O-notation: $(\geq)$ $\quad O(g(n)) = \{f(n) : there\ exists\ positive\ constant\ c\ and\ n_0 such\ that\ 0 \leq f(n) \leq cg(n)\ for\ all\ n \geq n_0$

Ω-notation: $(\leq)$ $\quad \Omega(g(n)) = \{f(n) : there\ exists\ positive\ constant\ c\ and\ n_0 such\ that\ 0 \leq cg(n) \leq f(n)\ for\ all\ n \geq n_0$

Θ-notation: $(=)$ $\quad \Theta(g(n)) = \{f(n) : there\ exists\ positive\ constant\ c\ and\ n_0 such\ that\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\ for\ all\ n \geq n_0$

Induction: $for\ n = 1,\ \sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2} \quad for\ n > 1, assume\ \sum_{i=1}^{n} i = \frac{n(n+1)}{2}\ \ show\ \ \sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2}$

$$\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1) = \frac{n(n+1)}{2} + (n+1) = \frac{n(n+1)+2(n+1)}{2} = \frac{(n+1)(n+2)}{2}$$

Logarithims: $\lg^k n = (\lg n)^k \quad \lg\lg n = \lg(\lg n) \quad \log x^y = y\log x \quad \log xy = \log x + \log y \quad \log\frac{x}{y} = \log x - \log y$

$\log_a x = \log_a b \log_b x \qquad a^{\log_b x} = x^{\log_b a} \qquad a = b^{\log_b a}$

Arithmetic series: $\sum_{k=1}^{n} k = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$ **Harmonic series**: $\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \cdots + \frac{1}{n} = \ln n$

Geometric series: $\sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^n = \frac{x^{n+1}-1}{x-1} (x \neq 1)$ **-special case (x <1)**: $\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$

Other important formulas: $\sum_{k=1}^{n} \lg k \approx n\lg n \qquad \sum_{k=1}^{n} k^p = 1^p + 2^p + \cdots + n^p \approx \frac{1}{p+1} n^{p+1}$

Master method: Solving reoccurrences of the form: $T(n) = aT\left(\frac{n}{b}\right) + f(n)\ \ where,\ a \geq 1,\ b \geq 1,\ and\ f(n) > 0$

Case 1: $if\ f(n) = O\left(n^{\log_b a - \epsilon}\right) for\ some\ \epsilon > 0,\ then: T(n) = \Theta\left(n^{\log_b a}\right)$

Case 2: $if\ f(n) = \Theta(\left(n^{\log_b a}\right), then: T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

Case 3: $if\ f(n) = \Omega\left(n^{\log_b a + \epsilon}\right) for\ some\ \epsilon > 0, and\ if\ af\left(\frac{n}{b}\right) \leq cf(n)\ \ for\ some\ c < 1\ and\ all$ sufficiently $large\ n,$ $then: T(n) = \Theta(f(n))$

Expected value: $E[X] = \sum_x x \Pr\{X = x\}\ \ where \Pr(X) = \frac{The\ number\ of\ outcomes\ in\ X}{The\ number\ of\ outcomes\ possible}$

Comparison sorts: $Insertion\ Sort:\ incremental,\ sorts\ in\ place,\ \Theta(n^2)$ $\qquad Merge\ Sort:\ divide\ and\ conquer,\ \Theta(n\lg n)$

$Quick\ Sort: divide\ and\ conquer,\ sorts\ in\ place,\ \Theta(n\lg n)$

Order of growth: $lim\frac{T(n)}{g(n)}\ as\ n\ approaches\ \infty =\ 0\ order\ of\ growth\ of\ T(n) < order\ of\ growth\ of\ g(n)$

$= \ c > 0\ order\ of\ growth\ of\ T = order\ of\ growth\ of\ g(n)$

$= \ \infty\ order\ of\ growth\ of\ T(n) > order\ of\ growth\ of\ g(n)$

All logarithmic functions $\log_a n$ belong to the same class $\Theta(\lg n)$ no matter what the logarithm's base a > 1 is

All polynomials of the same degree k belong to the same class: $a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0 \in \Theta(n^k)$

