Contents

Notes: By Nota

Nota

Nota, is a simple script to manage notes. It manages all my class notes which are markdown files and compiles them to PDF versions. This is so people can see my notes and I can share them easier. Not only that but the PDF version makes for easier reading, while the markdown versions are easy to edit and easy to search for items. Hope you enjoy!

Topic: Running Times

Algorithm Analysis

- Space Efficiency: Somewhat Important
- Time Efficiency: **SUPER IMPORTANT!**

Running Time

`Running Time = T(n)` - N is usually the size of the input: - Number of items to sort - Number of items to search - Size of objects

- Cases:
  - Worst Case (Most Common)
  - Average Case
  - Amortized
  - Best Case

- Factors To Ignore:
  - Small Input Size
  - Speed of the Machine

Big O

Formula

$$n \geq n_0, f(n) \leq g(n)$$

Models of Computation

- A mathematical model that represents the actual computers on which algorithms will be run
- Provides a way to analyze algorithms without having to actually run them
- Examples:
  - Turning Machine (TM)
  - Random Access Machine (RAM)
  - Parallel Random Access Machine (PRAM)
- RAM: Rules for running-time analysis
  1. Each simple arithmetic operation takes constant time
  2. Each assignment takes constant time
  3. Running time of a sequence is the sum of each statement
  4. Running time of an if is the sum of all sections
  5. Running time of a loop is iterations times body
  6. Nested loops are Rule 5 from inside out

Topic: ADTs

- A description of a data structure containing:
  - I. Some information about how the data is organized (maybe)
  - II. A list of primitive operations that acess or modify the data
  - **No Implementation Details**

ADT #01- List ADT

An ordered sequence of elements (not necessarily sorted)

Primitive Operations

- `Length(list)` - Returns the number of elements in the list
- `GetFirst(list)` - Returns the first element in the list
- `GetLast(list)` - Returns the last element in the list
- `Prepend(list, x)` - Inserts x into list at the beginning
- `Append(list, x)` - Inserts x into list at the end
- `RemoveFirst(list)` - Removes the first element in the list
- `RemoveLast(list)` - Removes the last element in the list
- `CreateEmptyList()` - Reurns a newly created, empty list

- `IsEmpty(list)` - Returns `True` if list has no elements, else `False`

Implementations

Array

Description

Continuous block of memory which is not dynamically allocated. Ex. Java.

Advantages

- Easy to work with and write
- Easy access to any element within the array

Disadvantages

- O(n) time to insert new elements
- Memory allocation issues
- Can't increase size without O(n)

Linked List

Description

- There are two types of objects - Node & Header:
  - List elements are stored in the nodes
  - Header is used to access the list

Advantages

- Improved running time over array
- Solves storage problems since it's dynamic

Disadvantages

- Harder to work with and implement
- Inserting elements is still O(n)

Summary

List ADTs are used to store information in an categorized, non sorted fashion. The two implementations we studied are an Array with extra buffer space and a linked list. While the array could do most operations quickly, prepend was innefficient and the extra buffer space meant a waste of memory that didn't have a purpose. Basically memory leaking by using this method. The Linked List with a header allowed for faster times of pretty much all the operations except append which required a loop to the second to last item in the list.

Topic: Sorting Algorithms

Merge Sort

Merge Sort

- Split the array into two halves. Recursively sort each half.
- To merge 2 sorted arrays, create an empty array to hold the combined. Put a finger at the beginning of both. Copy the lesser of the two keys into the next empty cell and advance one position. Continue doing this till one array runs out. Move the rest of the array.

Quick Sort

- Partition the array so that everything in the left part is less than everything in the right part.
- Recusively sort each half. >All the work is done in the partition algorithm

Selection Sort

- Find the smallest element in the array and swap into the first cell. The first cell is now correct so don't touch it. Repeat the process until each cell is correct.

Insertion Sort

- Divided the array into sorted(left) and unsorted(right) sections. Start the size of the section sorted at 1. Take the first element following the sorted secioin and march it down into it's correct position in the sorted section, increasing the size of the sorted section by 1 for each element.

Bubble Sort

- From beginning to end, compare each element to it's neighbor, swapping if they are out of place. After each run, one more element is in its correct position at the end of the array, so repeat n *(really n-1)* times.