CS521 Midterm Cheat-Sheet

Insertion Sort:

Each element is pushed as far as possible to the left (sorted) part of the array. Running time: $O(n^2)$ worst-case (when sorted in reverse), $\Omega(n)$ best-case (when already sorted). Sorts **in-place**, space complexity $O(n)$.

Asymptotic Notations:

Big-O notation:
$f(n) = O(g(n)) \Leftrightarrow \exists c, n_0 > 0 \forall n \geq n_0$:
$0 \leq f(n) \leq c \cdot g(n)$
Or: $\boxed{\lim_{n\to\infty} g(n)/f(n) > 0}$

Small-o notation:
$f(n) = o(g(n)) \Leftrightarrow \exists c, n_0 > 0 \forall n \geq n_0$:
$o \leq f(n) < c \cdot g(n)$
Or: $\boxed{\lim_{n\to\infty} g(n)/f(n) = \infty}$

Big-Ω notation:
$f(n) = \Omega(g(n)) \Leftrightarrow \exists c, n_0 > 0 \forall n \geq n_0$:
$0 \leq c \cdot g(n) \leq f(n)$
Or: $\boxed{\lim_{n\to\infty} g(n)/f(n) = c \geq 0}$ (non-negative constant)
Note that $f(n) = \Omega(g(n)) \Leftrightarrow g(n) = O(f(n))$

Small-ω notation:
$f(n) = \omega(g(n)) \Leftrightarrow \exists c, n_0 > 0 \forall n \geq n_0$:
$0 \leq c \cdot g(n) < f(n)$
Or: $\boxed{\lim_{n\to\infty} g(n)/f(n) = 0}$
Note that $f(n) = \omega(g(n)) \Leftrightarrow g(n) = o(f(n))$

Big-Θ notation:
$f(n) = \Theta(g(n)) \Leftrightarrow \exists c_1, c_2, n_0 \forall n \geq n_0$:
$0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$
Or: $\boxed{\lim_{n\to\infty} g(n)/f(n) = c}$ (some constant)

Notes:
$f(n) = \Theta(g(n)) \Leftrightarrow f(n) \in O(g(n)) \cap \Omega(g(n))$
$f(n) = O(g(n)) \wedge g(n) = O(f(n)) \Rightarrow f(n) = \Theta(g(n))$
$f(n) = \Omega(g(n)) \wedge g(n) = \Omega(f(n)) \Rightarrow f(n) = \Theta(g(n))$

Divide and Conquer – recurrences:

Break a problem into sub-problems, solve sub problems and merge solutions.
Hanoi-tower: $T(n) = 2T(n-1) + 1 = O(2^n)$

Merge-sort:

Divide the sequence into two $n/2$ sequences, apply the algorithm recursively and merge solutions.
$T(n) = \underbrace{2T(n/2)}_{divide} + \underbrace{O(n)}_{merge} = O(n \lg n)$

Solving Recurrences:

Substitution method:
$T(n) = aT(n/b) + f(n) =$
$a[aT(n/b^2) + f(n/b)] + f(n) =$
$a^2 T(n/b^2) + af(n/b) + f(n) = \cdots$
$\underbrace{a^{\lg_b n}}_{=\Theta(n^{\lg_b a})} \cdot \underbrace{T(n/b^{\lg_b n})}_{=T(1)=\Theta(1)} + \sum_{k=0}^{\lg_b n - 1} a^k f(n/b^k) =$
<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 $\Theta(1)$

The Master Theorem:
For $a \geq 1, b > 1$, $f(n)$ function over non-negative integers and $T(n)$ the recurrence:
$T(n) = aT(n/b) + f(n)$

1. $\exists \epsilon > 0: f(n) = O(n^{\lg_b a - \epsilon}) \Rightarrow \boxed{T(n) = \Theta(n^{\lg_b a})}$
2. $f(n) = \Theta(n^{\lg_b a}) \Rightarrow \boxed{T(n) = \Theta(n^{\lg_b a} \lg n)}$
3. $\exists n_0, \epsilon > 0, c < 1: f(n) = \Omega(n^{\lg_b a + \epsilon})$ and $\forall n \geq n_0: af(n/b) \leq c \cdot f(n) \Rightarrow$ $\boxed{T(n) = \Theta(f(n))}$

Notes for MT:
The ϵ addition/subtraction denotes **polynomial** difference in running time between $f(n)$ and $n^{\lg_b a}$.
Cases (1) and (2) derive Θ for some relation between a, b.

Quick Sort:

Partition(A, p, r): linear procedure that partitions A in-place around $x = A[r]$ s.t.: $\boxed{\leq x \quad |x| \quad > x}$ and returns the index of x – the pivot.
Preferable over merge-sort, as it sorts in-place.
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:
$E[T(n)] = \sum_k \Pr[k - split] \cdot T(n|k - split) =$
$2/n \cdot \sum_{k=1}^{n-1} (T(k) + \Theta(n)) = \boxed{\Theta(n \lg n)}$

Heap:

Left filled structure that satisfies **partial order** (node > its children, no particular relation between children).
For node i: left child at $2i$, right at $2i+1$, parent $\lfloor i/2 \rfloor$
Height is $\lg n$ – the cost of **insertion, extract-max, delete**.
Insert: put node x at the end and heapify up (switch with parent(x) until $x <$ parent(x) or x is root.
Remove-max: remove root and put last node of the heap in its place. Heapify it down, each level switching with the greater child of the two until satisfies heap properties.
Build-heap: for node $n/2$ down to 1, heapify down.
Amortized analysis – each level k but lowest will be visited at most $k \cdot (n+1)/2^k$. Total: $(n+1)\sum_{k=0}^{\lg(n+1)} (k/2^k)$.
The sum part is $O(1) \Rightarrow$ total: $O(n)$.

