Solving Recurrence Relations

Page 62, [CLR]

Recurrence; Cond	Solution
$T(n) = T(n-1) + O(1)$	$T(n) = O(n)$
$T(n) = T(n-1) + O(n)$	$T(n) = O(n^2)$
$T(n) = T(n-c) + O(1)$	$T(n) = O(n)$
$T(n) = T(n-c) + O(n)$	$T(n) = O(n^2)$
$T(n) = 2T(n/2) + O(n)$	$T(n) = O(n \log n)$
$T(n) = aT(n/b) + O(n);$ $a = b$	$T(n) = O(n \log n)$
$T(n) = aT(n/b) + O(n);$ $a < b$	$T(n) = O(n)$
$T(n) = aT(n/b) + f(n);$ $f(n) = O(n^{\log_b a - \epsilon})$	$T(n) = O(n)$
$T(n) = aT(n/b) + f(n);$ $f(n) = O(n^{\log_b a})$	$T(n) = \Theta(n^{\log_b a} \log n)$
$T(n) = aT(n/b) + f(n);$ $f(n) = \Theta(f(n))$ $af(n/b) \leq cf(n)$	$T(n) = \Omega(n^{\log_b a} \log n)$

Sorting

- Input is a list of n items that can be compared.
- Output is an ordered list of those n items.
- Fundamental problem that has received a lot of attention over the years.
- Used in many applications.
- Scores of different algorithms exist.
- Task: To compare algorithms
  - On what bases?
    - Time
    - Space
    - Other

Sorting Algorithms

- Selection Sort
- Insertion Sort
- Bubble Sort
- Shaker Sort
- Shell Sort
- Merge Sort
- Heap Sort
- Quick Sort

- Bucket & Radix Sort
- Counting Sort

Selection Sort

Array Position		0	1	2	3	4	5
Initial State		8	5	9	2	6	3
After Iteration 1		2	5	9	8	6	3
After Iteration 2		2	3	9	8	6	5
After Iteration 3		2	3	5	8	6	9
After Iteration 4		2	3	5	6	8	9
After Iteration 5		2	3	5	6	8	9

Selection Sort

```
 algorithm selectionSort( array a, integer N)
// given array a[0..N-1]
   {
      for( int p = 0; p < N; p++ )
      {
         Compute j, the index of the smallest item in a[p..N];
         Swap a[p] and a[j];
      }
   }
```


Selection Sort

```
 algorithm selectionSort( array a, integer N)
// given array a[0..N-1]
   {
      for( int p = 0; p < N-1; p++ )
      {  // Compute j, the index of the smallest item in a[p..N];
        j = p;
        for (int m = p+1; p < N; p++)
                if (a[m] < a[j]) then j = m;
          // Swap a[p] and a[j];
        temp = a[p];      a[p] = a[j];    a[j] =temp;
      }
   }
```

Figure 8.3
Basic action of insertion sort (the shaded part is sorted)

Array Position	0	1	2	3	4	5
Initial State	8	5	9	2	6	3
After a[0..1] is sorted	5	8	9	2	6	3
After a[0..2] is sorted	5	8	9	2	6	3
After a[0..3] is sorted	2	5	8	9	6	3
After a[0..4] is sorted	2	5	6	8	9	3
After a[0..5] is sorted	2	3	5	6	8	9

Figure 8.4
A closer look at the action of insertion sort (the dark shading indicates the sorted area; the light shading is where the new element was placed).

Array Position	0	1	2	3	4	5
Initial State	8	**5**				
After a[0..1] is sorted	5	8	**9**			
After a[0..2] is sorted	5	8	9	**2**		
After a[0..3] is sorted	2	5	8	9	**6**	
After a[0..4] is sorted	2	5	6	8	9	**3**
After a[0..5] is sorted	2	3	5	6	8	9

Insertion Sort

```
 algorithm insertionSort( array a, integer N)
// given array a[0..N-1]
   {
      for( int p = 1; p < N; p++ )
      {  // insert a[p] in its right location
          temp = a[p];
          int j = p;
        while (j > 0 && temp < a[j-1])
                 a[j] = a[j-1];
                 j = j-1;
          a[j] = temp;
      }
   }
```