Module 6

Recursion, Stack & Queue Applications

Types of Algorithm

- Iterative
  - Function with a loop
  - Uses frequency count\Incremental approach to analyze time function
- **Recursive**
  - **Function calling itself**
  - **Uses recursive equations to analyze time function**
- Not recursive or iterative
  - No dependency of running time on input size
  - Time will be constant

Recurrence Relation

- An equation that defines function in terms of its values on smaller input
- An equation which is defined in terms of itself (breaking into smaller inputs)

- Examples

```
A(n)
  if condition
      return A(n/2) + A(n/2)
```

Time to analyze the algorithm: $T(n) = c + 2T(n/2)$

```
A(n)
  if n > 1
      return A(n-1)
```

Time to analyze the algorithm: $T(n) = c + T(n\text{-}1)$

Linear Search

```
LINEAR-SEARCH(A, n, key)        // Array: A, Array Size: n
if n == 0
        return null
if A[n] == key
        return n;
return LINEAR-SEARCH(A, n-1, key)
```

$$T\,(n) = c + T\,(n-1)$$

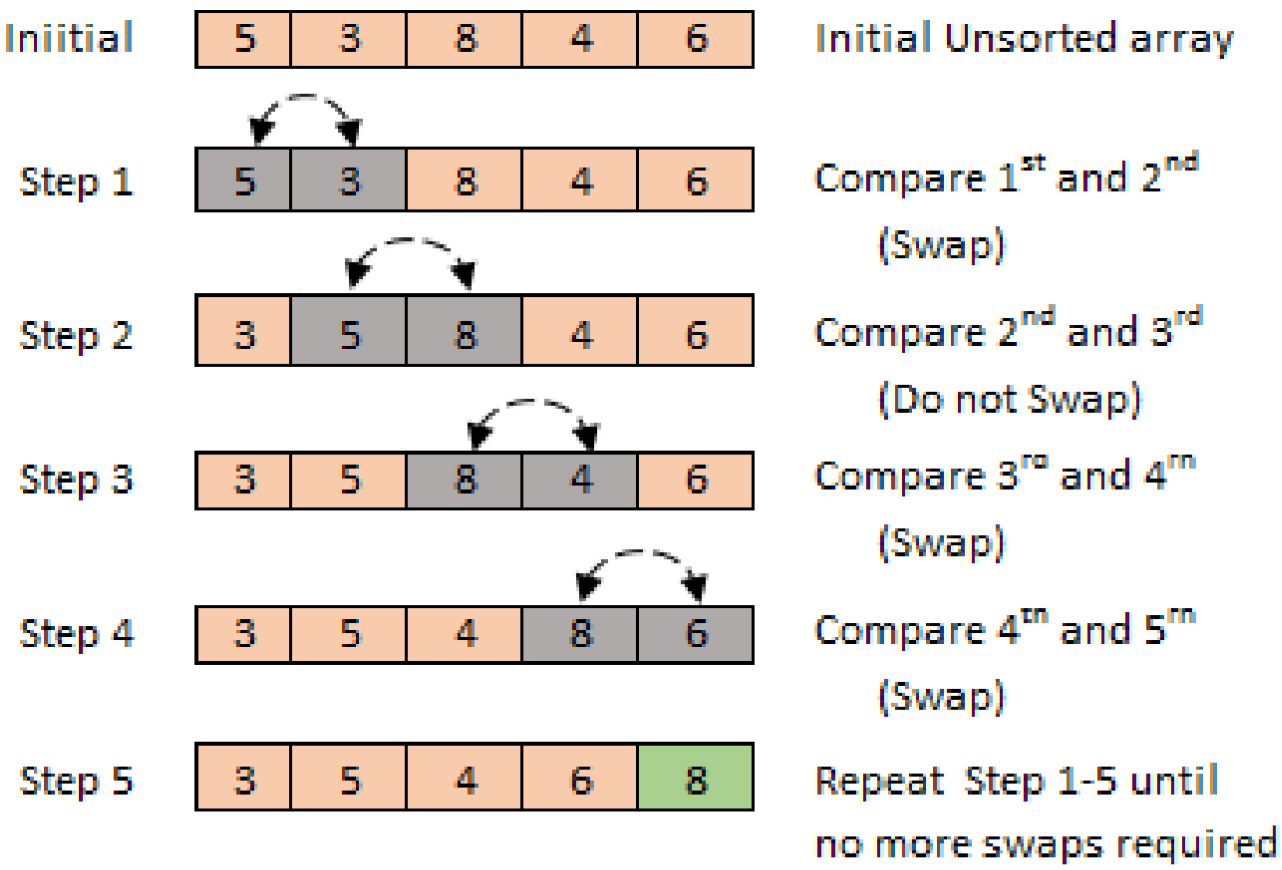
Binary Search

```
BINARY-SEARCH(A, start, end, key)
if start>end
        return null
mid = (start + end) / 2
if A[mid] == key
        return mid
if key < A[mid]
        return BINARY-SEARCH(A, start, mid-1, key)
else
        return BINARY-SEARCH(A, mid+1, end, key)
```