Exponential functions a^n have different orders of growth for different a's

Order $\log n <$ order n^α $(\alpha > 0) <$ order $a^n <$ order $n! <$ order n^n

Asymptotic Notation Examples:

1. $f(n) = \log n^2; g(n) = \log + 5$	$f(n) = \Theta(g(n))$
2. $f(n) = n; g(n) = \log n^2$	$f(n) = \Omega(g(n))$
3. $f(n) = \log\log n; g(n) = \log n$	$f(n) = O(g(n))$
4. $f(n) = n; g(n) = \log^2 n$	$f(n) = \Omega(g(n))$
5. $f(n) = n\log n + n; g(n) = \log n$	$f(n) = \Omega(g(n))$
6. $f(n) = 10; g(n) = \log 10$	$f(n) = \Theta(g(n))$
7. $f(n) = 2^n; g(n) = 10n^3$	$f(n) = \Omega(g(n))$
8. $f(n) = 2^n; g(n) = 3^n$	$f(n) = O(g(n))$

$T(n) = T(n-1) + n \qquad \Theta(n^2)$
– Recursive algorithm that loops through the input to eliminate one item

$T(n) = T(n/2) + c \qquad \Theta(\lg n)$
– Recursive algorithm that halves the input in one step

$T(n) = T(n/2) + n \qquad \Theta(n)$
– Recursive algorithm that halves the input but must examine every item in the input

$T(n) = 2T(n/2) + 1 \qquad \Theta(n)$
– Recursive algorithm that splits the input into 2 halves and does a constant amount of other work

Indicators Random Variable: Given a sample space S and an event A, we define the indicator random variable X_A associated with A: $X_A = \begin{cases} 1, & If\ A\ occurs \\ 0, & If\ A\ does\ not\ occur \end{cases}$ The expected value of an indicator random variable X_A is $E[X_A] = \Pr\{A\}$

Binary tree lemma: Any binary tree of height h has at most 2^h leaves

Heap: A heap is a nearly complete binary tree with the following two properties: All levels are full except possibly the last one, which is filled L to R

Order (MAX heap) property- for any node x, Parent(x) ≥ x

Operations on Heaps:

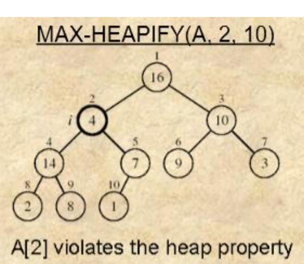
MAX-HEAPIFY	Maintain the max-heap property	$O(\lg n)$
BUILD-MAX-HEAP	Create a max-heap from an unordered array	$O(n)$
HEAPSORT	Sort an array in place	$O(n\lg n)$
Priority queue operations		

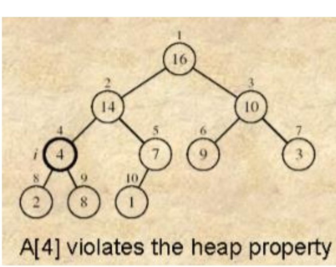
– Root of tree is A[1]
– Left child of A[i] = A[2i]
– Right child of A[i] = A[2i + 1]
– Parent of A[i] = A[⌊i/2⌋]
– Heapsize[A] ≤ length[A

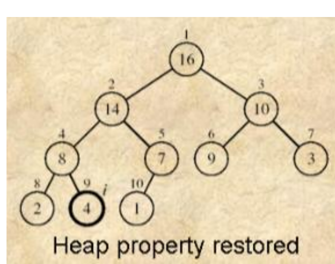
Priority Queue Operations:

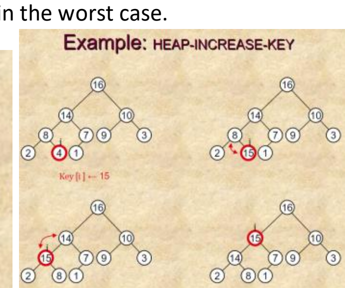
MAXIMUM(S)	Returns elements of S with largest key	$O(1)$
EXTRACT-MAX(S)	Removes and returns element of S with largest key	$O(\lg n)$
INCREASE-KEYS(S, x, k)	Increases value of element x's key to k (Assume k ≥ x's current key value)	$O(\lg n)$
INSERT(S, x)	Inserts element x into set S	$O(\lg n)$

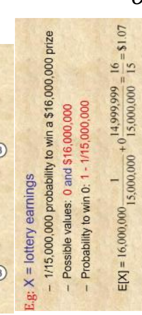
Lower Bound for Comparison Sorts: Any comparison sort algorithm requires $\Omega(n\lg n)$ comparisons in the worst case.

