What we will do today

- Look at the basic topics in these areas
  - More of a listing of topics and key points than explanation
  - NOT all topics are mentioned – just what I prepared when I took the exam.
  - You are responsible for referring to the syllabus and prescribed text book for complete information and using that as a guide for your preparation

- Discuss the previous years exams
  - Again, these are my solutions, not necessarily THE CORRECT solutions

- This WILL BE VERY FAST as we have only an hour
Algorithms and Data Structures
Topics

- Arrays and Matrices
- Stacks, Queues, Priority Queues, D Queues
- Lists
- Graphs
- Trees and Heaps

- Algorithm Complexity
- Recursion
- Searching
- Sorting
- Hashing
Algorithm’s Order of Complexity

- Speed of our algorithm for large input sizes - $n$
- A function $f(n)$ is $O(g(n))$ if “increase” of $f(n)$ is not faster than that of $g(n)$
- $\Omega(n)$ – lower bound, $o(n)$ – asymptotic equality
Arrays

- Collection of Homogenous entities
- Index – position of an element – used to access the element
- Array methods
  - $A = \text{char}[\text{index}]$\quad O(1)
  - $\text{Char}[\text{index}] = \text{‘A’}$\quad O(1)
  - Sequential Search\quad O(n)
  - Binary Search (Sorted)\quad O(\log n)
  - Insertion and Removal\quad O(n)
Matrices

- Multi dimensional arrays
  - Set of indices
  - $O(1)$ – look up
  - $O(k*l*m)$ – Sequential access

- Sparse Matrix – Matrix populated primarily by same element 0
  - Store only non zeros with their indices – lesser memory
  - $j^{th}$ non zero element - value $a$, index $i$
  - Specialised algorithms for addition etc

- Read up more from text book

\[ A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \]

\[
\begin{array}{c|ccc}
\bullet & 2 & 5 & 1 \\
x[j].a & 5 & 3 & 7 \\
\end{array}
\]
## Stacks and Queues

- **LIFO structure**
- **Push, Pop, Peek, Isempty, Isfull, TOS – O(1)**

In a stack, all operations take place at the "top" of the stack. The "push" operation adds an item to the top of the stack. The "pop" operation removes the item on the top of the stack and returns it.

<table>
<thead>
<tr>
<th>Before push(83)</th>
<th>After push(83)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>123</td>
<td>123</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Push</th>
<th>Stack</th>
<th>Pop</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>A</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>25</td>
<td>A</td>
<td>D</td>
<td>25</td>
</tr>
<tr>
<td>123</td>
<td>A</td>
<td>D</td>
<td>123</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>D</td>
<td>5</td>
</tr>
<tr>
<td>17</td>
<td>A</td>
<td>D</td>
<td>17</td>
</tr>
</tbody>
</table>

- **FIFO structure**
- **EnQ (add to tail), DeQ (Remove from head)**
- **Peek, Isempty, Isfull**
- **Both – O(1)**
Priority Queues, DQueues

- PQs – Similar to Qs but the elements are ordered by priority
  - So DeQ – removes (lowest) highest priority item
  - EnQ – O(n), DeQ – O(1)

- DQs – Add / Remove from both front and back of the structure
  - Push(), Pop(), EnQ() and DeQ() operations defined
  - So Stack and Qs – specialized form of DQs

- Postfix and Prefix notations – refer to the text
Linked Lists

- Data type in which “next” element – not defined by position but explicitly linked to
  - Singly linked
  - Doubly linked
  - Circularly linked
    - Singly circular
    - Doubly circular
  - Double ended list
  - Sorted list
  - Sentinel nodes

- List Operations
  - Find – (value), next(), before()
  - Insert – Before() / after()
  - Delete – Before() /after() / (value)

- May have some other special operations depending on type of List and implementation

- Same operation – different complexity for different type of list
Recursion

- Define a function in terms of itself
- Equivalent of the mathematical technique of induction

```
function Factorial(x: integer): integer;
Begin
  if x <= 1 then
    Factorial := 1
  else
    Factorial := x * Factorial(x-1);
end
```

Is recursion more efficient than a while or a for loop?

