Tuto 1

[T-2005-1-Tutorial 1.pdf]

1.

```
input sequence of numbers array n
input value v

for every number in array n
    if value at index i equals v
        return index i
    else
        increment i
    return null
```

2.

$\theta(n^3)$

3.

a. represents upper bound for a function to within a constant factor

b. when n => 4, A = O(B).
the algorithm B is smaller in size when input is small but is larger when input is big

c. Algorithm A. Algo B is better for smaller dataset but Algo A is less complex with more dataset

d. it does not effect my preferences. As A = 0(B)

e.
1. Input size. for small input sizes the runtime may not align with asymptotic analysis because analysis is made for large dataset.
2. Input sequence.

4.

a.

f(n) = 3n + 2
g(n) = n

f(n) <= cg(n)
3n + 2 <= c . n
3 + 2/n <= c

assume n -> infinity, 2/n -> 0. So 3 <= c
thus f(n) = O(g(n))

g(n) <= cf(n)
n <= c . 3n + 2
1 <= c (3+ 2/n)
1/(3 + 2/n) <= c

assume n -> infinity, 2/n -> 0. So 1/3 <= c
g(n) = O(f(n))

so both cases are valid

b.

f(n) = (n^2 - n)/2
g(n) = 6n

f(n) <= cg(n)
(n^2 - n)/2 <= c . 6n
(n^2 - n) <= c. 12n
n - 1 <= c. 12

assume n -> infinity, n - 1 -> infinity. So inequality is false
f(n) !=O(g(n))
thus g(n) = O(f(n))

c.

f(n) = n+2√n
g(n) = n^2

f(n) <= cg(n)
n+2√n <= c . n^2
1+ 2/n^-1/2 <= c . n

assume n-> infinity, $2/n^{-1/2} \to 0$.
$1 \le c \cdot n$
thus $f(n) = O(g(n))$

d.

$f(n) = n^2 + 3n + 4$
$g(n) = n^3$

$f(n) \le cg(n)$
$n^2 + 3n + 4 \le c \cdot n^3$
$1 + 3/n + 4/n^2 \le c \cdot n$

assume n -> infinity, $3/n + 4/n^2 \to 0$.
$1 \le c \cdot n$
thus $f(n) = O(g(n))$

5.

a. $O(n)$

b. $O(n^2)$

c. $O(n^{1/2})$

d.

```
outermost loop (i)

starting loop: n/2
end loop: n
increment: 1
total run:
end loop - starting loop
= n - n/2
= n/2
=> total run: n/2

outermost loop (i) complexity: O(n)
```

```
middle loop (j)
```

```
starting loop: n/2
end loop: n
increment: 1
total run:
end loop - starting loop
= n - n/2
= n/2
=> total run: n/2

middle loop (j) complexity: O(n)
```

```
innermost loop (k)

starting loop: 1
end loop: n
increment: k * 2 = 2^m
total run:
2^m >= n
m >= log2 n
=> total run: log2 n

innermost loop (k) complexity: O(log2 n)
```

total time complexity:
= O(n) *O(n)* O(log2 n)
= n^2 log2 n
=> O(n^2 log2 n)