# Algorithm Cheat Sheet

## Asymptotics

| $f(n) = ?$ | o | O | $\Theta$ | $\Omega$ | $\omega$ |
|---|---|---|---|---|---|
| $f(n)/g(n)$ | 0 | $\leq$ | $[c_1, c_2]$ | $\geq c$ | $\infty$ |

### Loop Invariant

Initialization Maintenance Termination

## Solving Recurrences

### Master Theorem

| $f(n)$ | $T(n)$ | $g = n^{log_b a}$ | $T$ |
|---|---|---|---|
| $O(n^{log_b a - \epsilon}), \epsilon > 0$ | $\Theta(n^{log_b a})$ | $f < g$ | $\Theta(g)$ |
| $\Theta(n^{log_b a})$ | $\Theta(n^{log_b a} \log n)$ | $f = g$ | |
| $\Theta(n^{log_b a} \log^k n)$ | $\Theta(n^{log_b a} \log^{k+1} n)$ | $f = \Theta(g \log^k n)$ | $\Theta(g \log^{k+1} n)$ |
| $\Omega(n^{log_b a + \epsilon}), \epsilon > 0$ regularity condition: $af(n/b) \leq cf(n)$, some $c<1$ | $\Theta(f(n))$ r.c. holds for $f(n) = n^k > g$ | $f > g$ & $af(n/b) \leq cf(n)$ | $\Theta(f)$ |

### Simplified

$T(n) = aT(n/b) + cn$

| $a < b$ | $a = b$ | $a > b$ |
|---|---|---|
| $O(n)$ | $O(n \log n)$ | $O(n^{\log_b a})$ |

**Substitution method**:

$T(n) = aT(n/b) + f(n) =$

$a[aT(n/b^2) + f(n/b)] + f(n) =$

$a^2T(n/b^2) + af(n/b) + f(n) = \cdots$

$$\underbrace{a^{\lg_b n}}_{=\Theta(n^{\lg_b a})} \cdot \underbrace{T(n/b^{\lg_b n})}_{=T(1)=\Theta(1)} + \sum_{k=0}^{\lg_b n - 1} a^k f(n/b^k) =$$

<last row> + <all other rows>

**Recursion tree**:

Build a tree from the levels in the substitution:

Level 0: $f(n)$

Level 1: $a$ elements, each costs $f(n/b)$

...

Level $k$: $a^k$ elements, each costs $f(n/b^k)$

Last level: $n^{\lg_b a}$ elements , each costs $\Theta(1)$

## Quicksort

General recurrence: $T(n) = T(k) + T(n-k-1) + \Theta(n)$

W.C.: array already sorted, partition at the beginning:

$T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$

**Note**: for any $n$-proportional division we get $\Theta(n \lg n)$:

$T(n) = T(n/\alpha) + T((\alpha - 1)n/\alpha) + \Theta(n)$

For $\alpha = 2$ we will get a balanced tree.

**Expected running time**:

$E[T(n)] = \sum_k \Pr[k - split] \cdot T(n|k - split) =$

$2/n \cdot \sum_{k=1}^{n-1}(T(k) + \Theta(n)) = \boxed{\Theta(n \lg n)}$

## Selection

**RANDOMIZED-SELECT(A,p,r,i)**

q = RANDOMIZED-PARTITION(A,p,r)

RANDOMIZED-SELECT(A,q+1,r,i-k)

Expected running time

$X_k = I\ \{$subarray $A[p..q]$ has exactly $k$ eles $\}$

$T(n) \leq \sum_{k=1}^{n} X_k \cdot (T(\max(n-k, k-1)) + O(n))$

## Deterministic Selection Algorithm

1. Divide A into groups of 5 elements
2. Find the median in each group (within 7 cmps) $\rightarrow$ subset M
3. Recursively select the median x of M
4. Partition A with respect to pivot x
   at least $3n/10 - 6$ eles are $\leq / \geq x$
5. Recurse on the appropriate part
   $T(n) \leq T(\lceil \frac{n}{5} \rceil) + T(\frac{7n}{10} + 6) + \Theta(n) = O(n)$

## Search Tree

**TREE-SUCCESSOR(x)**

if right[x] $\neq$ NIL

  then return TREE-MINIMUM(right[x])

y $\leftarrow$ p[x]

*//find 1st left parent*

while **y** $\neq$ **NIL** and x = right[y] do

  x $\leftarrow$ y

  y $\leftarrow$ p[y]

return y

## Btree

k keys & k+1 children

#children [t, 2t]. (root except: [2, 2t]

All leaves at same depth

Leaves: same restriction on #keys

$h = \Theta(\log n / \log t)$

If absorb red nodes into their black parents $\rightarrow$ 2-3-4 tree.

