Weekplan: Searching and Sorting

Philip Bille

Reading

Introduction to Algorithms, Cormen, Rivest, Leisersons and Stein (CLRS): Chapter 2.

Exercises

1 Run by Hand and Properties Solve the following exercises.

1.1 CLRS [w] 2.1-1.

1.2 CLRS [w] 2.1-2.

1.3 CLRS 2.2-3.

1.4 CLRS [w] 2.3-1.

1.5 CLRS [BSc] 2.3-4.

1.6 CLRS 2.3-6.

2 Duplicates and Close Neighbours Let $A[0..n-1]$ be an array of integers. Solve the following exercises.

2.1 [w] A *duplicate* in A is a pair of entries i and j such that $A[i] = A[j]$. Give an algorithm that determines if there is a duplicate in A in $\Theta(n^2)$ time.

2.2 Give an algorithm that determines if there is a duplicate in A in $\Theta(n \log n)$ time. *Hint:* use merge sort.

2.3 A *closest pair* in A is a pair of entries i and j such that $|A[i] - A[j]|$ is minimal among all the pairs of entries. Give an algorithm that finds a closest pair in A in $\Theta(n \log n)$ time.

3 [BEng†] **Implementation of Binary Search** Implement the binary search algorithm.

4 Implementation and Correctness of Merge Sort Solve the following exercises.

4.1 [†] Implement the merge algorithm.

4.2 [†] Implement the merge sort algorithm.

4.3 [BSc] Show that merge sort sorts all tables correctly. *Hint:* use induction.

5 2Sum and 3Sum Let $A[0..n-1]$ be an array of integers (positive and negative). The array A has a 2-*sum* if there exist two entries i and j such that $A[i] + A[j] = 0$. Similarly, A has a 3-*sum* if there exists three entries i, j and k such that $A[i] + A[j] + A[k] = 0$. Solve the following exercises.

5.1 [w] Give an algorithm that determines if A has a 2-sum in $\Theta(n^2)$ time.

5.2 Give an algorithm that determines if A has a 2-sum in $\Theta(n \log n)$ time. *Hint:* use binary search.

5.3 [w] Give an algorithm that determines if A has a 3-sum in $\Theta(n^3)$ time.

5.4 Give an algorithm that determines if A has a 3-sum in $\Theta(n^2 \log n)$ time. *Hint:* use binary search.

5.5 [$**$] Give an algorithm that determines if A has a 3-sum in $\Theta(n^2)$ time.

6 Selection, Partition, and Quick Sort Let $A[0..n-1]$ be an array of distinct integers. The integer with *rank* k in A is the kth largest integer among the integers in A. *The median* of A is the integer in A with rank $\lfloor (n-1)/2 \rfloor$. Solve the following exercises.

6.1 Give an algorithm that given a k finds the integer with rank k in A in $\Theta(n \log n)$ time.

A *partition* of A is a separation of A into two arrays A_{low} and A_{high} such that A_{low} contains all integers from A that are smaller than or equal to the median of A and A_{high} contains all the integers from A that are larger than the median of A. Assume in the following that you are given a linear time algorithm to determine the median of an array.

6.2 Give an algorithm to compute a partition of A in $\Theta(n)$ time.

6.3 [$*$] Give an algorithm to sort A in $\Theta(n \log n)$ time using recursive partition.

6.4 [$**$] Give an algorithm that given a k finds the integer with rank k in A in $\Theta(n)$ time.

M Mandatory Exercise: Smallest Missing Integer Let A be an array of length n such that each entry of A contains a unique integer from $\{1, 2, \ldots, 2n\}$, i.e., half of the integers from the set $\{1, 2, \ldots, 2n\}$ are present in A and the remaining numbers are missing. We interested in efficient algorithms to compute the *smallest missing integer* in A, that is, smallest integer from $\{1, 2, \ldots, 2n\}$, that does not appear in A. For instance given $A = [2, 7, 1, 8]$ ($n = 4$) the smallest missing integer is 3. Solve the following exercises.

M.1 Give an algorithm that solves the problem in $\Theta(n)$ time. *Hint:* use an extra array of length $2n$.

M.2 We now want to solve the problem fast, but also reduce the memory consumption as much as possible. Give an algorithm that solves the problem in $\Theta(n^2)$ time and only uses a constant number of extra variables (e.g. 42 `int` variables in Java).

M.3 Now consider the case where the integers in A are chosen from the set $\{1, 2, \ldots, n+1\}$ instead of $\{1, 2, \ldots, 2n\}$. Give an algorithm that solves the modified problem in $\Theta(n)$ time and only uses a constant number of extra variables.