IbraFSG™ 7 - Week 11; Recursive Sorting Algorithms

Ibrahim Chehab

UTM RGASC

March 27, 2024

Table of Contents

FSG HouseKeeping

1. **Reminder that a new FSG Session has been added:**
   - **When?** *Fridays, 12:00-1:00PM*
   - **Where?** *IB210*

FSG HouseKeeping

1. **Reminder that a new FSG Session has been added:**
   - **When?** *Fridays, 12:00-1:00PM*
   - **Where?** *IB210*
2. **Assignment 2 Deadline Coming Up!** Assignment 2 is due on April 3rd! Make sure you're actively working on it!

FSG HouseKeeping

1. **Reminder that a new FSG Session has been added:**
   - **When?** *Fridays, 12:00-1:00PM*
   - **Where?** *IB210*
2. **Assignment 2 Deadline Coming Up!** Assignment 2 is due on April 3rd! Make sure you're actively working on it!
3. **New Study Session Drop Ins!**
   Feeling Stressed about exams? Don't know where to start? Drop by the MN3260 for a study skills session!
   - **When:** March 27^{th} 3-4PM, and April 3^{rd} 1-2PM
   - **Where:** MN3260

FSG HouseKeeping

1. **Reminder that a new FSG Session has been added:**
   - **When?** *Fridays, 12:00-1:00PM*
   - **Where?** *IB210*
2. **Assignment 2 Deadline Coming Up!** Assignment 2 is due on April 3rd! Make sure you're actively working on it!
3. **New Study Session Drop Ins!**
   Feeling Stressed about exams? Don't know where to start? Drop by the MN3260 for a study skills session!
   - **When:** March 27^{th} 3-4PM, and April 3^{rd} 1-2PM
   - **Where:** MN3260
4. **Join the UTM CS Discord Server!** https://discord.gg/utmcs
   - **Note:** The server is not affiliated with the RGASC, CSC148H5, or the University of Toronto Mississauga
5. **Stay tuned for the CSC148 MEGA FSG ...**

Welcome back to IbraFSGs™

- Welcome back to IbraFSGs™! Hello to new people and welcome back to tenured members.
- This week we will be going over recursive sorting algorithms like *QuickSort* and *MergeSort*
- Recall in CSC108 we defind the following sorting algorithms:
  - Bubble Sort - Worst case $\mathcal{O}(n^2)$
  - Selection Sort - Worst case $\mathcal{O}(n^2)$
  - Insertion Sort - Worst case $\mathcal{O}(n^2)$

Welcome back to IbraFSGs™

- Welcome back to IbraFSGs™! Hello to new people and welcome back to tenured members.
- This week we will be going over recursive sorting algorithms like *QuickSort* and *MergeSort*
- Recall in CSC108 we defind the following sorting algorithms:
  - Bubble Sort - Worst case $\mathcal{O}(n^2)$
  - Selection Sort - Worst case $\mathcal{O}(n^2)$
  - Insertion Sort - Worst case $\mathcal{O}(n^2)$
- Today we will show how we can improve on these algorithms using recursion, achieving $\mathcal{O}(n \log(n))$ time complexity
- https://www.toptal.com/developers/sorting-algorithms

Welcome back to IbraFSGs™

- Welcome back to IbraFSGs™! Hello to new people and welcome back to tenured members.
- This week we will be going over recursive sorting algorithms like *QuickSort* and *MergeSort*
- Recall in CSC108 we defind the following sorting algorithms:
  - Bubble Sort - Worst case $\mathcal{O}(n^2)$
  - Selection Sort - Worst case $\mathcal{O}(n^2)$
  - Insertion Sort - Worst case $\mathcal{O}(n^2)$
- Today we will show how we can improve on these algorithms using recursion, achieving $\mathcal{O}(n \log(n))$ time complexity
- https://www.toptal.com/developers/sorting-algorithms
- **Fun Fact:** In *CSC263*, you will use *STA256* knowledge to analyze the time-complexity of randomized-quicksort using Expected Values and Indicator Random Variables

Welcome back to IbraFSGs™

- Welcome back to IbraFSGs™! Hello to new people and welcome back to tenured members.
- This week we will be going over recursive sorting algorithms like *QuickSort* and *MergeSort*
- Recall in CSC108 we defind the following sorting algorithms:
  - Bubble Sort - Worst case $\mathcal{O}(n^2)$
  - Selection Sort - Worst case $\mathcal{O}(n^2)$
  - Insertion Sort - Worst case $\mathcal{O}(n^2)$
- Today we will show how we can improve on these algorithms using recursion, achieving $\mathcal{O}(n \log(n))$ time complexity
- https://www.toptal.com/developers/sorting-algorithms
- **Fun Fact:** In *CSC263*, you will use *STA256* knowledge to analyze the time-complexity of randomized-quicksort using Expected Values and Indicator Random Variables
- **Fun Fact #2:** In CSC236, you will learn how to formally prove the correctness of these algorithms using loop invariants and induction

Tl;Dr CS gets real hard real fast.

