CHAPTER 4

Sequential Sorting Algorithms and Their Analysis
Purposes

- introducing some well known algorithms
- illustrating various techniques and features relating to the design and complexity analysis of algorithms
Shell Sort

- As mentioned in CMPE250, Insertion Sort is an order-optimal adjacent-key sorting algorithm.
- ShellSort (Named after Donald Shell) is a comparison based but a non adjacent key sorting algorithm.
- ShellSort aims to reduce the work done by insertion sort (i.e. scanning a list and inserting into the right position).
- ShellSort is faster than $O(n^2)$.
The choice of GAP

- Done by sorting subarrays of equally spaced indices
- This space is called the GAP.
- Choosing the gap sizes as prime numbers is efficient (to prevent sorting the same number again and again)
- Choosing odd numbers as a gap size is also appropriate
Shell Sort Illustration

GAP=3

45 9 6 4 24 8 7 45
23 6 4 24 8 7 23
4 4 8 23 9 23 4
9 6 7 45 23 12 4
6 8 7 24 91 12 6
7 9 23 24 91 45 7
91 9 12 23 45 91 91

GAP=2

4 6 6 8 7 7 8
4 6 8 7 7 8 8
4 6 8 7 7 8 8
4 6 8 7 7 8 8

GAP=1

4
4
4
4
4
4
4
Analysis of Shell Sort: GAP Size

- $O(n^{1.5})$ when the gap size is $2^k - 1$ (Hibbard)
- $O(n^{1.33})$ when the gap size follows $9 \times 4^i - 9 \times 2^i + 1$ (Sedgewick)
- Using the increments of the form $2^i \times 3^j$ it is $\Theta(n \log^2 n)$ (Pratt)
- A well known sequence is 1, 4, 10, 23, 57, 132, 301, 701, 1750, .. (Ciura)
- Empiric sequence with Fibonacci numbers (leaving out one of the starting 1's) to the power of two times the golden ratio, which gives the following sequence: 1, 9, 34, 182, 836, 4025, 19001, 90358, 428481, 2034035, 9651787, 45806244, 217378076,...
Merge Sort

- Divide an array into halves
  - Sort the two halves
  - Merge them into one sorted array

Referred to as a divide and conquer algorithm

- This is often part of a recursive algorithm
- However recursion is not a requirement
Merging two sorted lists into a sorted list

First array
3 5 7 9
3 > 0, so copy 0 to new
3 5 7 9
3 > 2, so copy 2 to new
3 5 7 9
3 < 4, so copy 3 to new
3 5 7 9
5 > 4, so copy 4 to new
3 5 7 9
5 < 6, so copy 5 to new
3 5 7 9
7 > 6, so copy 6 to new
3 5 7 9
The entire second array has been copied to the new array
Copy the rest of the first array to the new array

Second array
0 2 4 6
0 2 4 6
0 2 4 6
0 2 4 6
0 2 4 6
0 2 4 6
0 2 4 6
0 2 4 6
0 2 4 6
Algorithm for MergeSort

Algorithm mergeSort(a, first, last)
// Sorts the array elements a[first] through a[last] recursively.
if (first < last)
{
    mid = (first + last)/2
    mergeSort(a, first, mid)
    mergeSort(a, mid+1, last)
    Merge the sorted halves a[first..mid] and a[mid+1..last]
}
Recursive Calls and Merges
Analysis of Merge Sort

- Efficiency of the merge sort
  - Merge sort is $O(n \log n)$ in all cases
  - Its need for a temporary array is a disadvantage
Quick Sort

Divides the array into two pieces
- Not necessarily halves of the array
- An element of the array is selected as the pivot

Elements are rearranged so that:
- The pivot is in its final position in sorted array
- Elements in positions before pivot are less than the pivot
- Elements after the pivot are greater than the pivot
Algorithm for Quick Sort

**Algorithm quickSort(a, first, last)**

// Sorts the array elements a[first] through a[last] recursively.

if (first < last)
{
    Choose a pivot
    Partition the array about the pivot
    pivotIndex = index of pivot
    quickSort(a, first, pivotIndex-1) // sort Smaller
    quickSort(a, pivotIndex+1, last) // sort Larger
}
Partitioning in Quick Sort

\[ \leq \text{pivot} \quad | \quad \text{pivot} \quad | \quad \geq \text{pivot} \]

Smaller

Larger
A Partitioning Strategy

continued
End of Partitioning
Quick sort rearranges the elements in an array during partitioning process.

After each step in the process:
- One element (the pivot) is placed in its correct sorted position.

The elements in each of the two subarrays:
- Remain in their respective subarrays.
Analysis of Quick Sort

- Quick sort is $O(n \log n)$ in the average case
- $O(n^2)$ in the worst case
- Worst case can be avoided by careful choice of the pivot
Tree Sort

- Create empty Binary Search Tree
- Insert (ordered) each element into BST
- Inorder traverse BST
- (destroy BST)
Analysis of Tree Sort

Although the worst case for creating a binary search tree is $\Theta(n^2)$, the average case is $\Theta(n \log n)$.
Overview of Tree Sort

Advantages:
- n elements, \( \log(n) \) insert \( \Rightarrow \) \( O(n\log(n)) \) sort
- don’t need to have a fixed set of data, nodes can be inserted and deleted dynamically

Disadvantages:
- additional overhead of entire Tree
- if data arrives in order or reverse order degenerate to \( O(n^2) \) behavior just like QuickSort