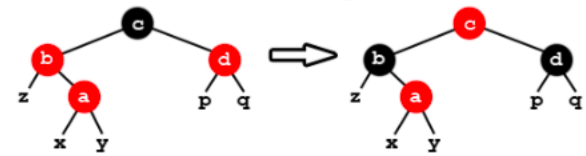
Red-Black trees can be mapped to a 2-3-4 tree and vice-versa

## Red-Black Trees

- Children, parent of a red node are black
- root & leaves(NIL) is black
- All black paths have same # of blacks (black height)
- $h \leq 2\log(n+1)$ The subtree rooted at any node $x$ contains $\leq 2bh(x) - 1$ internal nodes. (by induction on $h$, $bh \leq h/2$

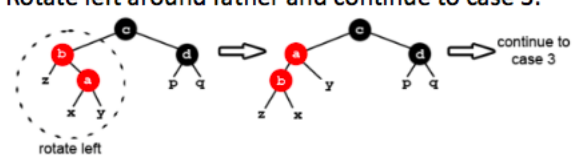
Case 1: a's uncle is red:

Color father and uncle black, and grandfather red.
Problem moved up 2 levels to grandfather.



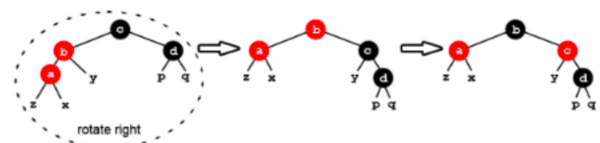

Case 2: a's uncle is black, a is a right child:

Rotate left around father and continue to case 3.




Case 3: a's uncle is black, a is left child:

Rotate right around grandfather, switch colors between father and new sibling.




**RB-INSERT(T, z)**

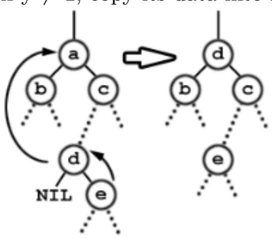
  y = T.NIL //

  x = T.root

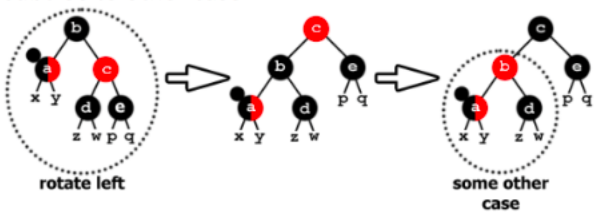
```
while x ≠ T.NIL do
  y = x
  if z.key ¡ x.key then
    x = x.left
  else x = x.right
z.p = y
z.left = T.NIL //
z.right = T.NIL //
z.color = RED //
if y == T.NIL then
  T.root = z
elseif z.key ¡ y.key then
  y.left = z
else
  y.right = z
RB-INSERT-FIXUP(T, z)//
```

### delete

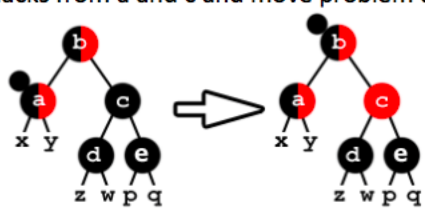
Determine which y to splice out: either z: no/one child or z's successor.
x: NIL or a non-NIL child of y
y is removed by manipulating pointers of p[y] and x. (when y is root!
if y ≠ z, copy its data into z (y is successor)



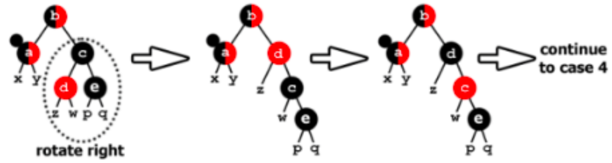

**Case 1: a's sibling is red:**
Rotate left around father, switch colors between father and grandfather and continue with circled subtree to other case.



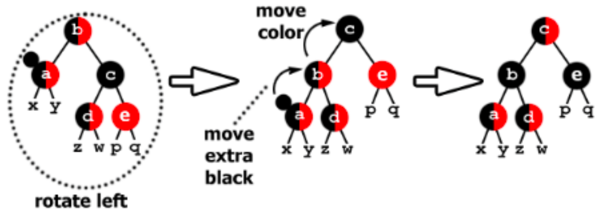

**Case 2: a's sibling and nephews are black:**
Take blacks from a and c and move problem up.



**Case 3: a's sibling is black with left red and right black:**
Rotate right around sibling, switch colors between new sibling and old sibling and continue to case 4.




