ECE 242 FALL 2015
Data Structures and Algorithms

http://www.ecs.umass.edu/~polizzi/Teaching/ECE242/

Review Mid-Semester

Prof. Eric Polizzi
Lectures 1-2-3: Arrays

- **Array have fixed size** – good estimate of the number of items is necessary
- Review of Object oriented class- to ease the life of the user (a higharray class handles privately indexes and number of items)
- Introduction to Big-O notation
  - Indicates how the running time $T$ is affected by $\# \text{items } N$
  - Allows relative comparisons of algorithms performances
    \[ O(1) < O(\log N) < O(N \log N) < O(N) < O(N^2) < O(N^3) < O(2^n) \]
  - Asymptotic analysis of functions and analysis of algorithm complexity (discussed various examples)
- Unordered arrays offer: fast insertion $O(1)$ but slow linear search and deletion $O(N)$
- Ordered arrays arrays offer: **fast binary search** $O(\log N)$ but slow insertion and deletion $O(N)$ (items need to be shifted/copies)
- **Ordered array** useful in situation where once the array is ordered, search are very frequent but insertion/deletion are not - Example: search words in dictionary
Lectures 4-5: Simple Sorting

Main Step - sort N items:

- Start from the left
- Compare neighbors and swap them if not in order
- Move one position right and go back to previous step

Cost: N-1 comparisons and 0 to N-1 swaps

Go Back to Main Step but sort for the first N-1 items left
Lectures 4-5: Simple Sorting

Main Step - sort N items:
- Find the smallest item
- Swap it with the item at first position

Cost:
N-1 comparisons and 1 swap

Go Back to Main Step but sort for the last N-1 items left
### Lectures 4-5: Simple Sorting

<table>
<thead>
<tr>
<th>Sorting Algorithm</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bubble Sort</td>
<td>$O(N^2)$ comp.+ $O(N^2)$ swaps</td>
</tr>
<tr>
<td>Selection Sort</td>
<td>$O(N^2)$ comp.+ $O(N)$ swaps</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>$O(N^2)$ comp.+ $O(N^2)$ copies</td>
</tr>
</tbody>
</table>

#### Insertion Sort

- **Outer-loop**: pivot selection shifts from 0 to N-1. The pivot at position $j$ is inserted into the ordered array $[0:j-1]$
- For each pivot, this insertion requires:
  - a linear search: $O(N)$ comparisons
  - $O(N)$ copies for shifting the items

- O(N) total if data are already almost sorted!
- Insertion sort basic idea can be found in many other sorting algorithms: enhanced version, linked list insertion sort, shellsort
Lectures 4-5: Simple Sorting

<table>
<thead>
<tr>
<th>Simple Sorting</th>
<th>Time Complexity</th>
<th>Additional Info</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bubble Sort</strong></td>
<td>O(N^2)</td>
<td>O(N^2) comp.+ O(N^2) swaps</td>
</tr>
<tr>
<td><strong>Selection Sort</strong></td>
<td>O(N^2)</td>
<td>O(N^2) comp.+ O(N) swaps</td>
</tr>
<tr>
<td><strong>Insertion Sort</strong></td>
<td>O(N^2)</td>
<td>O(N^2) comp.+ O(N^2) copies</td>
</tr>
<tr>
<td><strong>Enhanced Insertion Sort</strong></td>
<td>O(NlogN) comp.+ O(N^2) copies</td>
<td>Use binary search rather than linear search</td>
</tr>
</tbody>
</table>

- For each pivot, the insertion requires:
  - a binary search: O(logN) comparisons
  - O(N) copies for shifting the items

![Image of sorted array with key (pivot) and locally sorted highlighted]
Lectures 6-9: Stacks and Queues

- Stacks and Queues are **abstract data structures**; Complexity $O(1)$.

