CS2010: Data Structures and Algorithms II

Topic 01: Sorting Algorithms

Ivana.Dusparic@scss.tcd.ie

Lecture Outline

- Introduce algorithm design techniques
- Review sorting algorithms
  - Insertion sort
- Learn some new ones
  - Bubble sort
  - Selection sort
  - Shellsort
  - Mergesort
  - Quicksort
- Analyse and classify by
  - Order of growth
  - Best, average, worst running time
  - Design approach
  - Stable vs unstable
- Textbook and lecture notes: Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne

Algorithm Design Approaches

Algorithm design

- Brute-force/exhaustive search
- Decrease and conquer
- Divide and conquer
- Transform and conquer
- Greedy
- Dynamic programming

Introduction to the Design and Analysis of Algorithms, Anany Levitin, 3rd edition, Pearson, 2012

Brute-force/exhaustive search

- Systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement
- Often simplest to implement but not very efficient
- Impractical for all but smallest instances of a problem
- Examples:
  - Selection sort
  - Bubble sort
  - In graphs – depth-first search (DFS), breadth-first search (BFS)

Decrease and conquer

- Establish relationship between a problem and a smaller instance of that problem
- Exploit that relationship top down or bottom up to solve the bigger problem
- Naturally implemented using recursion
- Examples
  - Insertion sort
  - In graphs – topological sorting

Divide and conquer

- Divide a problem into several subproblems of the same type, ideally of the same size
- Solve subproblems, typically recursively
- If needed, combine solutions
- Examples:
  - Mergesort
  - Quicksort
  - Binary tree traversal – preorder, inorder, postorder
    - Visit root, its left subtree, and its right subtree

Transform and conquer

- Modify a problem to be more amenable to solution, then solve
  - Transform to a simpler/more convenient instance of the same problem – *instance simplification*
  - Transform to a different representation of the same instance – *representation change*
  - Transform to an instance of a different problem for which an algorithm is available – *problem reduction*
- Examples:
  - Balanced search trees – AVL trees, 2-3 trees
  - Gaussian elimination – solving a system of linear equations

Dynamic programming

- Similar to divide and conquer, solves problems by combining the solutions to subproblems
  - In divide and conquer subproblems are disjoint
  - In dynamic programming, subproblems overlap, ie share subsubproblems
    - Solutions to those are stored, index and reused

- Examples:
  - A more efficient solution to Knapsack problem
  - Warshall's and Floyd's shortest path algorithms

Greedy

- Always make the choice that looks best at the moment
  - Does not always yield the most optimal solution, but often does
- Examples
  - Graphs:
    - Dijkstra – find the shortest path from the source to the vertex nearest to it, then second nearest etc
    - Prim
    - Kruskal
  - Strings
    - Huffman coding tree

Checkpoint – is the material clear enough?

Example: Binary Search

Goal. Given a sorted array and a key, find index of the key in the array?

Binary search. Compare key against middle entry.

- Too small, go left.
- Too big, go right.
- Equal, found.

successful search for 33

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑ lo														↑ hi

Binary Search is an example of what kind of algorithm:

- A – divide and conquer
- B – decrease and conquer
- C – brute-force
- D – greedy

https://responseware.turningtechnologies.eu/responseware/polling or use Turning Point App

Session id:

Fun fact: ex-bug in JDK implementation of binary search

› https://research.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html

  – Blog post by Joshua Bloch, Software Engineer who implemented binary search in java.util.Arrays

  – Undiscovered for 9 years

```
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1
         else if (midVal > key)
            high = mid - 1;
         else
            return mid; // key found
     }
     return -(low + 1);  // key not found.
 }
```