Advanced Algorithms Final Notes

Recurrences

Master Method

$$T(n) = aT(\frac{n}{b}) + f(n)$$

$$f(n) = \begin{cases} O(n^{log_b^a}) & \text{if } f(n) = O(n^{log_b^a - \epsilon}) \\ O(f(n)logn) & \text{if } f(n) = O(n^{log_b^a}) \\ O(f(n)) & \text{if } f(n) = O(n^{log_b^a + \epsilon}) \end{cases}$$

Sum of Sequences

$$\frac{n(a_1 + a_{50})}{2}$$

Order Notation

$$O = \text{Upper Bound} \quad \Theta = \text{Tight Bound} \quad \Omega = \text{Lower bound}$$

Binary Search

1. Sorted Sequence
2. Check if middle value is the value you want
3. If not, rerun alogrithm on the top or bottom partition based on the number you wants value
4. if low $\leq$ hi then the value is not found

Power function

```
power(X,n)
if n == 0
    return 1
else if n == 1
    return X
else
    S = power(x, n/2)
    if n is odd
        return S*S*X
```

```
    if n is even
        return S * S
```

Merge sort

```
Merge(A,B,P,q,r)
//Precondition: A[p...q], A[q+1...r)] are sorted
//B is for temp work
Copy A[p...r] into B[p...r]
i = p
j = q+1
for k=p to r
    if j > r or (i <= q and B[i] <= B[j])
        A[k] = B[i++]
    else
        A[k] = B[j++]
```

Mergesort use merge after spliting each side into two parts, then run mergesort of them

Rod-Cutting problem

Input: n, P[1...n]
Output: max revenue from rod of length n

Recurrence

$$r(n) = \begin{cases} max(p_i + r(n-i)) & \text{if } n > 0 \\ 0 & \text{if } n = 0 \end{cases}$$

Dynamic program

```
cutRod(Price[], int n)
    price[0] = 0
    for i = 1 to n
        for j = 0 to i
            max_val = max(max_val, price[j] + val[i-j-1])
    val[i] = max_val
return val[n]
```

Longest Common Subsequence

Input: X[1...m] Y[1...n]

Output: Z[1...k] that is a subsequence of X and Y

Recurrence

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ 1 + LCS(i-1, j-1) & \text{if } x[i] = y[i] \\ max(LCS(i-1,j), LCS(i, j-1)) & \text{if } x[i] \neq y[i] \end{cases}$$

Dynamic program

```
for i = 0 to m
    L[i,0] = 0
for j = 0 to n
    L[0,j] = 0
for i = 1 to m
    for j = 1 to n
        if x[i] = y[j]
            L[i,j] = 1 + L[i-1,j-1]
            D[i,j] = 1
        else if L[i,j-1] > L[i-1,j]
            L[i,j] = L[i-1,j]
            D[i,j] = 2
        else
            L[i,j] = L[i,j-1]
            D[i,j] = 3
```

Reconstructing

```
\\Input: x,y,L,D
\\Output: Z[1...k]
K = L[m,n]
i = m
j = n
while k > 0
    if D[i,j] = 1
        Z[k] = x[i]
        k--
        i--
        j--
    else if D[i,j] = 2
        i--
    else
        j--
return Z
```

Activity Selection

Input: StartTimes s, FinishTimes f, Values v
Output: Find a compatable subset Q

Recurrence

$$ASP(i) = \begin{cases} 0 & \text{if } i = 0 \\ max(ASP(i-1), v_i + ASP(j)) & \text{if } i > 0 \end{cases}$$

Dynamic program

```
A[0] = 0
for i = 1 to n
    j = i-1
    while f[j] > s[i]
        j--
        A[i] = max(A[i-1], v[i] + A[j])
return A
```

Adjacency list representation

Each vertex has a list of its neighbors (In a directed graph, outbound neighbors only)

Breadth First Search (BFS)

```
//s is the source vertex
BFS{G,s)
    for each u in V
        u.color = white
        u.d = infinity
        u.pi = null

    s.d = 0
    s.color = grey
    Create a queue<vertex> Q contatining s
    while Q is not empty
        u = Q.remove()
        for each edge e =(u,v) in Adj[u]
            if v.color == white
                v.pi = u
                v.d = u.d + 1
                v.color = grey
```

```
        Q.add(v)
    u.color = black
```

Depth First Search (DFS)

```
DFS(G)
    for u in V
        u.color = white
        u.pi = null
    time = 0
    topNum = |v|
    for u in V
        if u.color == white
            DFS_Visit(u)

DFS_Visit(u)
    u.color = grey
    u.dis = ++time
    for each edge e=(u,v) in Adj[u]
        if v.color == white
            v.pi = u
            DFS_Visit(v)
    u.color = black
    u.fin = ++time
    u.top = topNum--
```