1 Lower Bounds for Sorting

- Insertion-sort, mergesort, heapsort, quicksort are all based on comparison. We sort by comparing elements in the array, and a sorting algorithm’s time complexity is the number of comparisons.

- Question: How many comparisons are needed in the worst case in order to sort \((a_1, a_2, \ldots, a_n)\)?

- Answer: \(\Omega(n \log n)\). The following is a proof.

- A comparison-based sorting algorithm can be represented as a binary tree, called decision tree or comparison tree, where each internal node indicates a comparison \(a_i : a_j\), and each leaf indicates an outcome.

- There are \(n!\) possible outcomes in sorting an array \((a_1, a_2, \ldots, a_n)\).

- The comparison tree of a sorting algorithm (for \(n\) elements) must be tall enough such that it has at least \(n!\) leaves.

- Let \(h\) be the height of the comparison tree. Then \# of leaves \(\leq 2^h\). (A binary tree of height \(h\) has at most \(2^h\) leaves.)

- So, we need \(2^h \geq n!\), or \(h \geq \log(n!) = \Omega(n \log n)\).

- That is, any comparison-based sorting algorithm needs \(\Omega(n \log n)\) time in the worst case.

- Question: Is it possible to sort an array using whatever mechanism in less than \(n \log n\) time?

- Yes, if the values to be sorted are in a small range.
2 Counting Sort

- Input: $0 \leq A[1..n] \leq k$.
- Output: $B[1..n]$
- Auxiliary: $C[0..k]$
- Running time: $\Theta(n + k)$.
- If $k = O(n)$, then the running time is $\Theta(n)$.
- Counting sort is stable.

procedure Counting-Sort($A[1..n], B[1..n], k$)
    for $i \leftarrow 0$ to $k$ do
        $C[i] \leftarrow 0$
    for $j \leftarrow 1$ to $n$ do
        $C[A[j]] \leftarrow C[A[j]] + 1$
        /* $C[i]$ = the number of elements equal to $i$. */
    for $i \leftarrow 1$ to $k$ do
        $C[i] \leftarrow C[i - 1] + C[i]$
        /* $C[i]$ = # of elements less than or equal to $i$. */
    for $j \leftarrow n$ downto $1$ do /* Place $A[j]$ in the right position. */
        $B[C[A[j]]] \leftarrow A[j]$
        reduce $C[A[j]]$ by 1
3 Radix sort

- Input: $A[1..n]$; each key is a $d$-digit integer, with digit number 1 being the least significant.

- Algorithm:

  ```plaintext
  procedure Radix-Sort($A[1..n]$, $d$)
  for $i ← 1$ to $d$ do
    use a stable sort (e.g. Counting Sort) to sort array $A[1..n]$ on digit $i$
  ```

- Running time: $O(dn) = O(n)$ if $d$ is a constant.
4 Sorting $n$ integers in the range $0$ to $n^2$

- Input: $A[1..n]$; each key is an integer in the range $0$ to $n^2$.
- Assume that each integer is represented as a binary. Then, each key has $2 \log n$ bits (more precisely, $\lfloor 2 \log n \rfloor + 1$ bits.)
- What is the running time if we sort the array using counting sort?
- What is the running time if we sort the array using radix sort?
- How to sort the array in $O(n)$ time?