CS162 Lecture 23

Data Structures
Housekeeping

Reminder: Assignment 5 Due this Sunday!

Only turn in list.c!

No Design Document!

Pop quiz: Compile with?

((((gcc)))))
Data Structures

- The conceptual share or arrangement of the data
- Arrays:
  - Stored in contiguous memory
  - Random access to data elements
  - Same data type
- Lists:
  - Not stored contiguously
  - No random access
  - Same data type
  - Sequential Ordering
Abstract Data Type - Stack

Stack: entries are only inserted and removed at the head

- Last in, First out (LIFO)
- Push: add to the top/front
- Pop: remove from the top/front
- Ideal for storing items that must be retrieved in the reverse order from which they are stored
- Examples:
  - Stack of plates
  - Recursion:
    - sum(list)
      - If len == 0
        - return val
      - return sum(list - 1) + val
Abstract Data Type - Queue

Queue: entries only removed at the front, entries only added to the back

- FIFO
- Push: add to the back
- Pop: remove from the front
- Examples:
  - Printing at Library
  - Customer Service Call
  - buffer
Abstract Data Type - Trees

- Way more general than it sounds.
  - Technically: Any collection whose entries have a hierarchical organization
- Each position is called a node
- Node at the top is the root node
- Notes at the end are called terminal nodes (leaves)
- Binary tree: a tree where each parent has no more than two children
  - Uniquely useful; you may catch me using “tree” when I mean “binary tree” because this is so always what we actually use.
  - You’ll know you’re in a different class when you hear ‘red black tree’
- Examples:
  - Ancestry
  - Employee Hierarchy
Runtime Complexity

- Algorithms take time to run
- Clock time varies
  - Can vary based on input
  - Can very based on number and kind of steps
- Typically talk about runtime in an abstract sense
  - Big O - worst base
  - Big Omega - best case
  - Big Theta - average case
## Different Times

<table>
<thead>
<tr>
<th>$O(1)$</th>
<th>constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(\log n)$</td>
<td>logarithmic</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>linear</td>
</tr>
<tr>
<td>$O(n \log n)$</td>
<td>“n log n”</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>quadratic</td>
</tr>
<tr>
<td>$O(n^3)$</td>
<td>cubic</td>
</tr>
<tr>
<td>$n^{O(1)}$</td>
<td>polynomial</td>
</tr>
<tr>
<td>$2^{O(n)}$</td>
<td>exponential</td>
</tr>
</tbody>
</table>
O(1) algorithm example: push()

struct node* push (struct node * head, int n) {
    struct node *temp = malloc(sizeof(struct node));
    temp ->val = n;
    temp ->next = head;
    head = temp;
    return head;
}
O(n) algorithm example: length()

```c
int length(struct node *head) {
    int n = 0;
    while(head != NULL) {
        n++;
        head = head->next;
    }
    return n;
}
```
O(n^2) algorithm example: bubble_sort

```c
void bubble_sort(struct node *head, int size) {
    ... 
    int iteration, il
    for(iteration=1; iteration<size; iteration++) {
        for(i=0; i<size-iteration; i++) {
            if(current->val > current->next->val) {
                //swap values
            }
            //move current to next node
        }
        current = head;
    }
}
```
O(log n) algorithm example: binarySearch

```c
int binarySearch(const int list[], int length, int item) {
    int first = 0, last = length - 1, mid;
    int found = 0;
    while (first <= last && found == 0) {
        mid = (first + last) / 2;
        if (list[mid] == item)
            found = 1;
        else if (list[mid] > item)
            last = mid - 1;
        else
            first = mid + 1;
    }
    return found==1?mid:-1; // if found return mid, else return -1 (not in list[])
} //end binarySearch
```