | | |
|---|---|---|---|---|---|---|
| **Insert #** | Size | Capacity | Copies | Size | Capacity | Copies |
| **1** | 1 | 1 | 0 | 1 | 1 | |
| **2** | 2 | 2 | 1 | 2 | 2 | 1 |
| **3** | 3 | 3 | 2 | 3 | 4 | 2 |
| **4** | 4 | 4 | 3 | 4 | 4 | |
| **5** | 5 | 5 | 4 | 5 | 8 | 4 |
| **6** | 6 | 6 | 5 | 6 | 8 | |
| **7** | 7 | 7 | 6 | 7 | 8 | |
| **8** | 8 | 8 | 7 | 8 | 8 | |
| **9** | 9 | 9 | 8 | 9 | 16 | 8 |
| **10** | 10 | 10 | 9 | 10 | 16 | |
| **11** | 11 | 11 | 10 | 11 | 16 | |
| **12** | 12 | 12 | 11 | 12 | 16 | |
| **13** | 13 | 13 | 12 | 13 | 16 | |
| **14** | 14 | 14 | 13 | 14 | 16 | |
| **15** | 15 | 15 | 14 | 15 | 16 | |
| **16** | 16 | 16 | 15 | 16 | 16 | |
| | | Total Copies | **120** | | Total Copies | **15** |

# What about memory requirement?

- So far, we have been speaking of running time
- What about how much memory/space is occupied by a data structure?
- Can also use O(n) concept:
  - Does memory usage go up proportionately to number of elements?

# Acknowledgements

Most of the material contained herein has been authored by:

- Dr. Anand Panangadan; California State University, Fullerton [ref1]
- Professor Allen Holliday; California State University, Fullerton [ref2]