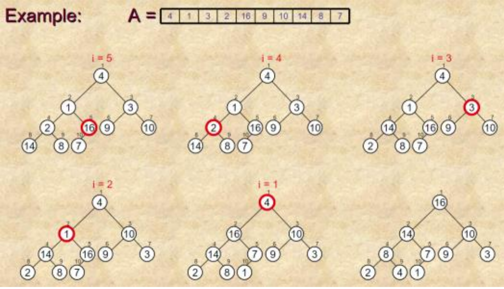




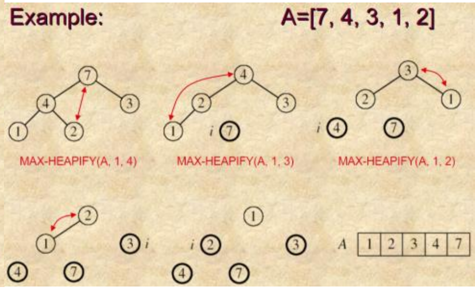




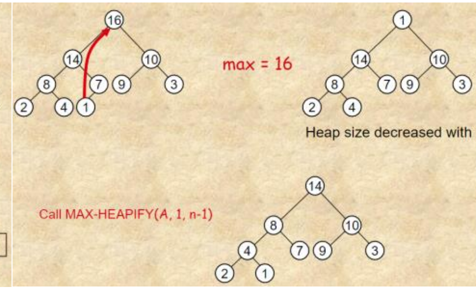




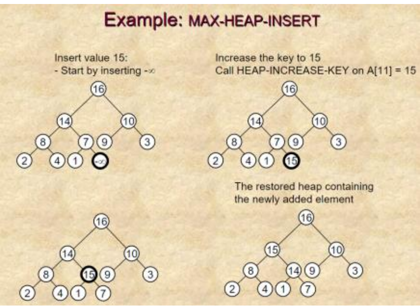
BUILD-HEAP



HEAPSORT



HEAP-EXTRACT-MAX



MAX-HEAP-INSERT

7.2-1
Use the substitution method to prove that the recurrence $T(n) = T(n-1) + \Theta(n)$ has the solution $T(n) = \Theta(n^2)$, as claimed at the beginning of Section 7.2.

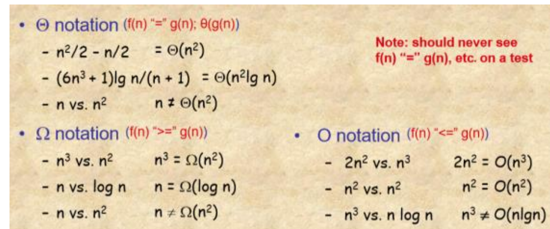
$$
\begin{aligned}
T(n) &\le T(n-1) + dn \\
&= c(n-1)^2 + dn \\
&= cn^2 - 2cn + c + dn \\
&\le cn^2
\end{aligned}
$$

where the last step holds for when $2c > d$ and $c \ge \frac{dn}{2n-1}$.

7.2-2
What is the running time of QUICKSORT when all elements of array A have the same value?

7.2-3
Show that the running time of QUICKSORT is $\Theta(n^2)$ when the array A contains distinct elements and is sorted in decreasing order.

7.2-4
Banks often record transactions on an account in order of the times of the transactions, but many people like to receive their bank statements with checks listed in order by check number. People usually write checks in order by check number, and merchants usually cash them with reasonable dispatch. The problem of converting time-of-transaction ordering to check-number ordering is therefore the problem of sorting almost-sorted input. Explain persuasively why the procedure INSERTION-SORT might tend to beat the procedure QUICKSORT on this problem.



