<Onsite interview Guide>

Technical Preparation tips:
The main areas software engineers should prepare to succeed at interview at Common Computer:

Algorithm Complexity: It's fairly critical that you understand big-O complexity analysis. Again run some practice problems to get this down in application.

Sorting: Know how to sort. Don't do bubble-sort. You should know the details of at least one n*log(n) sorting algorithm, preferably two (say, quicksort and merge sort). Merge sort can be highly useful in situations where quicksort is impractical, so take a look at it.

Hashtables: Arguably the single most important data structure known to mankind. You absolutely should know how they work. Be able to implement one using only arrays in your favorite language, in about the space of one interview.

Trees: Know about trees; basic tree construction, traversal and manipulation algorithms. Familiarize yourself with binary trees, n-ary trees, and trie-trees. Be familiar with at least one type of balanced binary tree, whether it's a red/black tree, a splay tree or an AVL tree, and know how it's implemented. Understand tree traversal algorithms: BFS and DFS, and know the difference between inorder, postorder and preorder.

Graphs: There are 3 basic ways to represent a graph in memory (objects and pointers, matrix, and adjacency list); familiarize yourself with each representation and its pros & cons. You should know the basic graph traversal algorithms:
breadth-first search and depth-first search. Know their computational complexity, their tradeoffs, and how to implement them in real code.
If you get a chance, try to study up on fancier algorithms, such as Dijkstra and A*.

Other data structures: You should study up on as many other data structures and algorithms as possible. You should especially know about the most famous classes of NP-complete problems, such as traveling salesman and the knapsack problem, and be able to recognize them when an interviewer asks you them in disguise. Find out what NP-complete means.

Mathematics: Some interviewers ask basic discrete math questions.
Spend some time before the interview refreshing your memory on (or teaching yourself) the essentials of combinatorics and probability. You should be familiar with n-choose-k problems and their ilk – the more the better.

Operating Systems: Know about processes, threads and concurrency issues. Know about locks and mutexes and semaphores and monitors and how they work. Know about deadlock and livelock and how to avoid them. Know what resources a processes needs, and a thread needs, and how context switching works, and how it's initiated by the operating system and underlying hardware. Know a little about scheduling. The world is rapidly moving towards multi-core, so know the fundamentals of "modern" concurrency constructs.

Coding: You should know at least one programming language really well, and it should preferably be C++ or Java. You will be expected to write some code in at least some of your interviews. You will be expected to know a fair amount of detail about your favorite programming language.

Sample Topics:

Coding
Sample topics: construct / traverse data structures, implement system routines, distill large data sets to single values, transform one data set to another.

Algorithm Design / Analysis
Sample topics: big-O analysis, sorting and hashing, handling obscenely large amounts of data, elegance and efficiency, decompose large problems and solve parts. Also see topics listed under 'Coding'.

System Design
Sample topics: features sets, interfaces, class hierarchies, designing a system under certain constraints, simplicity and robustness, tradeoffs.

Development Practices
Sample topics: validating designs, testing whiteboard code, preventing bugs, code maintainability and readability, refactor/review sample code.

Open-Ended Discussion / Role Related Knowledge
Sample topics: biggest challenges faced, best/worst designs seen, performance analysis and optimization, testing, ideas for improving existing products.