**Stack (LIFO)**

Basic operations using a Stack
- push(item); pop(); peek()
- isEmpty(); isFull(); size()

**Queue (FIFO)**

Basic operations using a Queue
- enqueue(item); dequeue(); peek()
- isEmpty(); isFull(); size()
Lectures 6-9: Stacks and Queues

- **Stacks-** Selected Applications:
  - Delimiter matching: a[b(c[d]e)f] matching?
  - Parsing arithmetic expression- Evaluate RPN (postfix expression)
  - Example: “2+3=” (infix) becomes “23+” (postfix)

```java
class StackApp {
    public static void main(String[] args) {
        Stack mystack = new Stack(5);
        mystack.push(2);
        mystack.push(3);
        mystack.add();
        System.out.print(mystack.peek());
    }
}
```
Linked-List:
- Take as much memory as needed
- No need to move items around

Abstract Data Type (ADT) must implement User Interfaces
Lectures 10-13: Linked-List

- **Simple Linked-list**
  - InsertFirst, O(1)- push
  - DeleteFirst, O(1)-pop
  - DisplayList, O(N)
  - Find, O(N)
  - Delete, O(N)

- **Simple double-ended**
  - InsertLast, O(1)
  - InsertLast enqueue
  - DeleteFirst dequeue

- **Doubly linked-list**
  - Traverse backward
  - More flexibility in removing, inserting objects
  - DeleteLast O(1)
Lectures 14-15: Recursion

Recursion

- Programming technique in which a method calls itself
- Uses **Base step**+**Recursive step**
- Well-suited for the “**divide and conquer**” strategy

Various Examples

- Simple Recursion:
  - Triangular number: \( \text{sum}(n) = \text{sum}(n-1) + n \) with \( \text{sum}(1) = 1 \)
  - Factorials: \( \text{fact}(n) = \text{fact}(n-1) \times n \) with \( \text{fact}(1) = 1 \)
  - Anagrams: find all the \( n! \) permutations for \( n \) letters
  - Binary search: keeps dividing in half
  - Display Linked list

- Two recursions (divide and conquer)
  - Fibonacci (2 recursions) \( \text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2) \) with \( \text{fib}(1) = 1, \text{fib}(0) = 0 \)
  - Hanoi Tower
  - MergeSort

```java
int fact(int n) {
    if (n==1) // base step
        return 1;
    else // recursive step
    {
        int temp=n*fact(n-1);
        return temp
    }
}
```
public void recMergeSort(int[] array, int lower, int upper)
{
    if (lower==upper)  // no need to sort-base step
        return;
    else
    {
        int mid=(lower+upper)/2;  // find midpoint
        recMergeSort(array,lower,mid);  // sort low-half
        recMergeSort(array,mid+1,upper);  // sort high-half
        merge(array,lower,mid+1,upper);  // merge them- textbook p289
    }
}
Lectures 16-17: Advanced Sorting

- Improvement of insertion sort using increment
  - large increment: smaller sub-list decrease the number of copies/comp.
  - Small increment: larger sublist but better sorted so less copies/comp.
- Various increment sequences possible, for example $h=3h+1$

<table>
<thead>
<tr>
<th>ShellSort</th>
<th>$O(N(\log N)^2)$ in average</th>
<th>'Insertion sort' using increment</th>
</tr>
</thead>
</table>

increment of 5
(example)
Lectures 16-17: Advanced Sorting

- Partitioning: action of dividing depending on key value (pivot)

QuickSort

- O(NlogN)
- Comp.>swaps
- Divide and Conquer
- Uses partitioning recursively

```java
public void recQuickSort(int left, int right)
{
    // size 1- base step- sorted
    if (right-left<=0)  return;
    // size 2 or larger- recursive step
    else{
        // partition range using pivot at right and return pivotIndex
        int pivotIndex=partition(left,right,array[right]);
        // sort left side
        recQuickSort(left,pivotIndex-1);
        // sort right side
        recQuickSort(pivotIndex+1,right);
    }
}
```
That's all folks (dance)