COMP 3270

Book Notes: Ch.8

● comparison sorts: sorting algorithms in which the sorted order is determined based only on comparisons between the input elements

● any comparison sort must make Ω(n lg(n))

● 3 comparison sorts:

1. counting sort

2. radix sort

3. bucket sort

**8.1 Lower bounds of sorting**

● in this section we assume that all elements are distinct, so no a(sub i) = a(sub j) and we assume that all elements are compared with a(sub i) ≤ a(sub j)



**The decision-tree model**

● above is an EX of a decision-tree

● decision tree: a full binary tree that represents the comparisons between elements that are performed by a particular sorting algorithm operating on an input of a given size

● requirements of a decision-tree:

− the range is where n is the number of elements on the input sequence

− annotation of leaves is permutation as follows < >

− the left subtree holds the comparisons a(sub i) a(sub j), and the right subtree holds the comparisons a(sub i) > a(sub j)

− the established ordering is as follows

− because any correct sorting algorithm must be able to produce each permutation of its input, each of the n! permutations on n elements must appear as one of the leaves of the decisions tree for a comparison tree to be correct

− furthermore, each of the leaves must be reachable from the root by a downward path corresponding to an actual comparison sort

**A lower bound for the worst case**

● the length of the longest simple path form the root of a decision tree to any of its reachable leaves represents the worst-case number of comparisons that the corresponding sorting algorithm

● the worst-case number of comparisons for a given corresponding sort algorithm equals the height of its decision tree

● a lower bound on height of all decision trees in which each permutation appears as a reachable leaf is therefore the lower bound on the running time of any comparison sort algorithm; the bound is described in Theorem 8.1

**Theorem 8.1**

● Any comparison sort algorithm requires Ω(n lg(n)) comparisons in the worst-case

**Corollary 8.2**

● Heapsort and merge sort are asymptotically optimal comparison sorts

● Proof: Ω(n lg(n)) is the worst-case lower bound for heapsort and merge sort

**8.2 Counting Sort**

● counting sort: assumes that each of the n input elements is an integer in the range 0 to k, for some integer k; when k = O(n) the sort runs in θ(n) time

− it determines, for each input element x, the number or elements less than x, and it uses this info to directly place element x into its position in the array

− it requires the input array (in code A), an array to hold the sorted output (in code B), and an array to serve as a temporary storage (in code C)

− in code given, A’s index is A[1..n], B’s index is B[1…n], C’s index is C[0…k]

● Figure 8.2 (diagram of counting-sort)



− in the above k = 5, b/c each element of A is no larger than 5

● COUNTING-SORT (A, B, k)

let C[0…k] be a new array

for i = 0 to k

C[i] = 0

for j = 1 to A.length

C[A[j]] = C[A[j]] + 1

// C[i] now contains the number of elements equal to i

for i = 1 to k

C[i] = C[i] + C[i – 1]

// C[i] now contains the number of elements less than or equal to i

for j = A.length downto 1

B[C[A[j]]] = A[j]

C[A[j]] = C[A[j]] − 1

● running time:

T(n) = (1st for loop) + (2nd for loop) + (3rd for loop) + (4th for loop)

→ T(n) = θ(k + 1) + θ(n) + θ(k ) + θ(n)

→ T(n) = θ(k) + θ(n) + θ(k) + θ(n)

→ T(n) = θ(k + n) + θ(k + n)

→ T(n) = 2θ(k + n)

we usually use counting sort when k = O(n)

→ T(n) = θ(n)

● counting sort is stable

**8.3 Radix Sort**

● radix sort: is the algorithm used by the card-sorting machines found in computer museums

**Lemma 8.3**

**Lemma 8.4**

**8.4 Bucket Sort**