Heap-sort:

Build a heap and then run $\sim n$ times: Swap the max with the last element, decrease heap size by 1 and heapify the new root down. Result: the array is sorted.
Running time: $O(n \lg n)$, sorts in-place.

Selection problems:

The Select algorithm for finding the ith o.s.:
$Select(A, p, q, i)$:
Divide A into $n/5$ groups of size 5.
Find the median for each group and store in A'.
Call $select(A', 1, n/5, n/10)$ to find $x =$ median(A').
Let $k = partition(A, x)$:

- If $i = k$ return x.
- If $i < k$ return $Select(A, p, k-1, i)$
- If $i > k$ return $Select(A, k, q, i-k)$

Running time: $O(n)$.
Select's recurrence:
$T(n) = T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n) = O(n)$ (proof by induction for upper bound.
Simplified Master Theorem:
For $T(1) = d, T(n) = aT(n/b) + cn$:

1. $a < b \Rightarrow T(n) = O(n)$
2. $a = b \Rightarrow T(n) = O(n \lg n)$
3. $a > b \Rightarrow T(n) = O(n^{\lg_b a})$

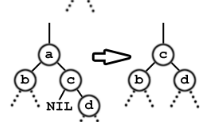
Binary Search Trees:

Satisfy: left(x) $\leq$ key(x) < right(x)
Unbalanced trees: all operations cost $O(h)$, where h is the height of the tree, between $\lg n$ and n.
Successor of x:

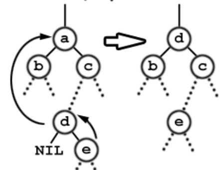
- Left most leaf of x's right subtree.
- If doesn't have one, then the lowest ancestor with left subtree.

Deletion:
Case 1,2:
No left/right child. Put sole child instead.

Case 3:
Successor is right child. Put successor instead.



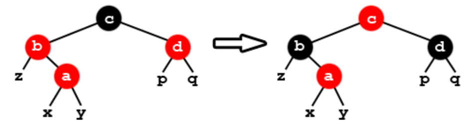
Case 4:
Successor is not direct right child. Swap successor with its sole (right) child, and then swap deleted node with successor.



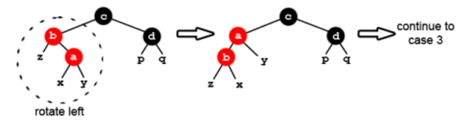
Red-Black Trees:

- Every node is either red or black
- Root and leafs (NIL) are black
- If a node is red, its children will be black
- All black paths have same # of blacks (black height)

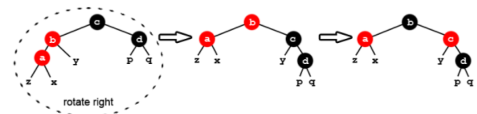
Height: maximum $2 \lg n$
Insertion:
Create new red node with 2 black leafs and put in place. Corrections (a inserted):
Case 1: a's uncle is red:
Color father and uncle black, and grandfather red. Problem moved up 2 levels to grandfather.



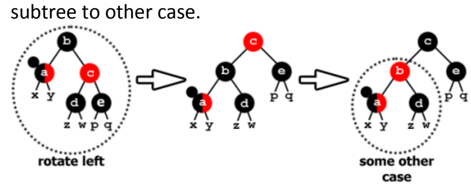
Case 2: a's uncle is black, a is a right child:
Rotate left around father and continue to case 3.



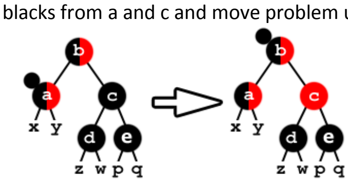
Case 3: a's uncle is black, a is left child:
Rotate right around grandfather, switch colors between father and new sibling.



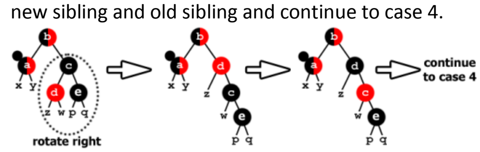
Deletion:
Delete as for any BST. If successor was black, leave behind "extra" black and correct:
Case 1: a's sibling is red:
Rotate left around father, switch colors between father and grandfather and continue with circled subtree to other case.



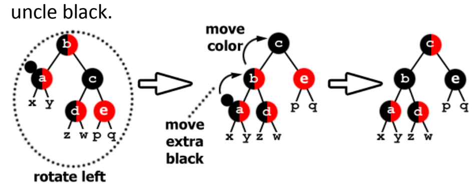
Case 2: a's sibling and nephews are black:
Take blacks from a and c and move problem up.



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:

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

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

Some recurrences:

Binary search: $T(n) = T(n/2) + 1 = O(\lg n)$
Linear search: $T(n) = T(n-1) + 1 = O(n)$
$T(n) = 4T(n/3) + n\lg n \Rightarrow \Theta\left(n^{\lg_3 4}\right)$ (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)
$T(n) = \sqrt{n}T\left(\sqrt{n}\right) + n \Rightarrow T(n) = \Theta(n\lg\lg n)$ (tree)

Math stuff:

$\sum_{i=1}^{n} i = n(n+1)/2 = \Theta(n^2)$
$a^{\lg_b c} = c^{\lg_b a}$
$\sum_{k=1}^{n} \frac{1}{k} = \Theta(\lg n)$ – harmonic series