- Tradeoff between overhead of calling function and history of previous states that needs to be stored
Searching

- Finding if a specified value exists
  - Linear Search $O(n)$
  - Binary search $O(\log n)$
  - Hash Table – $O(1)$ to $O(n)$
  - Tree based searches – BFS and DFS
  - Graph based searches - Djikstra’s, Kruskal’s, Prim’s
Simple Sort algorithms

- Bubble sort – $O(n^2)$ worst
- Selection sort – $O(n^2)$ worst, but lesser number of swaps
- Insertion sort - $O(n^2)$ worst, $O(n^2/4)$ average
- Quick sort – $O(n \log n)$ average, $O(n^2)$ worst
- Merge sort – $O(n \log n)$ average and worst
- Shell sort - $O(n^2)$ worst, may be optimized to $O(n \log^2 n)$
Hash tables

- Index a function of data stored
- Hashing functions
- Collisions
  - Open addressing
    - Linear probing
    - Quadratic Probing
    - Double Hashing
  - Chaining
- Efficiency of hashing
- Limitations of hashing
Graphs

- $G=(V,E)$
  - paths – directed, weighted
  - Adjacent vertices

- Types of graphs
  - Connected
  - Cyclic
  - Minimum Spanning Tree (n-1 E)
    - Prim Jarnik

- Representing a graph
  - Adjacency matrix
  - Adjacency list

- Traversal
  - DFS – A B F H C D G I E
  - BFS – A B C D E F G H I

- Shortest Path - Djikstra
Trees

- Connected acyclic Graph
- Parent – child, Root, leaf, Subtree
- Level
- Traversal, visit,

- Binary Tree
- Finding a node – similar to binary search from root
Tree Traversal

- $V = \text{vertex, } L = \text{left, } R = \text{right}$

- **Preorder (VLR)**
  - F, B, A, D, C, E, G, I, H

- **Postorder (LRV)**
  - A, C, E, D, B, H, I, G, F

- **In-order (LVR)**
  - A, B, C, D, E, F, G, H, I

- Level-order traversal yields: F, B, G, A, D, I, C, E, H

- Inserting / Deleting a node
Advanced Topics in Trees

- Red Black Trees
- AVL Trees
- 2,3,4 Trees

- Basic operations like
  - How to make a vector into this tree
    - Addition of a node
    - Balancing the tree
  - How to delete a node
  - Searching for a value
Heap

- Binary tree with 2 properties
  - Level – All levels are complete, at least filled from left to right
  - Heap – All nodes are not greater (lesser) than its children

- Unlike Binary Tree – no L < V < R – much more loosely arranged – so search is almost O(n)

- But Removemax and Add (node) are fast O(logn) – so efficient to implement Priority Qs

- Heapify() – make a tree after insert / removal into a Heap – O(logn)

- Heap sort – n insertions and heapify – O(n logn)
Old Question Paper Solutions

- These are not official solutions.

- These are not necessarily THE CORRECT solutions.
Consider a hash table with \( S \) entries that is used to store \( N \) records, \( S > N \).

a) Under the assumption of perfect hashing, what is the algorithmic complexity (using \( O() \) notation) of inserting a new element to the table?

1) Using the hash function to calculate the address in which this data should be stored
2) Since Ideal – No collision at that location
3) Store the data at that location

Constant Time – \( O(1) \)
b) What is the complexity of hash table lookups? Explain your answer.

1) Using the hash function to calculate the address in which this data should be stored
2) Since Ideal – No collision at that location – So no need to explore other locations
3) Access the data from that location

Constant Time – O(1)
Describe two techniques that can be used to handle collisions during hash table insertion, and discuss their relative advantages and disadvantages. Illustrate with an example.

- **Open Addressing**
  - At low load factor (Proportion of used slots) the number of probes required to find a location is small. So storage and access is close to O(1).
  - But as load factor increases the insertion and lookup take close to O(n) – if they terminate at all.

- **Chaining**
  - Works good for high load factor. Will terminate. Performance better than O(n).
  - But at small load factor the overhead of maintaining lists is too high.
c) Under the assumption of imperfect hashing function, what is the worst-case complexity of hash table lookups? Explain your answer.

- Using the hash function to calculate the address in which this data should be stored.
- Since imperfect hashing – collisions could have occurred. By some method (probing or chaining) this collision would have been resolved.
- If all the data stored in the table had the same hash – then this would involves searching all the locations – going through n locations.
- Find the data and access it – \( O(n) \)
Consider a vector $V[1..N]$ which contains $N$ positive integer entries.

a) Use pseudo-code to describe an algorithm $j=$FindLargest($V$) which returns $j$ as the *index* of the largest entry of vector $V$ (without modifying its contents) with complexity $O(N)$.

Findlargest($V$)
{
    $j=0$
    for $i = 1:1:$Length($V$)
        If $V(i)>V(j)$ Max = i
    End for
}

<table>
<thead>
<tr>
<th>3</th>
<th>2</th>
<th>8</th>
<th>1</th>
<th>2</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
Use pseudo-code to describe an algorithm $S=\text{Sort}(V)$ which returns a sorted vector $S[1..N]$ and uses $\text{FindLargest}(V)$ as a function to assist in the sorting of the entries in $V$. You can use temporary variables and vectors. What is the complexity of your sort algorithm?

