Notation

f(n) = $Og(n)$ iff $\exists$ c > 0 s.t f(n) <= cg(n), $\forall$ n >= n_0
f(n) = $\Omega g(n)$ iff $\exists$ c > 0 s.t f(n) => cg(n), $\forall$ n >= n_0
f(n) = $\Theta g(n)$ iff $\exists$ c_1, c_2 > 0 s.t c_1g(n) <= f(n) <= c_2g(n), $\forall$ n >= n_0
f(n) = $og(n)$ iff $\lim_{x\to\infty} \frac{f(n)}{g(n)} = 0$
Notes on asymptotic behavior: Omit coefficients!, n^a dominates n^b if a > b, Any exponential dominates any polynomial: (Ex. 3^n dominates n^5), Any polynomial dominates any logarithm: n dominates $(logn)^3$. Implies that $n^2 > nlog(n)$

Algorithms

Binary Search

Average case: $O(log(n))$
General proof: There are n elements in an array, and $n + 1$ locations that lead to an error in binary search, resulting in $2n + 1$ total positions binary search can land during a traversal. $A(n) = \sum_{t=1}^{k} t * Probability$(it takes t comparisons). The algebra results in

Merge Sort

Divide n element array into 2 $\frac{n}{2}$ sets. We conquer by sorting the two subsets recursively. We combine by merging the two sorted sets (2 pointer method to sort the two lists): Mergesort(A,p,r):
if p < r then q = floor($\frac{p+r}{2}$)
Mergesort(A,p,q)
Mergesort(A,q+1,r)
merge(A,p,q,r)
Algorithm is $\Theta(nlogn)$.

Quick Sort

SPLIT: Given an array A[p..r], we want to position A[p] such that everything to its left is less than it while everything to its right is greater than it. We compare A[p] with every element until we find the first element less than it and swap that element with A[p + 1] (If i = 1 then it swaps it with itself). Then continue this comparing A[p] starting with A[p + i + 1] until we find an element less than A[p], which we swap with the first element in the >= section. Repeat until array is empty. Finally, swap A[p] with last element in < section.
Quick sort then is defined as: Quicksort (A, p, r)
if p <r then q = SPLIT (A, p, r)
Quicksort (A, p, q)
Quicksort (A, q + 1, r)
Worst case: $\Theta(n^2)$. Average case (randomized quick sort): $\Theta(nlogn)$

Heap Sort

FixHeap (A, i) swaps the maximum of the three values A[i], A[2i], A[2i + 1] with A[i]. This requires two comparisons. If this causes changing the key value of one of the left or right children, then it repeats the above to the corresponding subtree recursively. Its complexity is O(log n). We then define heapsort by swapping the largest element with the last element in the array, removing it, and then applying FixHeap. This is done n times, each FixHeap is $log(n)$, therefore the above is $O(nlog(n))$. To do this, we must first build a heap with the recursive procedure: BuildHeap (A, i)
BuildHeap (A, 2i)
BuildHeap (A, 2i + 1)
FixHeap (A, i)
The above satisfies the relation T(n) <= 2T($\frac{n}{2}$) + $O(log(n)) = O(n)$.

Integer Multiplication

Karatsuba Algorithm: For x and y Decompose x using a = high ceiling(n/2) bits, b = low floor(n/2) bits and y similarly w/ c and d. Form $w1 = a + b$, $w2 = c + d$, $u = w1 * w2$, $v = ac$, $w = bd$. Solve $xy = 2^n(v) + (2^{\frac{n}{2}} * (u - v - w)) + w$

Polynomial

Vandermonde Matrix (Interpolation) Solve Ax = b via Gaussian Elimination where A is the matrix of computed coefficients, x represents the n coefficients of the n degree polynomial, and b is the y values of the ordered pairs. Horner's rule: $f(x_0) = a_0 + a_1x_0 + a_2x_0^2 + ... + a_nx_0^n$ can be more quickly computed via $f(x_0) = a_0 + x_0(a_1 + x_0(a_2 + x_0(a_3 + ... + (a_n - 1 + a_nx_0)....)$

Solving Recurrence Relations

Divide and Conquer

Generalize to T(n) = aT($\frac{n}{b}$) + D(n) + C(n): a = num subproblems n/b = size subproblems D(n) = time to divide, C(n) = time to combine.

Strategies

Iteration: Keep plugging in until you see a pattern that generalizes
Substitution: Guess a form and then prove by induction
Change of variables: Use some of the tricks below to simplify the problem
Master Theorem: For T(n) = a*T($\frac{n}{b}$)+$O(n^d)$, T(n) = $O(n^d)$ if d > $logs_ba$ | T(n) = $O(n^d * logn)$, if d = log_ba | T(n) = $O(n * log_ba)$, if d < log_ba.

Useful summation formulas

$\sum_{i=0}^{n} i = \frac{n*(n+1)}{2} = \Theta(n^2)$ $\quad$ $\sum_{i=0}^{n} i^2 = \frac{n*(n+1)(2n+1)}{6} = \Theta(n^3)$
$\sum_{i=0}^{n} i^3 = (\frac{n*(n+1)}{2})^2 = \Theta(n^4)$ $\quad$ $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$
$\sum_{i=0}^{n} 2^i = \frac{2^{k+1}-1}{2-1} = 2^{k+1} - 1$ $\quad$ $\sum_{i=1}^{n} i * 2^i = (1 * 2) + (2 * 2^2) + (3 * 2^3) + ... + (k * 2^k)$

Useful Recurrence Relation Tricks

If you see $T(n) = T(n - c) + x$, let $x = 2^n$
If you see $T(n) = T(n^{\frac{1}{c}}) + x$, let $n = \log k$