comparison
sort $\Omega(n\log n)$

Euclid Alg

$gcd(m,n) = gcd(n, m \bmod n)$ until $n=0$

BST starts with middle num

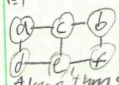
TSP

Find all combinations, calculate each

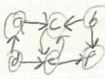
Graph Def

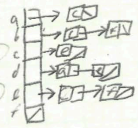
- A graph $G = \langle V, E \rangle$ = Finite, non-empty set V
  set E (sets cannot have dupes)

$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, $0 \le |E| \le |V|(n-1)/2$



$$\begin{array}{c|c} & \begin{matrix} a & b & c & d & e & f \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \end{matrix} & \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$





Algorithms

$1 > \log n > n > n\log n > n^2 > n^3 > 2^n > n!$

Big Oh - Set of all functions with lower/same order of growth $(n \in O(n^2))$

Big omega - Set of all func with higher/same $(n^3 \in \Omega(n^2))$

Big theta - Set of all with same $(n^3 \notin \Theta(n^2)$ & $n \notin \Theta(n^2))$
$(2n^2 \in \Theta(n^2))$

Recursion

M(n): Number of multiplications executed for the input number n.

Algorithm F(n)
  if n=0
    return 1
  else
    return F(n-1)·n

$M(0) = 0$ (initial)
$M(n) = M(n-1)+1$ (recur. relation)

$M(n) = M(n-1)+1$
$= [M(n-2)+1]+1$
$M(n-2)+2$
$M(n) = M(n-2)+2$
$[M(n-3)+1]+2$
$M(n-3)+3$
$= M(n-i)+i \quad i \in \mathbb{Z}$
$= \dots$
$= M(0)+n$
$= 0+n$
$= n \in \Theta(n)$

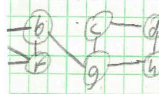
M(n) is the amount of multiplications for each n.

if n=1
  return (2 mults)
else
  return S(n-1) + n·n·n

$M(n-1)+2$
$= M(n-i)+2i \quad (i=n-1)$
$= M(1)+2\cdot(n-1)$
$= 0+2(n-1)$
$\in \Theta(n)$

DFS Uses Stack

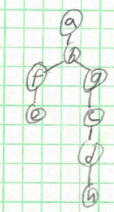
Mark array to track visited vertices



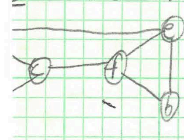
Mark Array

a	1
b	2
c	6
d	7
e	4
f	3
g	5
h	8

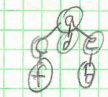
DFS Tree

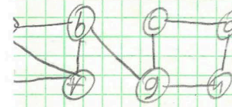


BFS Uses Queue $\Theta(|V|^2)$

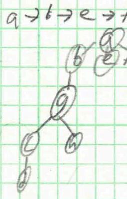


$a \to c \to d \to e \to f \to b$





$a \to b \to e \to f \to g \to c \to h \to d$



Brute Force

Selection Sort

```
int i,j
int aLength
for (i=0; i<aLength-1; i++) {
  int jMin = i;
  for (j = i+1; j<aLength; j++) {
    if (a[j] < a[jMin]) {
      jMin = j;
    }
  }
  if (jMin != i) {
    swap(a[i], a[jMin])
  }
}
```

Bubble Sort

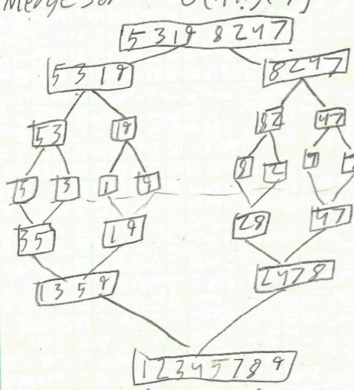
```
func bubbleSort(A[])
  n = length(A)
  while(swapped == false)
    for (i = 1 to n-1)
      if A[i-1] > A[i]
        swap(A[i-1], A[i])
        swapped = true
    n = n-1
```

OR

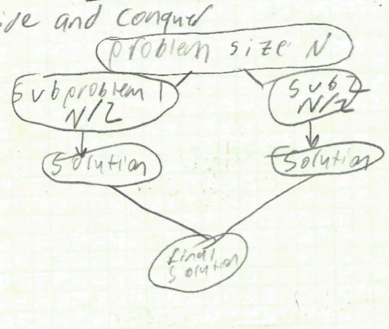
```
for i=0 to n-2 do
  for j=0 to n-2-i do
    if A[j+1] < A[j]
      swap(A[j], A[j+1])
```

Text NOBODY
NOT B≠T
NOT O≠N
NOT B≠N

Merge Sort - $O(n\log(n))$



Divide and Conquer



Master The orem

b = number of subproblems
a = number of subproblems to be solved
d = is the number in $\Theta(n^d)$ in which $\Theta(n^d)$ is the time to divide & conquer the original problem of size n

```
int sum(A[], start, end)
  if (start == end)
    return A[start]
  else
    int sum1 = sum(A, start, (start+end)/2)
    sum2 = sum(A), (start+end)/2, end)
    return(sum1 + sum2)
```

b = 2
a = 2
d = 0

Case 1 if $a < b^d$ then $\Theta(n^d)$
Case 2 if $a = b^d$ then $\Theta(n^d \cdot \log n)$
Case 3 if $a > b^d$ then $\Theta(n^{\log_b a})$

$T(n) = 2\cdot T(n/2) + 5n^3 + 7n$

a = 2
b = 2
d = 3 $(5n^3)$

$2 < 2^3$, therefore $\Theta(n^3)$