Algo S 2023 Crib Sheet Midterm Hayden Fuller

Notes:

Time Complexity:

O Θ Ω , $f(n)/g(n) = c \to f(n) \in \Theta(g(n))$, $= 0 \to f \in O(g(n))$, $= \infty \to f \in \Omega(g(n))$

$T(n) = 2^n \to T(n) = 1 + \Sigma_{i=0}^{n-1} T(i)$

bit complexity, $k = \log_2 N$, $N = 2^k$

Modular Arithmetic: $x \mod N = 0 \to x^y \mod N = 0$; GCD(a+b,b)=GCD(a,b) ; Fermat $a^n \mod n = a$, $a^{n-1} \mod n = 1$
$a^m \mod n = a^{m \bmod (n-1)} \mod n$; $a \equiv b \mod n \to a^k \equiv b^k \mod n$ ex: $3^{201} \mod 11 = 3^{201 \bmod 10} \mod 11 = 3^1 \mod 11 = 3$
$53^{1069} \mod 54 = (-1)^{1069} \mod 54 = -1 \mod 54 = 53$

Euclid's method:

```
def Euclid: if b=0: return a ; return Euclid(b,a mod b)
def extend: if b=0: return 1,0,a ; x,y,d=extend(b,a mod b) ; return y, (x-((floor(a/b))*y)), d #ax+by=d=GCD(a,b)
a/b   r=a mod b    x  y  d
25/11 3=25-2(11)   4 -9  1
11/3  2=11-3(3)   -1  4  1
3/2   1=3-1(2)     1 -1  1
2/1   0            0  1  1
1/0                1  0  1
GCD(25,11)=1=4(25)-9(11)=100-99=1
```

Recursion Master's Theorem:

master's theorem: $T(n) = aT(n/b) + O(n^d)$ for $a > 0, b > 1, d \geq 0$, n=nodes, divided by b each step of the tree
$d > \log_b a, T(n) = O(n^d)$, $d = \log_b a, T(n) = O(n^d \log n$, $d < \log_b a, T(n) = O(n^{\log_b a})$

Finding a particular element in a list:

Merge sort: O(n log n), T(n)=2T(n/2)+O(n)

```
def mergesort(S):
  if len(S) < 2: return S
  return merge(mergesort(S[:(len(S)//2)]), mergesort(S[(len(S)//2):]))
def merge(x,y):
  if len(x)==0: return y
  if len(y)==0: return x
  if x[1] <- y[1]: return x[1]+merge(x[2:],y)
  return y[1]+merge(x,y[2:])
```

Quick sort:

```
def quickSort(S,low,high):
    if low < high:
        r=random.randint(low,high) #this really doesn't need to be done randomly
        S[low],S[r]=S[r],S[low] #just uses the low value, but swaps with a random
        v=low
        i=low+1
        for j in range(low+1,high+1): #for everything in the current area
            if S[j] <= S[v]: #if current index is less than pivor
                S[i],S[j]=S[j],S[i] #swap it with pivot
                i+=1
        S[v],S[i-1]=S[i-1],S[v] #lmao idk, just accounting for things
        v=i-1
        quickSort(S,low,v-1) #continue sorting left side
        quickSort(S,v+1,high) #continue sorting right side
    return S #unnecessary, just return
ans = quickSort(S,0,n-1) #not necessary, just call quickSort(S,0,n-1) #driver
```

k'th element (just run with floor(k/2) for median): O(n^2), unlikely

```
def kth(S,k):
    v = S[random.randint(0,len(S)-1)] ; sl=[] ; sv=[] ; sr=[] ;
    for i in S:
        if i<v: sl.append(i)
        elif i>v: sr.append(i)
        else: sv.append(i)
    if k<len(sl): return selection(sl,k)
    elif k>len(sl)+len(sv): return selection(sr,k-len(sl)-len(sv))
    else: return v;
```

Average Case Analysis

Graph Theory DFS: O(V+E)

```
def dfs(g):
    v=[] ; pre=[None for i in range(len(g))] ; post = pre ; clk = 0
    for i in range(len(g)):     #visit everything not yet visited
        if i not in v:  v,clk,pre,post = dfss(g,v,i,clk,pre,post)
def dfss(g,v,i,clk,pre,post): #graph, visited, index, clock
    v.append(i) ; pre[i]=clk ; clk+=1
    for j in g[i]: #visit all connected nodes
        if j not in v: #only if not yet visited that haven't been visited
            v,clk,pre,post = dfss(g,v,j,clk,pre,post) #continue dfss on them
    post[i]=clk ; clk+=1
    return v,clk,pre,post
```

BFS: O(V+E)

```
def bfs(g,s): #graph, start
  dist = [infinity for i in range(len(g))] ; dist[s]=0
  Q=[s]
  while Q is not empty: #for evey node at this depth
    u=eject(Q)
    for v in g[u]: #for every node connected to this
      if dist[v]=infinity: #if unvisited
        inject(Q,v)
        dist[v]=dist[u]+1
```

Dijktra's

```
def bfs(g,l,s): #graph, lengths, start
  prev = [None for i in range(len(g))] ; dist = [infinity for i in range(len(g))] ; dist[s]=0
  H=makequeue(range(len(g))) #using dist values as keys
  while H is not empty: #for evey node at this depth
    u=deletemin(H)
    for v in g[u]: #for every node connected to this
      if dist[v]>dist[u]+l[u][v]: dist(v)=dist[u]+l[u][v] ; prev[v] = u ; decreasekey(H,v)
```