**DAA-Exam#1-Prep**

1) Use a method (either summation, substitution, or master method) to solve each T($n$) of the following recurrence relations. You need to show clear steps to justify your answers. If you encounter an irrational number, leave as it is, for example, $\sqrt{13}$ , $\lg_5 7$, etc.

**(a)** $T(n) = 4T(n/2) + n^2$ $O(n^2 \lg n)$.

**(b)** $T(n) = 6T(n/2) + \frac{n^3}{\lg n}$ $O(n^3)$

**(c)** $T(n) = 8T(n/3) + n\lg n$ $O(n^{\lg_3 8})$

**(d)** $T(n) = 8T(n/4) + \frac{n^2}{\sqrt{n}}$ $O(n^{1.5}\lg n)$

**(e)** $T(n) = 7T(n/2) + n^3$ $O(n^3)$

**(f)** $T(n) = T(n/4) + \frac{1}{\lg n}$ ~ O(loglogn)

**(g)** $T(n) = 16T(n/3) + n^2 \lg n$ ~ $O(n^{\lg_3 16})$

**(h)** $T(n) = 8T(n/4) + n\sqrt{n}$ $O(n^{1.5}\lg n)$

- O notation: asymptotic "less than": f(n) "≤" g(n)
- Ω notation: asymptotic "greater than": f(n) "≥" g(n)
- Θ notation: asymptotic "equality": f(n) "=" g(n)

2) For each problem, write valid asymptotic relationship between f and g using O, Ω, and Θ

1. $f(x) = x^2 + 1$, $g(x) = 3x - 2$, → $f(x) = \Omega(g(x))$
2. $f(x) = x^2 + 5$, $g(x) = 3x^2 + 4x$, → $f(x) = \Theta(g(x))$
3. $f(x) = 2^x + 3x$, $g(x) = 3^x + 2x + 1$, → $f(x) = O(g(x))$
4. $f(x) = 2x + 1$, $g(x) = 3x + 2$, → $f(x) = \Theta(g(x))$
5. $f(x) = 2\log x$, $g(x) = \log_3(2x)$, → $f(x) = \Omega(g(x))$
6. $f(x) = \sqrt{x}$, $g(x) = 4\log x$, → $f(x) = \Omega(g(x))$
7. $f(x) = 3\log^2 x + 2$, $g(x) = 2x + 1$, → $f(x) = O(g(x))$

For each of the following pairs of functions, either f(n) is O(g(n)), f(n) is Ω(g(n)), or f(n) = Θ(g(n)). Determine which relationship is correct.

1. $f(n) = \log n^2$; $g(n) = \log n + 5$ , $f(n) = \Theta(\log n) = \Theta(g(n))$
2. $f(n) = n$; $g(n) = \log n^2$ , $f(n) = \Omega(\log n) = \Omega(g(n))$
3. $f(n) = \log \log n$; $g(n) = \log n$ , $f(n) = O(\log n) = O(g(n))$

3). Show that

(a) $5n^2 = O(n^2)$ $5n^2 < 10n^2 + 100$

(b) $n^2 + 10n = O(n^2)$ $n^2 + 10n < 10n^2 + 100$

(c) $n = O(n^2)$ $n < 10n^2 + 100$

(d) $5n^2 = \Omega(n^2)$ $5n^2 > n^2$ , when n >= 1
(e) $n \neq \Theta(5n^2)$ $n = O(n^2)$ , but n != $\Omega(n^2)$
(f) $6n^2 + n \neq \Omega(n^3)$ , $n^3 > n^2$
(g) $a^n = O(b^n)$, b>a>1 , $b^n > a^n$ , $100^n > 2^n$

(h) lgn = O(n), lgn < n

4) **True** or false?

1. $n^2 = O(n^3)$ True
2. $n^2 = \Theta(n^3)$ False
3. $(n + k)^m = \Theta(n^m)$ True
4. $2^{n+1} = O(2^n)$ True
5. $2^{2n} = O(2^n)$, $4^n > 2^n$ False
6. $\sqrt{\log n} = O(\log\log n)$ False
7. $n^{\log n} = O(2^n)$ True
8. $4n^2 + 2n + 4 = O(n^2 + n + 1)$ True

Express the function $n^3/1000 - 100n^2 - 100n + 3$ in terms of Θ notation. $\Theta(n^3)$

5) Solve each T(*n*) of the following root functions (in recurrence relations).