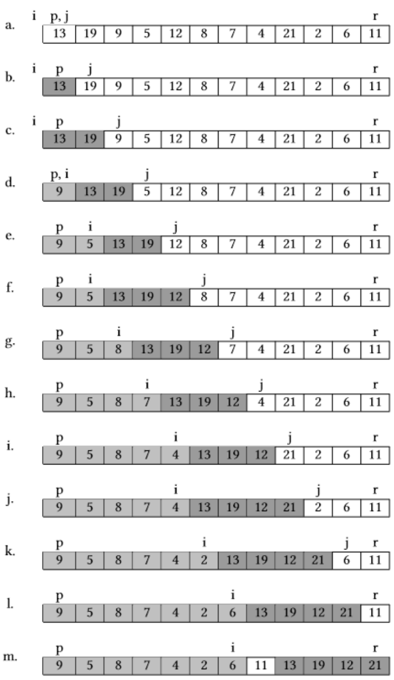
This case represents a worst-case scenario for the running of QUICKSORT because the $A[p..q-1]$ will always contain all the found elements, while the $A[q+1..r]$ partition will be empty. The runtime in this case is $\Theta(n^2)$ as outlined in the text. 7.2-2

This is another expression of the worst-case scenario of QUICKSORT. The pivot element selected from this array will always be less than all other elements and so we will partition the remaining elements such that one partition is empty. By partitioning the elements in this way, we get a recurrence of the form described in Exercise 7.2-1 which runs in $\Theta(n^2)$ time. 7.2-3

7.2-4

Insertion sort requires less labor the more sorted the array is. $\Theta(n + d)$, where d is the number of inversions in the array. Since there are typically few inversions in the aforementioned example, insertion sort will be close to linear.

However, if PARTITION does choose a pivot that is not participate in an inversion, it will result in an empty partition. QUICKSORT is quite likely to yield empty partitions because there aren't many inversions.



PARTITION on A={13,19,9,5,12,8,7,4,21,2,6,11}

7.4-3

$$
\begin{aligned}
f(q) &= q^2 + (n-q-1)^2 \\
f'(q) &= 2q - 2(n-q-1) = 4q - 2n + 2 \\
f''(q) &= 4.
\end{aligned}
$$

7.4-1
Show that the recurrence

$$T(n) = \max\{T(q) + T(n-q-1) : 0 \le q \le n-1\} + \Theta(n)$$

has a lower bound of $T(n) = \Omega(n^2)$.

$f'(q) = 0$ when $q = \frac{1}{2}n - \frac{1}{2}$. $f(q)$ is decreasing in the beginning of the interval and increasing in the end, which means that those two points are the only candidates for a maximum in the interval.

$$
\begin{aligned}
f(0) &= (n-1)^2 \\
f(n-1) &= (n-1)^2 + 0^2.
\end{aligned}
$$

7.4-2
Show that quicksort's best-case running time is $\Omega(n \lg n)$.

7.4-3
Show that the expression $q^2 + (n-q-1)^2$ achieves its maximum value over $q = 0, 1, \ldots, n-1$ when $q = 0$ or $q = n-1$.

7.4-4
Show that RANDOMIZED-QUICKSORT's expected running time is $\Omega(n \lg n)$.

7.4-4

$$
\begin{aligned}
[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \\
&= \sum_{i=1}^{n-1} O(\lg n) \\
&= O(n \lg n).
\end{aligned}
$$

