Problem Set 1
CSE 373 Fall 2015
September 2015

1 Degenerate sorting

Given an input array of length $n$, where each element has a value of 1, describe the run times of the sorting algorithms covered in class (selectionSort, mergeSort, quickSort), using the $O(n)$ notation.

2 Merge sort

Given an unsorted list of integers [15, 19, 0, 2, 1, 17, 100, 0, 12, 5, 18], show the list of intermediate arrays that will be produced during merge sort.

3 Binary search

A recursive binary search runs on an a sorted array $A$ of integers [1, 4, 6, 8, 9, 12, 14, 15, 19, 20]. We want to find $key=17$. List the sequence of elements of $A$ that will be compared with 17 during the search.

4 Recurrences

Solve the following recurrences. Assume $T(n) \leq c$ for some constant $c$ and for all $n \leq 10$.

- $T(n) \leq 4T(n/2) + n$
- $T(n) \leq 4T(n/2) + n \log n$
- $T(n) \leq 2T(n/4) + \sqrt{n}$
• \( T(n) \leq \sqrt{n}T(\sqrt{n}) + n \)
• \( T(n) \leq 7T(n/8) + n^{0.935784974} \)

5 Partition

The following array has been partitioned. Which elements could have been the pivot value?

\[ [31, 0, 25, 47, 53, 82, 79, 64, 98] \]

6 Duplicates

Write an algorithm that, given an array \( A \), outputs an array \( B \) that has the same set of elements as \( A \), but does not have any duplicate elements. \( B \) need not list the elements of \( A \) in the same order they occur in \( A \). What is the running time of your algorithm.

Now suppose you want \( B \) to have the same order as the first occurrence of each element in \( A \). Write an algorithm and give its running time.

7 2D searching

You are given a two-dimensional matrix \( A \) of size \( m \times n \). Each row of \( A \) is sorted. Each column of \( A \) is sorted. Analyze the running time of the following algorithms for searching for an element \( t \) in \( A \).

• Check each element of \( A \) to see if it is equal to \( x \).
• Perform binary search for \( x \) in each row of \( A \).
• Perform binary search for \( x \) in each column of \( A \).
• recursiveSearch1(\( A \), \( xlo \), \( xhi \), \( ylo \), \( yhi \), \( t \))
  - if \( xhi < xlo \) or \( yhi < ylo \), then return NOTFOUND
  - let \( xmid = \lfloor (xhi - xlo)/2 \rfloor \)
  - let \( ymid = \lfloor (yhi - ylo)/2 \rfloor \)
  - if \( A[xmid][ymid] == t \) then return \((xmid, ymid)\)
– else if $A[xmid][ymid] < t$
  * $a = \text{recursiveSearch1}(A, xmid + 1, xhi, ylo, ymid, t)$
  * $b = \text{recursiveSearch1}(A, xlo, xmid, ymid + 1, yhi, t)$
  * $c = \text{recursiveSearch1}(A, xmid + 1, xhi, ymid + 1, yhi, t)$
– else
  * $a = \text{recursiveSearch1}(A, xmid, xhi, ylo, ymid - 1, t)$
  * $b = \text{recursiveSearch1}(A, xlo, xmid - 1, ymid, yhi, t)$
  * $c = \text{recursiveSearch1}(A, xlo, xmid - 1, ylo, ymid - 1, t)$
– if any of $a$, $b$, or $c$ is not NOTFOUND, return it
– else return NOTFOUND

• Define and analyze an algorithm similar to the above, except that instead of picking the midpoint of $A$, it performs binary search on the diagonal of $A$ and recurses based on that.