**$T(n) = T(\sqrt{n}) + 1$**
$T(2^m) = T(2^{m/2}) + 1$
$T(2^m) = S(m)$, m = log n, $n = 2^m$
$S(m) = S(m/2) + 1$
$\log_2 1 = 0$, k = 0, p =0, case 2-1,
m = log n,
O(log**m**) → O(loglogn)

**$T(n) = 2T(\sqrt{n}) + 1$**

$T(2^m) = S(m)$, m = log n, $n = 2^m$
$S(m)=2S(m/2)+1$
$\log_2 2 = 1$, k = 0, p =0, case 1,
O(m)
O(logn)

**$T(n) = 2T(\sqrt{n}) + \log n$**

$T(2^m) = S(m)$, $m = \log n$, $n = 2^m$

$S(m)=2S(m/2)+m$

$\log_2 2 = 1$, $k = 1$, $p = 0$, case 2,

$O(m\log m)$

$O(\log n * \log\log n)$

**$T(n) = T(\sqrt{n}) + \log n$**

$T(2^m) = S(m)$, $m = \log n$, $n = 2^m$

$S(m)=S(m/2)+m$

$\log_2 1 = 0$, $k = 1$, $p = 0$, case 3,

$O(m)$

$O(\log n)$

**$T(n) = T(\sqrt{n}) + \log n * \log n$**

$T(2^m) = S(m)$, $m = \log n$, $n = 2^m$

$S(m)=S(m/2)+m^2$

$\log_2 1 = 0$, $k = 2$, $p = 0$, case 3,

$O(m^2)$

$O(\log n * \log n)$

**$T(n) = 16T(\sqrt{n}) + \log^4 n$**

$m = \log n$, $n = 2^m$

$T(2^m) = S(m)$,

$S(m)=16S(m/2)+ m^4$

$\log_2 16 = 4$, $k = 4$, $p = 0$, case 2,

$O(m^4 \log m)$

$O(\log^4 n * \log\log n)$

Solve each $T(n)$ of the following Multi-depth functions (in recurrence relations).

$T(n) = T(n/2) + T(n/4) + T(n/8) + n$ → $O(n)$

$T(n) = T(n/2) + T(n/4) + T(n/8) + 1$ → $O(\log n)$

$T(n) = T(n/3) + T(2n/3) + n$ → $O(n\log n)$ (quick sort case)

**Complexity Comparison:**

**$n^{\sqrt{n}} : 2^n : n^{3/2} : n\log n : n^{\log n}$**

**$2^n > n^{\sqrt{n}} > n^{\log n} > n^{3/2} > n\log n$**

(8) $f(n) = n^3$ $0 < n < 10{,}000$

$= n^2$ $n >= 10{,}000$

g(n) = n  $0 < n < 100$
= $n^3$  $n >= 100$

$3\mathbf{n}^{\sqrt{n}}$ : $\mathbf{2}^{\sqrt{n}\log n}$

Order the following 6 functions in increasing order of their growth rates:

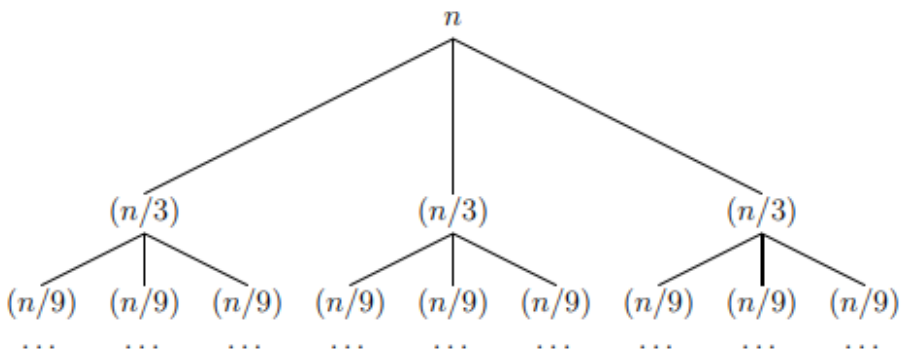
– *nlog n, $log^2 n$, $n^2$, $2^n$,* $\sqrt{\mathbf{n}}$ , *n.*

$log^2 n$
$\sqrt{\mathbf{n}}$
*n*
*nlogn*
$n^2$
$2^n$

Q. Consider a recursion tree that looks like this:

1. What recurrence relation could generate this recursion tree? T(n) = 3T(n/3) + n
2. How many levels would there be in this tree, as a function of n? $\log_3 n$
3. How many leaves would be in this tree, as a function of n? $n^{\log_3 3} = n$
4. Solve the recurrence to obtain an asymptotic expression for T(n) as a function of n.