A Recap of the UltraSheet™

- An *UltraSheet™* is a "cheat sheet" that you compile for **yourself** to review course materials
  - Sharing UltraSheets™ is **counter-productive** and **will not help you learn the material**
  - However, reviewing content in a group and simultaneously updating your UltraSheets™ is a good idea
- It acts like your own personalized textbook chapter
  - It allows you to **regurgitate all the course information in a contiguous, organized manner** and helps you **find gaps in your knowledge**
  - You should **not** be copying the textbook or lecture slides verbatim; You should be **summarizing** the content in your own words while tying in examples and analogies
- UltraSheets™ help with type 1 and 2 questions
- In case you didn't notice yet, Petersen *loves Type 1* and *Type 2* questions - Use this information how you will ;P

Key Terms

Required Key Terms: The following key terms are required for this week's content. You should be able to define and explain these terms in your UltraSheets™:

- **Divide and Conquer**

Key Terms

Required Key Terms: The following key terms are required for this week's content. You should be able to define and explain these terms in your UltraSheets™:

- **Divide and Conquer**
  - **This is a BIG one!** It comes back to **haunt** you in CSC236

Key Terms

Required Key Terms: The following key terms are required for this week's content. You should be able to define and explain these terms in your UltraSheets™:

- **Divide and Conquer**
  - **This is a BIG one!** It comes back to **haunt** you in CSC236
- **QuickSort**

Key Terms

Required Key Terms: The following key terms are required for this week's content. You should be able to define and explain these terms in your UltraSheets™:

- **Divide and Conquer**
  - **This is a BIG one!** It comes back to **haunt** you in CSC236
- **QuickSort**
  - *Partitioning*

Key Terms

Required Key Terms: The following key terms are required for this week's content. You should be able to define and explain these terms in your UltraSheets™:

- **Divide and Conquer**
  - **This is a BIG one!** It comes back to **haunt** you in CSC236
- **QuickSort**
  - *Partitioning*
  - *Pivot*

Key Terms

Required Key Terms: The following key terms are required for this week's content. You should be able to define and explain these terms in your UltraSheets™:

- **Divide and Conquer**
  - **This is a BIG one!** It comes back to **haunt** you in CSC236
- **QuickSort**
  - *Partitioning*
  - *Pivot*
- **MergeSort**

Key Terms

Required Key Terms: The following key terms are required for this week's content. You should be able to define and explain these terms in your UltraSheets™:

- **Divide and Conquer**
  - **This is a BIG one!** It comes back to **haunt** you in CSC236
- **QuickSort**
  - *Partitioning*
  - *Pivot*
- **MergeSort**
  - *Merging*

Key Terms (Cont'd)

Suggested Key Terms: The following key terms are *recommended* for further studying this week's content. Most are out of the scope of the course and are **not** required.

- **Big-O Notation ($\mathcal{O}(n)$)**
  - Upper Bound
  - A function $f(n)$ is $\mathcal{O}(g(n))$ if there exists a constant $c > 0$ and $n_0 > 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$
- **Big-Omega Notation ($\Omega(n)$)**
  - Lower Bound
  - A function $f(n)$ is $\Omega(g(n))$ if there exists a constant $c > 0$ and $n_0 > 0$ such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$
- **Big-Theta Notation ($\Theta(n)$)**
  - Tight Bound
  - A function $f(n)$ is $\Theta(g(n))$ if and only if $f(n)$ is $\mathcal{O}(g(n))$ and $f(n)$ is $\Omega(g(n))$
  - In other words; $f(n)$ is $\Theta(g(n))$ if and only if there exist constants $c_1 > 0$, $c_2 > 0$, and $n_0 > 0$ such that $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0$

Note: These are CSC236 concepts, and are **not** required for CSC148. I am including these to complement your lecture on efficiency and complexity next week.