Selectionsort($V$)
{
    Int j, swap
    for i=length($V$):1:1
    {
        jtemp = $\text{FindLargest}(V[0:i-1])$
        swap = $V[j]$; $V[j] = V[i]$; $V[i] = \text{swap}$;
    }
    O(N²)
}

O(N²)

http://www.cs.ust.hk/faculty/tcpong/cs102/summer96/aids/select.html
Consider a heap data structure implemented as a binary tree. What is the property that must be satisfied for every node “i” (except the root node) of the binary tree for it to be a heap?

For a Heap – 2 properties

- Level Property
- Heap Property – Every node except the root should not be greater than its parent
2006 Question

d) \text{p=FindLargest}(H) \text{ returns a } \textit{pointer} \text{ to the heap node which stores the largest element of the heap (without modifying its contents). Can this algorithm achieve lower complexity than the algorithm in a)}

- By Heap property – largest node is root
- So Findlargest(H) - O(1)

- If ascending heap (the root is the smallest element), then the leaf is the largest element.
  - To get to the leaf level – O(log N)
  - Number of nodes in leaf level – O(N)
  - So Findlargest(H) = O(N)
2006 Question

e) Describe how the Sort() algorithm would need to be modified to work with a heap data structure and the p=FindLargest(H) function. What is the complexity of the modified Sort() algorithm?

Heapsort(H, A)
{
    for i = 0:1:N
        O(N)
        p=Findlargest(H)
        O(1)
        A[i] = remove_element(H, p)
        O(1)
        Heapify(H)
        O(log N)
    end for
}
O(NlogN)
Operating Systems – Review
Topics

- General Introduction
- Processes
- I/O
- Memory Management
- File System
General Introduction

- What is an OS?
- Types of OS
  - Layered
  - Monolithic
  - Virtual Machines
  - Client Server
- System Call
Processes

- Process model
  - What is a process?
  - States – Running, ready, Blocked
  - Multiprogramming and Process Hierarchies – fork()
  - Process Table - Context
  - What is a system call

- Threads
  - Difference between process and thread

- IPC
  - Race, Deadlocks, Livelocks
  - Critical Section
  - Mutex – Mutual Exclusion
Various Mutex schemes
- Disabling Interrupts
- Lock – test a Lock variable – if free – set it and proceed
  - But now Mutex for lock variable
- Petersons Solution – enter_region(pid), leave_region(pid)
  - Process can cheat by not calling leave_region()
- TSL – atomic H/W instruction
  - Requires H/W Support

- Sleep and Wakeup – Producer Consumer Problem
  - Missed wakeup calls (The Grad Student Syndrome)
- Semaphores – Producer Consumer Problem
  - Complicated code, may lead to deadlock if written wrong
- Monitor
  - Encapsulate CS with a wrapper which checks for Mutex
- Message Passing
  - Requires careful design
IPC and Scheduling

- IPC Heuristics
  - Dining Philosophers, Sleeping Barber and Readers Writers

- Scheduling
  - Requirements of a good scheduling algorithm
  - Round robin, Priority, Multiple Priority Queues, Shortest First
  - Guaranteed scheduling, Lottery
  - Real time – Rate Monotonic, EDF, LL

- Two level scheduling
Input and Output

- Types of Devices – Block and Character

- Structure of I/O Software
  - Requirements – device independence, uniform naming, error handling travels upwards, synchronous and asynchronous, sharing of resources
  - 4 layers – Interrupt Handlers, Device Drivers, Device Independent OS and user Level Software

- Deadlock – what is it
  - Coffman conditions and modelling deadlock
  - Recovery / Avoidance algorithms – Ostrich, Find & Fix, Prevention, Avoidance using Bankers Algorithm

- Common I/O Devices - RAM Disks, HDD, Clocks and Timers, Terminals
Memory Management

- Different Types
  - Monoprogramming
  - Multiprogramming with fixed size partitions
  - Swapping
  - Virtual memory
    - Pages, Page Frames
    - VA – Page Table – PA
    - Reducing size of Page Table – Multilevel, Inverted
    - Quicker Translation – TLB – TLB Management
    - Page Replacement – NRU, FIFO, Second Chance, Clock Replacement, LRU
    - Design Issues – Working set and prepaging, Page Size
Memory Management

- Segmentation – why? – two part address – Segment number and Address within the segment
- Checkerboarding – memory compacting
- Segmentation + paging within the segment
  - Multics
  - Intel Pentium

- Keeping track of used and free memory
  - Bitmap
    - Required size for bitmap, finding a contiguous space for allocation
  - Linked list
    - Allocation – Best fit, first fit, next fit, worst fit, quick fit
File System

- Data storage container - Stored in external media also - Persistent data
  - File naming conventions
  - File structure – seq of bits, seq of records or tree of records
  - File types – dir, i/o files, ASCII files, exe
  - File Access – sequential, random
  - File Attributes – Protection, creator, modified, size etc