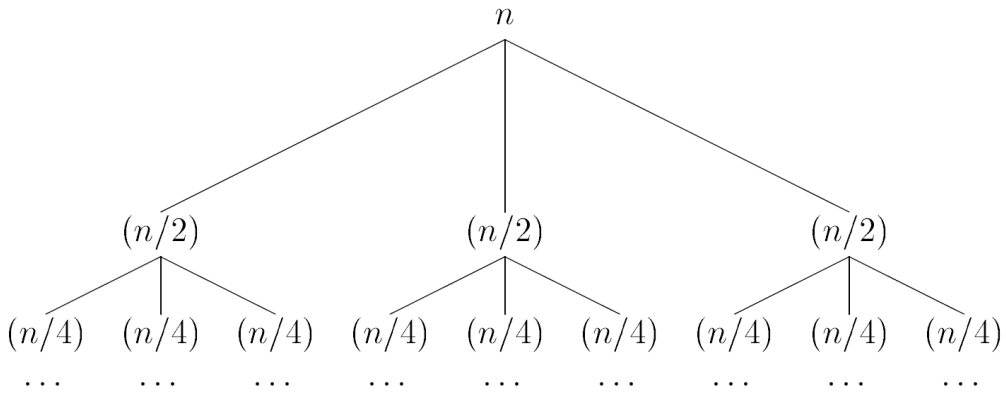
O(nlogn) master method case 2 (a = 3, b = 3, k = 1, p = 0)

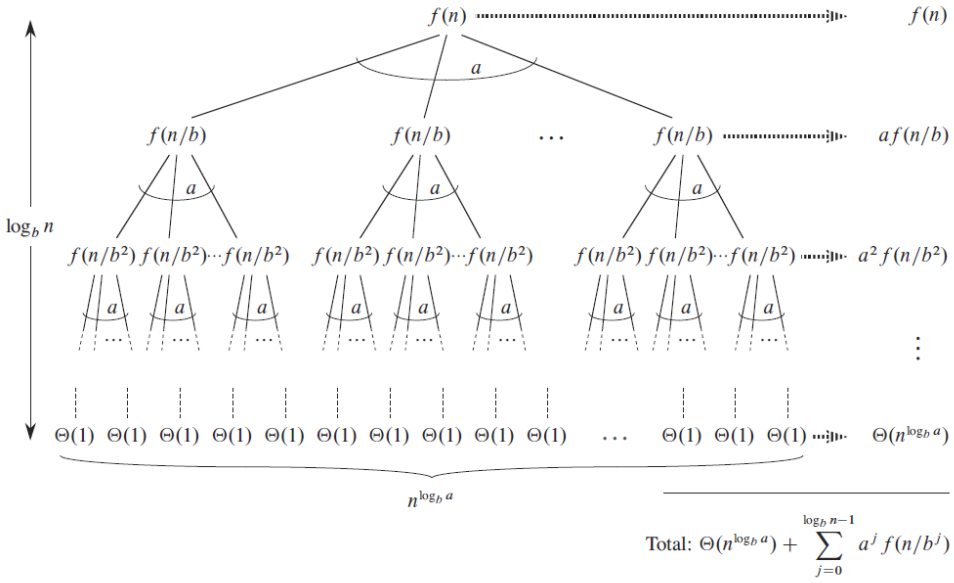



Q. Consider a recursion tree that looks like this:

1. What recurrence relation could generate this recursion tree? T(n) = 3T(n/2) + n
2. How many levels would there be in this tree, as a function of n? $\log_2 n$
3. How many leaves would be in this tree, as a function of n? $n^{\log_2 3}$
4. Solve the recurrence to obtain an asymptotic expression for T(n) as a function of n.

$O(n^{\log_2 3})$ master method case 1 (a = 3, b = 2, k = 1, p = 0)




1) The running time of the algorithm is the sum of running times for each statement executed; a statement that takes Ci steps to execute and executes *n* times will contribute Ci * *n* to the total running time. Compute T (*n*), the running time of the following algorithm on an input of *n* values. State an asymptotic upper bound using O-notation.

```
LinearSearch(A, key)
1   i ← 1
2   while i ≤ n and A[i] != key
3       do i++
4   if i ≤ n
5       then return true
6       else  return false
```

2) The running time of the algorithm is the sum of running times for each statement executed; a statement that takes Ci steps to execute and executes *n* times will contribute Ci * *n* to the total running time. Compute T (*n*), the running time of the following algorithm on an input of *n* values. State an asymptotic upper bound using O-notation.

**A()**:

```
int i, j, k, n;
for (i=1; i<=n; i++)
        for (j=1; j<=i^2; j++)
             for (k=1; k<=n/2; k++)
                  printf("Hello");
```

3) The running time of the algorithm is the sum of running times for each statement executed; a statement that takes Ci steps to execute and executes *n* times will contribute Ci * *n* to the total running time. Compute T (*n*), the running time of the following algorithm on an input of *n* values. State an asymptotic upper bound using O-notation.

