Recorrencia

- (Polinômio Característico)
Se a equação é da forma
$a_0 T(n) + a_1 T(n-1) + \dots + a_k T(n-k) = 0$
com o polinomio caracteristico:
$a_0 x^k + a_1 x^{k-1} + \dots + a_k x^0$
onde a resolução é da forma
$T(n) = c_1 \cdot (r_1)^n + \dots + c_k (r_k)^n$
Com $r_1, \dots, r_k$ raizes do polinomio caracteristico

Note: Se $r_1 = r_2 \neq r_3$ (raizes iguais)
$T(n) = c_1 (n^0)(r_1)^n + c_2 (n^1)(r_2)^n + c_3 (r_3)^n$

- (Substituição)
Se $a_0 T(n) + \dots + a_k T(n-k) = f(n)$
então podemos fazer uma substituição caindo no caso de Polinomio Caracteristico.

- (Árvore de Recursão)
Da forma $T(n) = k \cdot T(f(n)) + g(n)$
Trata-se de construir uma tree apartir do $g(n)$, onde a soma dos nos é $T(n)$ e cada no se divide em k outros.
Assim podemos definir uma função soma por linha.

- (Master Method)
Da forma $T(n) = a\, T(n/b) + f(n)$
$a \geq 1$
$b > 1$ $\quad f(n) = O(n^k \log^p n)$
Case 1: $\log_b a > k \Rightarrow O(n^{\log_b a})$
Case 2: $\log_b a = k \Rightarrow$
$p > -1 \Rightarrow O(n^k \log^{p+1} n)$
$p = -1 \Rightarrow O(n^k \log\log n)$
$p < -1 \Rightarrow O(n^k)$
Case 3: $\log_b a < k \Rightarrow$
$p \geq 0 \Rightarrow O(n^k \log^p n)$
$p < 0 \Rightarrow O(n^k)$

Slide 1 (Algorithmic Analysis, Divide and Conquer)

- (big-O notation)
Se $f(n) = O(g(n))$
$\exists n_0, c$ t.q. $\forall n > n_0$ vale que
$f(n) \leq c \cdot g(n)$

- (big-Ω Notation)
Se $f(n) = \Omega(g(n))$
$\exists n_0, c$ t.q. $\forall n > n_0$ vale que
$f(n) \geq c \cdot g(n)$

- (big-Θ Notation)
Se $f(n) = \Theta(g(n))$
$\exists n_0, c_1, c_2$ t.q. $\forall n > n_0$ vale que
$c_1 g(n) \leq f(n) \leq c_2 g(n)$

- (Insertion Sort)
Worst-case $\Theta(n^2)$
best-case $\Theta(n)$

- (Divide and Conquer)
Trata-se de dividir o problema em casos menores

Slide 2 (Divide and Conquer II)

- (Merge Sort)
divide a lista na metade até ter tamanho 1 daí volta apicando sort
runtime = $O(n \log n)$

- (Karatsuba's Algorithm 1960)
Multiplicar 2 inteiros do tipo:
$(a \cdot 10^{n/2} + b)(c \cdot 10^{n/2} + d)$
$= a \cdot b\, 10^n + (ad + b \cdot c) \cdot 10^{n/2} + b \cdot d$
Com $ad + b \cdot c = (a+b)(c-d) - ac - bc$ *

Em * fazemos 3 multiplicações em vez de 4
runtime $(n^{\log_2 3})$

Slide 3 (Sorting Lower Bounds And Binary Search Trees)

- (Counting sort)
Uma lista de n inteiros de 0 a k, esse algoritmo conta as aparições de cada número numa lista de k+1 elementos.
runtime $(n+k)$

- (Bucket sort)
Uma lista de inteiros com a, onde temos buckets e a quantidade de valores por bucket(k) onde adicionamos os elementos em cada bucket de acordo com o intervalo.
runtime $(n+k)$

- (Radix sort)
  Nesse caso usamos o Bucket sort onde temos d o número máximo de dígitos de um número da lista e K a variedade de algarismos. Onde usamos o bucket mas em vez de intervalos usamos d e K.

- (Red-Black tree)
Trata-se de uma (BST) onde os nos podem ser red ou black com as propriedades:
1. Todos os vertices tem cor red ou black.
2. O primeiro vertice é black.
3. Vertices anteriores a Null são black.
4. Vertices anteriores a red vertice são black.
5. Todo caminho de um Null a raiz tem a mesma quantidade de vertices black.

Slide 4 (Randomized Algorithms I)

- (Bogo sort)
  permuta de forma aleatoria a lista e verifica se está ordenada.
  runtime:
  Expected: $O(n \cdot n!)$ Worst-case: $O(\infty)$

- (Quick sort)
  Tome um número aleatório da lista, para os números maiores que ele para um lado e os menores para outro e repete com as sublistas.
  runtime:
  Expected: $O(n \cdot \log n)$ Worst-case: $O(n^2)$

- (Quick select)
  Agora queremos encontrar K na lista, e fazemos e forma equivalente ao Quick sort porém em vez de aplicar nas 2 sublistas aplicamos na que está o K (ou esperamos), até o K ser o pivo escolhido.
  runtime:
  Expected: $O(n)$ Worst-case $O(n^2)$

- (Majority Element)
  Supondo que $\exists$ um elemento que se repete pelo menos $\lfloor n/2 \rfloor + 1$ vezes na lista
  1) Podemos dividir a lista ao meio verificando o que mais se repete até sobrar só um elemento.
  runtime: $O(n \log n)$
  2) Escolher um elemento aleatório e verificar se é ele o que mais se repete
  runtime
  Expected: $O(n)$ Worst-case: $O(\infty)$

Slide 5 (Randomized Algorithms II)

- (Data Structures)

	Sorted linked lists	Sorted Arrays	Balanced BSTs	Hash Tables
Search	$O(n)$	$O(\log n)$	$O(\log n)$	Exp.: $O(1)$ Wor: $O(n)$
Insert/ Delete	$O(n)$	$O(n)$	$O(\log n)$	Exp.: $O(1)$ Wor.: $O(n)$

- (Hash table)
  Queremos construir uma lista de forma equivalente a Bucket sort. But não existe uma função que adiciona em cada bucket de forma a todos os buckets terem $O(1)$.

- (Universal Hash Family)
  São funções "boas" para Hash table por exemplo: a função $h_{a,b}(x)$ é
  $h_{a,b}(x) = ax + b \bmod p \bmod n$
  onde
  $|U|$ todos os possiveis
  p o menor primo t.q. $p \geq |U|$
  a um aleatorio menor que p
  b um aleatorio menor que p