7.4-2 We'll use the substitution method to show that the best-case running time is $\Omega(n \lg n)$. Let $T(n)$ be the best-case time for the procedure QUICKSORT on an input of size n. We have:

$$T(n) = \min_{1 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n).$$

Suppose that $T(n) \ge c(n \lg n + 2n)$ for some constant c. Substituting this guess into the recurrence gives:

$$
\begin{aligned}
T(n) &\ge \min_{1 \le q \le n-1} (cq \lg q + 2cq + c(n-q-1)\lg(n-q-1) + 2c(n-q-1)) + \Theta(n) \\
&= (cn/2)\lg(n/2) + cn + c(n/2-1)\lg(n/2-1) + cn - 2c + \Theta(n) \\
&\ge (cn/2)\lg n - cn/2 + c(n/2-1)(\lg n - 2) + 2cn - 2c\Theta(n) \\
&= (cn/2)\lg n - cn/2 + (cn/2)\lg n - cn - c\lg n + 2c + 2cn - 2c\Theta(n) \\
&= cn \lg n + cn/2 - c\lg n + 2c - 2c\Theta(n).
\end{aligned}
$$

7.4-1 We guess $T(n) \ge cn^2 - 2n$

$$
\begin{aligned}
T(n) &= \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n) \\
&\ge \max_{0 \le q \le n-1} (cq^2 - 2q + c(n-q-1)^2 - 2n - 2q - 1) + \Theta(n) \\
&\ge c \max_{0 \le q \le n-1} (q^2 + (n-q-1)^2 - (2n + 4q + 1)/c) + \Theta(n) \\
&\ge cn^2 - c(2n-1) + \Theta(n) \\
&\ge cn^2 - 2cn + 2c && (c \le 1) \\
&\ge cn^2 - 2n.
\end{aligned}
$$

Proof:
lg(n!) = lg n + lg(n-1) + … + lg (n/2) + lg ((n-1)/2) + … + lg 3 + lg 2 + lg 1
≥ lg n + lg(n-1) + lg(n-2) + … + lg (n/2)
≥ lg(n/2) + lg(n/2) + lg (n/2) + … + lg (n/2)
= ½ n lg(n/2)
= ½ n lg n - ½ n
≥ c n lg n

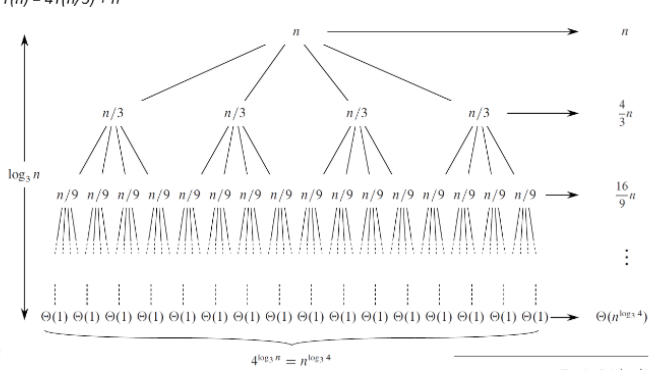
8.1-1
What is the smallest possible depth of a leaf in a decision tree for a comparison sort?

8.1-2
Obtain asymptotically tight bounds on $\lg(n!)$ without using Stirling's approximation. Instead, evaluate the summation $\sum_{k=1}^{n} \lg k$ using techniques from Section A.2.

For a permutation $a_1 \le a_2 \le \ldots \le a_n$, there are $n-1$ pairs of relative ordering, thus the smallest possible depth is $n-1$.

$\Theta(n^{\log_3 4})$

T(n) = 4T(n/3) + n



Prob. 4 (4.5-1) Answer:

In all parts of this problem, we have $a = 2$ and $b = 4$, and thus $n^{\log_b a} = n^{\log_4 2} = n^{1/2} = \sqrt{n}$.

a. $T(n) = \Theta(\sqrt{n})$. Here, $f(n) = O(n^{1/2-\epsilon})$ for $\epsilon = 1/2$. Case 1 applies, and $T(n) = \Theta(n^{1/2}) = \Theta(\sqrt{n})$.

b. $T(n) = \Theta(\sqrt{n}\lg n)$. Now $f(n) = \sqrt{n} = \Theta(n^{\log_b a})$. Case 2 applies, with $k = 0$.

c. $T(n) = \Theta(\sqrt{n}\lg^3 n)$. Now $f(n) = \sqrt{n}\lg^2 n = \Theta(n^{\log_b a}\lg^2 n)$. Case 2 applies, with $k = 2$.