- for an ordered array A, finds if x is in the array A[lo…hi]

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

```
if (lo > hi)
    return FALSE
mid ← ⌊(lo+hi)/2⌋
if x = A[mid]
    return TRUE
if ( x < A[mid] )
    BINARY-SEARCH (A, lo, mid-1, x)
if ( x > A[mid] )
    BINARY-SEARCH (A, mid+1, hi, x)
```

*Algorithm Loop(n)*

```
p=1
for i = 1 to 2n
p = p*i
```

*Algorithm Loop(n)*

```
        s=0
        for i = 1 to n
                for j = 1 to i
                        s = s + I
```

Algorithm Loop(n)

```
p=1
for i = 1 to n^2
        p = p*i
```

| | Best case | Worst case | In place | Comparisons | Exchange | |
|---|---|---|---|---|---|---|
| Insertion | O(n) | $O(n^2)$ | yes | $O(n^2)$ | $O(n^2)$ | |
| Bubble | $O(n^2)$ | $O(n^2)$ | yes | $O(n^2)$ | $O(n^2)$ | |
| Selection | $O(n^2)$ | $O(n^2)$ | yes | $O(n^2)$ | O(n) | |
| Merge | O(nlgn) | O(nlgn) | no | | | |
| Quick | O(nlgn) | $O(n^2)$ | yes | | | |

Q. What is the asymptotic expected time bound for quicksort?

Q. What is the asymptotic expected worst case time bound for quicksort?

For each of the following statements, indicate whether it is true or false. To get credit, you must give brief reasons for your answer.

T F InsertionSort's worst case running time is $\Theta(n^2)$
T F Both MergeSort and QuickSort have Θ(nlgn) best case running time.
T F The Master method can be used to solve the recurrence $T(n) = \sqrt{n}T(n/2) + n$, but not the recurrence $T(n) = 2T(n/\sqrt{n}) + n$.
T F Suppose that you write a program that frequently sorts arrays whose size varies between 5 and 10. You should do this using MergeSort instead of InsertionSort, since the running time of MergeSort is Θ(nlgn), while that of InsertionSort is O(n 2 )
T F QuickSort's running time depends on whether the partitioning is balanced or unbalanced. If the partitioning is balanced, running time is Θ(nlgn), however, when the partitioning is unbalanced, the running time is Θ(n 2 ).

The median of n elements can be found in O(n) time. What is the worst case time complexity of quick sort in which median is selected as pivot?

The (n/4)th smallest element is selected as pivot using O(n) time. What is the worst case time complexity of quick sort?

Problem: Sort a file of <u>huge records</u> with <u>tiny keys</u> → It means less comparison overhead but more exchange overhead
Example application: Reorganize your Video files
Which method to use?

A. merge sort, guaranteed to run in time ~NlgN
B. selection sort
C. bubble sort
D. a custom algorithm for huge records/tiny keys
E. insertion sort

Problem: sort a file that is already almost in order
Applications:

- Re-sort a huge database after a few changes
- Doublecheck that someone else sorted a file

Which sorting method to use?

A. Mergesort, guaranteed to run in time ~NlgN
B. Selection sort

C. Bubble sort
D. A custom algorithm for almost in-order files
E. Insertion sort

- Selection sort?
  - NO, always takes quadratic time
- Bubble sort?
  - NO, bad for some definitions of "almost in order"
  - Ex: B C D E F G H I J K L M N O P Q R S T U V W X Y Z A
- Insertion sort?
  - **YES**, takes linear time for most definitions of "almost in order"
- Mergesort or custom method?
  - Probably not: insertion sort simpler and faster
  - Takes less memory space

Find the recurrence that describes the running time of the above algorithm.
Solve the recurrence describing the running time of the above algorithm.

| Data Structure | Insert | Delete | Search | Space |
|---|---|---|---|---|
| Unsorted array | $O(1)$ | $O(1)$ | $O(n)$ | $O(n)$ |
| Sorted array | $O(n)$ | $O(n)$ | $O(\log n)$ | $O(n)$ |
| Unsorted linked list | $O(1)$ | $O(1)$ | $O(n)$ | $O(n)$ |
| Sorted linked list | $O(n)$ | $O(1)$ | $O(1)$ | $O(n)$ |
| Hash table | $O(1)$ | $O(1)$ | $O(1)$ | $O(n)$ |

**Your reference:**

$x = \log_b a$ is the exponent for $a = b^x$.

Natural log: $\ln a = \log_e a$
Binary log: $\lg a = \log_2 a$

$\lg^2 a = (\lg a)^2$
$\lg \lg a = \lg (\lg a)$

$$a = b^{\log_b a}$$
$$\log_c (ab) = \log_c a + \log_c b$$
$$\log_b a^n = n \log_b a$$
$$\log_b a = \frac{\log_c a}{\log_c b}$$
$$\log_b (1/a) = -\log_b a$$
$$\log_b a = \frac{1}{\log_a b}$$
$$a^{\log_b c} = c^{\log_b a}$$