CS 305
Design and Analysis of Algorithms

09 / 10 / 2024

Instructor: Michael Eckmann

Today's Topics

- Questions / Comments?
- Arithmetic Series
- SelectionSort, InsertionSort
- MergeSort
- Recursion tree analysis for MergeSort runtime
- Start Recurrence Relations for Divide and Conquer algorithms

Arithmetic Series

- Let me introduce Arithmetic Series on the board.
- This is one of 3 sums you should memorize.
- The other two being Geometric and Harmonic.
- See Appendix A of our text (pgs. 1141, 1142)

Arithmetic Series

Let's prove the sum is big $O(n^2)$ on the board

I leave you to prove that it is also Big Omega of n^2

Which means that it is Big Theta of n^2

Sorting algorithms

- Reminder of what Selection Sort and Insertion Sort do (online animation).
- Let's discuss Selection Sort.
- Keep current "last".
- Continually find max element index in list and swap with the current "last".
  - How long does 1 find max take?
  - How many times do we do find max?
  - Are there any best or worst case situtations? All the same?

Sorting Algorithms

- Insertion Sort pseudocode on the board with run time analysis.
- Are there any best or worst case situtations? All the same?

Sorting Algorithms

- Let's look at an instance of Merge Sort and see how it works.
- Are there any best or worst case situtations? All the same?

MergeSort pseudocode
see code in section 2.3 of CLRS

Sorting Algorithms

- Let's see a recursion tree for n elements of MergeSort.
  - How many elements processed in each level?
  - How many levels?

Sorting Algorithms

- Additional comments on the sorts
  - InsertionSort and SelectionSort are both in-place algorithms and have small constants (as compared to MergeSort)
    - InsertionSort and SelectionSort each require constant extra space Big Theta (1)
  - MergeSort is NOT in-place (requires n additional space) and has large constants
    - MergeSort extra space analysis: Big Theta (n)

Recurrence Relations

- For MergeSort
- Time(n) = 2 * Time(n/2) + n
  - Number of recursive calls is 2
  - The time for one recursive call is Time(n/2)
  - The time in a call is n
- In general for divide and conquer
- T(n) = a * T(n/b) + f(n)
  - Number of recursive calls is a
  - n/b is size of list in a recursive call
  - The time in a call is f(n)

Recurrence Relations

- In other words, a divide and conquer algorithm that creates *a* subproblems each a factor of *1/b* the size of the original problem and takes *f(n)* amount of time to do the divide and combine steps.
- T(n) = a * T(n/b) + f(n)
    - Number of recursive calls is **a**
    - **n/b** is size of list in a recursive call
    - The time in a call is **f(n)**

Recurrence Relations

- Let me draw the recursion tree for arbitrary a and b.
- How many leaves?
- Let's see an example with explicit values for a and b.
- How many leaves?
- $T(n) = a * T(n/b) + f(n)$
- Let's prove, by induction, that the number of leaves in a perfect binary tree is 2^d, where d = depth of the tree.