d. $T(n) = \Theta(n)$. This time, $f(n) = n^1$, and so $f(n) = \Omega(n^{\log_b a+\epsilon})$ for $\epsilon = 1/2$. In order for case 3 to apply, we have to check the regularity condition: $af(n/b) \le cf(n)$ for some constant $c < 1$. Here, $af(n/b) = n/2$, and so the regularity condition holds for $c = 1/2$. Therefore, case 3 applies.

e. $T(n) = \Theta(n^2)$. Now, $f(n) = n^2$, and so $f(n) = \Omega(n^{\log_b a+\epsilon})$ for $\epsilon = 3/2$. In order for case 3 to apply, we again have to check the regularity condition: $af(n/b) \le cf(n)$ for some constant $c < 1$. Here, $af(n/b) = n^2/8$, and so the regularity condition holds for $c = 1/8$. Therefore, case 3 applies.

Alg.: BINARY-SEARCH (A, lo, hi, x)

```
if (lo > hi)                                  ← constant time: c_1
    return FALSE
mid ← ⌊(lo+hi)/2⌋                             ← constant time: c_2
if x = A[mid]                                 ← constant time: c_3
    return TRUE
if ( x < A[mid] )
    BINARY-SEARCH (A, lo, mid-1, x)           ← same problem of size n/2
if ( x > A[mid] )
    BINARY-SEARCH (A, mid+1, hi, x)           ← same problem of size n/2
```

- $T(n) = c + T(n/2)$
- $T(n)$ – running time for an array of size n

Prob. 5 (4.5-4) Answer:

In order for af(n/b) ≤ cf(n) to hold with a = 1, b = 2, and f(n) = lg n, we would need to have (lg(n/2))/= lg n < c. Since lg(n/2) = lg n - 1, we would need (lg n - 1)/ lg n < c. For any constant c < 1, there exist an infinite number of values for n for which this inequality does not hold.

Furthermore, since $n^{\log_b a} = n^0$, there is no constant ϵ > 0 such that lg n = Ω(n^ϵ).

4.3-1

a. If $T(n) = O(n^2)$ we must prove $T(n) \le cn^2$

$T(n-1) \le c(n-1)^2$
$T(n) \le c(n-1)^2 + n$
$\le c(n^2 - 2n + 1) + n$
$= cn^2 - 2cn + c + n$
$= cn^2 + (1-2c)n + c$
$\le cn^2 + (1-2c)n + 1$ for $c \ge 1$
$\le cn^2 + (1-2c) + 1$ for $n \ge 1$
$= cn^2 + (2-2c)$
$\le cn^2$ since $(2-2c) \le 1$ and $c \ge 1$ the inequation holds for $c \ge 1$
$T(n) = O(n^2)$

b. If $T(n) = O(\lg n)$ we must prove $T(n) \le c\lg n$

$T(n/2) \le c\lg(n/2)$
$T(n) \le c\lg(n/2) + \Theta 1$
$= c\lg n - c\lg 2 + 1$
$= c\lg n - c + 1$
Prove that $T(n) \le c\lg(n-d)$ where d is some constant
$T(n) \le c\lg(n/2 - d) + 1$
$= c\lg(n/2 - d + 1) + 1$
$= c\lg((n - 2d + 2)/2) + 1$
$= c\lg(n - 2d + 2) - c\lg 2 + 1$
$= c\lg((n-d) - (d-2)) - (c-1)$
$\le c\lg(n-d)$ for $n \ge 6$ and $d \ge 2$ and $c \ge 2$
$T(n) = O(\lg n)$

c. If $T(n) = \Theta(n\lg n)$ we must prove $T(n) \le cn\lg n$)

$T(n) = 2T(n/2) + n$
$= 2cn/2\lg(n/2) + n$
$= cn\lg n - cn\lg 2 + n$
$= cn\lg n - cn + n$
$\le cn\lg n$
$T(n) = \Theta(n\lg n)$

d. If $T(n) = O(n\lg n)$ we must prove $T(n) \le cn\lg n + n$)

[redacted]