Key Terms (Cont'd)

Suggested Key Terms: The following key terms are *recommended* for further studying this week's content. Most are out of the scope of the course and are **not** required.

- **TimSort (Ibrahim's Favourite)**
  - A hybrid sorting algorithm derived from merge sort and insertion sort
  - Used in Python's `sorted()` function
  - Will haunt you in this weeks lab
- **HeapSort - CSC263**B
  - A comparison-based sorting algorithm
  - Uses a binary heap data structure
- **RadixSort (Prof. Petersen's Favourite)**
  - A non-comparative integer sorting algorithm
  - Sorts integers by processing individual digits

Jamboard Link

https://tinyurl.com/ibrafsg0327

0327 for March 27th

Note: Jamboard isn't ideal because of its limitations, hence I'm working on my own alternative. Stay tuned!

Practice Problem I Prelude

Is *QuickSort* always $\mathcal{O}(n\log(n))$? Is *MergeSort* always $\mathcal{O}(n\log(n))$? Justify your answer.
Hint: Consider what this depends on

Practice Problem I Prelude

Is *QuickSort* always $\mathcal{O}(n \log(n))$? Is *MergeSort* always $\mathcal{O}(n \log(n))$? Justify your answer.
Hint: Consider what this depends on
Hint # 2: Which data structure and traversal is most similar to QuickSort?

Practice Problem I: SortTheory

Dr. Halstead and Dr. Manning are arguing about the new hospital database system sorting upgrades. Dr. Manning argues Quicksort is the faster option, while Dr. Halstead is a firm believer in the efficiency mergesort. With the help of April, Dr. Choi runs some tests with both sorting algorithms on the hospital dataset. Here is what you know about the dataset:

1. 99% of the time, the hospitals dataset is mostly sorted
2. All the sorting algorithms are run in their non-in-place variants
3. Every other part of their software is as optimized as it can get - There is no other source of slowdowns other than the choice of algorithm

Note: Since these are doctors and not computer scientists, they do not know about TimSort (which would've been the ideal compromise for Will and Natalie)
Given what we know about Gaffney Chicago's database, what did Dr. Choi and April find in their tests?
Justify your answer

Practice Problem I: SortTheory

Dr. Halstead and Dr. Manning are arguing about the new hospital database system sorting upgrades. Dr. Manning argues Quicksort is the faster option, while Dr. Halstead is a firm believer in the efficiency mergesort. With the help of April, Dr. Choi runs some tests with both sorting algorithms on the hospital dataset. Here is what you know about the dataset:

1. 99% of the time, the hospitals dataset is mostly sorted
2. All the sorting algorithms are run in their non-in-place variants
3. Every other part of their software is as optimized as it can get - There is no other source of slowdowns other than the choice of algorithm

Note: Since these are doctors and not computer scientists, they do not know about TimSort (which would've been the ideal compromise for Will and Natalie)
Given what we know about Gaffney Chicago's database, what did Dr. Choi and April find in their tests?
Justify your answer
Pay attention to context clues in the question

Practice Problem II: QuickSort Troubles

Draw the resulting BSTs for the following lists, assuming the pivot is the element you will always insert first. What is the complexity of inserting the elements in the order given? What does this mean for the BST?

- `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
- `[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]`
- `[5, 3, 7, 2, 4, 6, 8, 1, 9, 10]`

Practice Problem II: QuickSort Troubles

Draw the resulting BSTs for the following lists, assuming the pivot is the element you will always insert first. What is the complexity of inserting the elements in the order given? What does this mean for the BST?

- `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
- `[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]`
- `[5, 3, 7, 2, 4, 6, 8, 1, 9, 10]`

Note: This is a visualization of the *QuickSort* algorithm, but is not an accurate representation of what actually happens in the algorithm
QuickSort can be simplified into constant insertion into a BST then an inorder traversal of the BST

Practice Problem III: MergeSort Troubles - If we have time

Draw the resulting Binary Tree for the following lists, following the MergeSort sorting algorithm.

- `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
- `[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]`
- `[5, 3, 7, 2, 4, 6, 8, 1, 9, 10]`

Practice Problem III: MergeSort Troubles - If we have time

Draw the resulting Binary Tree for the following lists, following the MergeSort sorting algorithm.

- `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
- `[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]`
- `[5, 3, 7, 2, 4, 6, 8, 1, 9, 10]`

Note: Do you see now why there are speed differences between QuickSort and MergeSort?

Thank you for coming!

Figure 1: Image courtsey of Looney Tunes™ and Warner Bros.™