$$T(n) = c + T(n/2)$$

Bubble Sort

```
BUBBLE-SORT(A, n)
if n == 1
      return
for i=1 to n-1
       if A[i] > A[i+1]
              temp = A[i]
              A[i] = A[i+1]
              A[i+1] = temp
BUBBLE-SORT(A, n-1)
```



$$\boldsymbol{T(n) = (n-1) + T(n-1)}$$

Solving Recurrence

- **Substitution Method**
  - Guess a bound (or the behavior of the function)
  - Use mathematical induction method to prove the guess correct

- **Recursion Tree Method**
  - Convert the recurrence equation into a tree where nodes represent the cost incurred at various levels of recursion
  - Summation of all the costs (till last level) to solve the recurrence

- **Master's Theorem**
  - Provides a cook-book or bounds to solve recurrence of the following form: $T(n) = aT(n/b) + f(n)$

Substitution Method

- Substitute the guessed answer when mathematical induction hypothesis is applied to smaller values

- Powerful method but slow

- It can be applied only when it is easy to guess the form of the solution
  - Unfortunately, there is no general way to guess the correct form/solution
  - It takes experience, practice, and creativity

Example: Substitution Method

```
ALGO(n)
1 if n>0
2       Do something
3       ALGO(n-1)
```

- Time taken by the algorithm *A*

Linear Search

$$T(n) = \begin{cases} 1 + T(n - 1) & if \ \ n \geq 1 \\ 1 & if \ \ n = 0 \end{cases}$$

Solving $T(n) = T(n - 1) + 1$

$$T(n) = T(n-1) + 1 \quad (1)$$

- Divide the task further.

$$T(n-1) = T(n-2) + 1 \quad (2)$$
$$T(n-2) = T(n-3) + 1 \quad (3)$$

- Substitute Eq 2 in Eq 1

$$T(n) = T(n-2) + 2 \quad (4)$$

- Substitute Eq 3 in Eq 4

$$T(n) = T(n-3) + 3 \quad (5)$$

Solving $T(n) = T(n-1) + 1$

- After k iterations

$$T(n) = T(n-k) + k \tag{6}$$

- Termination/Stability condition $(n - k) = 0 \Rightarrow n = k$

$$T(n) = 1 + n \tag{7}$$

$$T(n) = O(n)$$

Exercises: Substitution Method

$$T(n) = \begin{cases} T(n-1) + n & if\ n > 1 \\ 1 & if\ n = 1 \end{cases}$$

Bubble Sort

$$\boldsymbol{T(n) = O(n^2)}$$

$$T(n) = \begin{cases} 2T(n/2) + c & if\ n > 1 \\ 1 & if\ n = 1 \end{cases}$$

$$\boldsymbol{T(n) = O(n)}$$

Exercises: Substitution Method

$$T(n) = \begin{cases} 2T(n/2) + n & if\ n > 1 \\ 1 & if\ n = 1 \end{cases}$$

$$\boldsymbol{T(n) = O(n \log n)}$$

$$T(n) = \begin{cases} T(n/2) + 1 & if\ n > 1 \\ 1 & if\ n = 1 \end{cases}$$

Binary Search

$$\boldsymbol{T(n) = O(\log n)}$$