$T(n) \le 2c(n/2 + 17)\lg(n/2 + 17) + n$
$= 2c(n/2 + 17)\lg(n/2 + n/4) + n$
$= (cn + 34c)\lg(3n/4) + n$
$= cn\lg n \breve{}\, cn\lg 4/3 + 34c\lg n \breve{}\, 34c\lg 4/3 + n$
$= cn\lg n \breve{}\, cn\lg 4/3 + 34c\lg n + n$
$\le cn\lg n + n(1 - c\lg 4/3) + kn$ As $\lg n = 0(n)$ therefore $34c\lg n \le kn$ and $n \ge n_0$
$\le cnlgn + n(k + 1 - clg4/3)$
$T(n) = O(n\lg n)$ for some $c \ge k$ and $n \ge max\{n_0, 68\}$

e. If $T(n) = \Theta(n)$ we must prove $T(n) \ge dn$)

$T(n) \ge 2T(n/3) + \Theta(n)$
$= 2d(n/3/3) + n/3$
$= 2dn + n/3$
$2dn + n/3 \ge dn$
$T(n) = \Theta(n)$

f. If $T(n) = \Theta(n^2)$ we must prove $T(n) \ge dn^2$)

$T(n) \ge 4T(n/2) + \Theta(n)$
$= 4d(n/2)^2 + n$
$= dn^2 + n$
$dn^2 + n \ge dn^2$
$T(n) = \Theta(n^2)$

Alg.: <u>function fib(n)</u>

```
if n = 0: return 0
create array f[0…n]
f[0] = 0, f[1] = 1
for i = 2...n
    f[i] = f[i-1] + f[i – 2]
return f[n]
```

Is it correct? Sure, again by definition.

Running time: ***T(n)* = 2**[loop addition] · ***(n - 1)***[for loop] **+ 2**[initialization] **= 2 (n – 1) + 2**

Cool! We have an *algorithm* that is *linear* in steps to process

E.g. T(200) ≈ 400 steps

Example (Exact Solution; check boundary conditions and the base case***):***

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n > 1 \end{cases}$$

1. *Guess: T (n) = n lg n + n*

2. *Induction:*

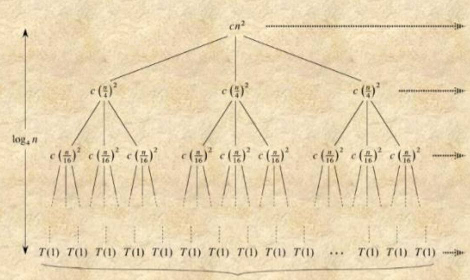
Basis: $n = 1 \Rightarrow n\lg n + n = 1 = T(n)$

Inductive step: Inductive hypothesis is that $T(k) = k\lg k + k$ for all $k < n$.

We'll use this inductive hypothesis for $T(n/2)$.

$T(n)$ $= 2T(n/2) + n$
$= 2(n/2\lg n/2 + n/2) + n$ (by inductive hypothesis)
$= n\lg n/2 + n + n$
$= n(\lg n - \lg 2) + n + n$
$= n\lg n - n + n + n$
$= n\lg n + n$

Example 2 $T(n) = 3T(n/4) + cn^2$



level	size	nodes	cost
0	$n/4^0$	1	cn^2
1	$n/4^1$	3^1	$3/16cn^2$
2	$n/4^2$	3^2	$(3/16)^2cn^2$
$\log_4 n$	$n/4^{\log n}$	$3^{\log n}$	

- Sub problem size hits 1 when $1 = n/4^i \Rightarrow i = \log_4 n$
- Number of nodes at level $i = 3^i \Rightarrow$ cost of nodes at level $i = (3/16)^i cn^2$
- Total cost: $T(n) = \sum_{i=0}^{\log_4 n-1}\left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4 3}\right) \le \sum_{i=0}^{\infty}\left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4 3}\right) = \frac{1}{1-\frac{3}{16}}cn^2 + \Theta\left(n^{\log_4 3}\right) = O(n^2)$