**Case 4: a's sibling is black with right red:**
Rotate left around father, color grandfather with father's color, color father with extra black, color uncle black.




## Examples

**PARTITION (A,p,r)**

```
x = A[r]
i = p-1
for j=p to r-1
  if A[j] ≤ x then
    i=i+1, swap(A[i],A[j])
swap(A[i+1],A[r])
return i+1
```

**HEAPSORT(A,n)**

```
BUILD-MAX-HEAP(A,n)
for i ← n downto 2 do
  exchange A[1]&A[i]
  MAX-HEAPIFY(A,1,i 1)
```

Top $k$: Compute k largest in sorted order in time $O(n + k \log n)$ Find $k$ largest in online stream: $n \gg k$ elements using space $O(k)$ in $O(n \log k)$ time

**Merge**

```
i=1, j=1
for t = 1 to n1 + n2
  if (i ≤ n1 and (j > n2 or K[i] < L[j])) then
    M[t] = K[i], i = i+1
  else M[t] = L[j], j = j+1
```

**Some hierarchies:** > is "o", = is "Θ"

$2^{2^{n+1}} > 2^{2^n} > (n+1)! > n! > e^n > n \cdot 2^n > 2^n > (3/2)^n > (\lg n)^{\lg n} = n^{\lg \lg n} > (\lg n)! > n^3 > n^2 = 4^{\lg n} > n \lg n = \lg(n!) > n = 2^{\lg n} > (\sqrt{2})^{\lg n} > 2^{\sqrt{2 \lg n}} > \lg^2 n > \ln n > \sqrt{\lg n} > \ln \ln n > 2^{\lg^* n} > \lg^* \lg n = \lg^* n > \lg \lg^* n > 1 = n^{1/\lg n}$

**Some recurrences:**

Binary search: $T(n) = T(n/2) + 1 = O(\lg n)$
Linear search: $T(n) = T(n-1) + 1 = O(n)$
$T(n) = 4T(n/3) + n \lg n \Rightarrow \Theta(n^{\lg_3 4})$ (MT case 1)
$T(n) = 3T(n/3) + n/\lg n \Rightarrow \Theta(n \lg \lg n)$ (tree)
$T(n) = 4t(n/2) + n^2\sqrt{n} \Rightarrow \Theta(n^{2.5})$ (MT case 3)
$T(n) = 3T(n/3 - 2) + n/2 \Rightarrow \Theta(n \lg n)$ (bounds)
$T(n) = 2T(n/2) + n/\lg n \Rightarrow \Theta(n \lg \lg n)$ (tree)
$T(n) = T(n/2) + T(n/4) + T(n/8) + n \Rightarrow \Theta(n)$ (bounds)
$T(n) = T(n-1) + 1/n \Rightarrow \Theta(\lg n)$ (substitution)
$T(n) = T(n-1) + \lg n \Rightarrow \Theta(n \lg n)$ (bounds)
$T(n) = T(n-2) + 1/\lg n \Rightarrow \Theta(n/\lg n)$ (bounds)
$T(n) = \sqrt{n}T(\sqrt{n}) + n \Rightarrow T(n) = \Theta(n \lg \lg n)$ (tree)

## Sorting in Linear Time

### Comparison lower bounds

- Stirling's formula: $n! = \sqrt{2\pi n}(n/e)^n(1 + \Theta(1/n))$
- Simultaneous max and min: $\lceil 3n/2 \rceil - 2$

COUNTING-SORT($A, B, n, k$)

```
for i ← 0 to k range
    do C[i] ← 0
for j ← 1 to n
    do C[A[j]] ← C[A[j]] + 1
for i ← 1 to k
    do C[i] ← C[i] + C[i − 1]
for j ← n downto 1
    do B[C[A[j]]] ← A[j]
       C[A[j]] ← C[A[j]] − 1
```

$\Theta(n + k)$. Auxiliary storage: $C[0..k]$

### Radix Sorting

$\Theta(d(n + k))$.
For n b-bit numbers ($n \le 2^b$), can partition into blocks of r bits, $r \le b \Rightarrow \Theta((b/r)(n + 2^r))$
Optimal: $r \approx \log n \Rightarrow \Theta(bn/\log n)$