- Directories
  - Hierarchical System
  - Path Names
  - Dir Operations – C, D, O, C, R, REN, LINK, UNLINK
File System

- Implementation of FS – allocation of disk blocks to proper file
  - Contiguous space reservation – obviously not scalable
  - Linked List – Random access slow
  - Linked list with index (FAT like) – huge table size
  - I nodes - degrees of indirection – accessing a file / dir

- Block size
- Disk space management
- Keeping track of free blocks – linked list / bitmap
- Reliability – consistency checks for every block

- Performance – buffer caches, write through caches, LFS
File Systems

- Protection Mechanisms
  - Mutex
  - Domains – different domains have different privileges for different files
  - Access Control List
  - Capabilities List
Old Question Paper Solutions

- These are not official solutions.
- These are not necessarily THE CORRECT solutions.
Consider a processor with physical memory addressable using 32 bits and a 40-bit virtual memory address space.

The processor supports paging with a fixed page size of 64Kbytes, and protection mechanisms allowing for each page to have read, write and execute permissions set.

a) What is the minimum size (in bits) of a page table entry (PTE) for this system? Justify your answer, describing what information needs to be stored in PTE.

Page Size = 64 KB = 2^{16} words

So Page Offset is 16 Bits. These are the 16 LSB of the Physical and Virtual Address

So the MSB 16 bits of the PA should come from the PTE

So including say 2 bits for protection, 1 bit for Valid, minimum size of PTE = 19 bits – but usually wrapped to 32 bits – a word size
b) Let the size of a PTE be a constant “S”. What is the minimum size of the page table for an O/S process if a “flat” page table is used? What is the minimum size if a two level page table is used? (Assume 14 bits index the first level and 12 bits index the second level of the table).

- One PTE is S bits.
- The VA is 40 bits, out of which the 16 LSB are used as page offset.
- So the 40-16 = 24 MSB index the page table. So there will be $2^{24}$ pages, with one entry for each page in the page table.
- So the size of the page table is 16S Mb.
If it is two level –

First level page table is indexed by 14 bits. So the number of PTE is $2^{14}$. Each PTE points to a second level page table – say a 32 bit physical address. So the size of this page table is $2^{14} \times 32$ - 128 KB.

The 32 bit will point to the second level table. This table has $2^{12}$ entries, each of size S. So the size of this table is $2^{12} \times S$ – 4SKB.

Total size of page tables in memory at any given time is 128+4S KB.
2005 Question

c) Assume the time it takes for the processor to read a PTE from memory is 100ns, and the time to look up a translation look-aside buffer (TLB) entry is 1ns. What TLB hit rate is necessary to obtain an average virtual-to-physical address translation time of 2ns?

- Average time = Hit Rate * TLB Access time + Miss Rate * PTE access time
- $2\text{ns} = H \times 1\text{ns} + (1-H) \times 100\text{ns}$
- Solving for $H$, $H = \frac{98}{99}$
2005 Question

- Explain three advantages and one disadvantage of using virtual addresses compared to physical addresses.

- Disadvantage:
  - Translation time - slows down memory access

- Advantage:
  - Can load programs with memory requirements much larger than available physical memory
  - Can do time-shared multiprogramming
  - Can relocate programs without requiring any new compilation
  - Better memory utilization
This question considers a typical UNIX-like multi-user, multi-task operating systems which employs virtual memory, processes, and preemptive CPU schedulers to support time-sharing of resources among multiple tasks. In answering the five parts of this question, state your assumptions if necessary.

1) Describe two major constituents of the context of a running process.

- PC, Stack Pointer, Processor State etc
2) Consider a process “fork” event. Describe when and how the different constituents of the context of the parent process are copied to the child process

- Page Table, Pointers (Stack Pointer etc), PC and registers are copied
- Actual memory pages are not copied initially
- They are copied only when modified - COW
3) Describe one hardware event and one software event that can trigger the execution of the O/S kernel scheduler and a context switch.

- Hardware – DMA completes and sends completion notification. This triggers the scheduler to examine if the process which was waiting on this DMA needs to be scheduled now.

- Software – A process completes execution. This obviously will trigger the scheduler.
2006 Question

4) Why does an O/S kernel flush the contents of the translation look-aside buffer when a process context switch is performed?

- Context switch happens before a different process begins to execute
- The virtual address to physical address mapping for this process need not be the same as the previous process
- So the TLB is flushed so that the mappings of the old process is not used for the new process
2006 Question

5) Why is it typically faster for the O/S kernel to switch among threads of a process than it is to switch process contexts?

- Threads – lightweight processes
- Share address space – so no need to flush TLB, no need to set up new page table
- May share other context like stack these need not be copied