$\Rightarrow T(n) = O(n^2)$

Medians - Example

1. S = {2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1}, v = 5; want **Selection**(S, 8)
   S_L = {2, 4, 1} S_v = {5, 5} S_R = {36, 21, 8, 13, 11, 20}
   - 8^{th} element must be in S_R, since $|S_L| + |S_v| = 5$
   - So want Selection(S_R, 8 - $|S_L|$ - $|S_v|$) = Selection(S_R, 3)

2. S = {36, 21, 8, 13, 11, 20}, v = 13; want **Selection**(S, 3)
   S_L = {8, 11} S_v = {13} S_R = {20, 21, 36}
   - 3^{rd} element must be in S_v
   - So 3^{rd} smallest element is 13

E.g.: flipping two coins:

- Earn $3 for each head, lose $2 for each tail
- X: random variable representing your earnings
- Three possible values for variable X:
  - 2 heads ➔ x = $3 + $3 = $6, Pr{2 H's} = ¼
  - 2 tails ➔ x = -$2 - $2 = -$4, Pr{2 T's} = ¼
  - 1 head, 1 tail ➔ x = $3 - $2 = $1, Pr{1 H, 1 T} = ½
- The expected value of X (your earnings) is:

E[X] = 6 ∗ Pr{2 H's} + 1 ∗ Pr{1 H, 1 T} – 4 ∗ Pr{2 T's} = 6 ∗ ¼ + 1 ∗ ½ - 4 ∗ ¼ = $1

Analysis of Merge Sort

- Running time $T(n)$ of Merge Sort:
- Divide: split into two arrays takes $\Theta(1)$
- Conquer: solving 2 subproblems takes $2T(n/2)$
- Combine: merging n elements takes $\Theta(n)$
- Total:

$$\begin{cases} T(n) = \Theta(1) & \text{if } n = 1 \\ T(n) = 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

➔ $T(n) = \Theta(n \lg n)$

Alg.: function fib(n)
 if n = 0: **return** 0
 if n = 1: **return** 1
 return fib(n - 1) + fib(n - 2)

Is it correct? Sure, by definition.
Total Running Time on input of size n, $T(n)$:

- *the number of primitive operations (steps) executed before termination*
- $T(n) \le 2$ for $n \le 1$
- $T(n) = T(n - 1) + T(n - 2) + 3$ for $n > 1$
  $\approx \mathbf{2}T(n\text{-}1) = \mathbf{2[2}T(n\text{-}2)] = \mathbf{2[2[2}T(n\text{-}3)] = \ldots = \mathbf{2^k}T(n\text{-}k)$

or $T(n) \approx 2^{(n-1)}2 = \mathbf{2^n}$ when **k = n-1**

- $T(200) \approx 2^{200}$ = **1.607×10^{60}** (actually < 2^{200})

- Expected value (expectation, mean) of a discrete random variable X is:

$$E[X] = \Sigma_x \; x \Pr\{X = x\}$$

- "Average" over all possible values of random variable X

E.g.: X = face of one fair dice
E[X] = $1 \cdot 1/6 + 2 \cdot 1/6 + 3 \cdot 1/6 + 4 \cdot 1/6 + 5 \cdot 1/6 + 6 \cdot 1/6 = 3.5$

4.3-2 Let us assume $T(n) \le cn^2$ for all $n \ge n_0$ where c and n_0 are positive constants.
$T(n) = 4T(n/2) + n$
$\le 4c(n/2)^2 + n$
$\le cn^2 + n$
Now, let us assume $T(n) \le cn^2 - bn$ for all $n \ge n_0$, where b, c, and n_0 are positive integers
$T(n) = T(n/2) + n$
$\le 4(c(n/2)^2 - (bn/2)) + n$
$\le 4(c(n^2/4) - (bn/2)) + n$
$\le cn^2 - 2bn + n$
$\le cn^2 - bn - (b-1)n$
$\le cn^2 - bn$ as long as $(b-1)n$ is positive, So $n_0 = 1$ and $b \ge 1$

- *Theorem:* $f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$
- **Transitivity:**
  - $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$
  - Same for O and Ω
- Reflexivity:
  - $f(n) = \Theta(f(n))$
  - Same for O and Ω
- Symmetry:
  - $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$
- Transpose symmetry:
  - $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$