HW #2

1.) a.) $\forall a \geq 1, 2^{an} \in O(2^n)$

Disprove: $\exists c \geq 1$, $\exists n_0 \geq 1$ such that $\forall n \geq n_0$ for $\forall a \geq 1$

$$2^{an} \leq c \cdot 2^n$$

$$\frac{2^{an}}{2^n} \leq c$$

$$2^{an-n} \leq c$$

for $a > 1$ $\lim_{n \to \infty} 2^{an-n} = \infty$

→ meaning $c \geq \infty$ as n rises

So $\forall a \geq 1$, 2^{an} & $O(2^n)$

b) b.) $2^{f(n)} \in O(2^{g(n)})$ implies $f(n) \in O(g(n))$

$\rightarrow \quad 2^{f(n)} \leq c \cdot 2^{g(n)}$

$f(n) \leq \log_2 c + g(n)$

$f(n) \leq g(n) + c_1 \leq c_2 \cdot g(n)$

Counter example:

$f(n) = 1 + \frac{1}{n} \quad \rightarrow \quad 2^{1+\frac{1}{n}} = 2 \cdot 2^{\frac{1}{n}} \quad c = 2$

$g(n) = \frac{1}{n} \quad \rightarrow \quad 1 + \frac{1}{n} \leq \frac{1}{n} \cdot c$

$\lim_{n \to \infty} 1 + \frac{1}{n} = 1 \qquad \lim_{n \to \infty} \frac{1}{n} \cdot c = 0$ so c doesn't exist

So $2^{f(n)} \in O(2^{g(n)})$

does not imply

$f(n) \in O(g(n))$

1.) c.) $f(n) \in O(g(n))$ implies $2^{f(n)} \in O(2^{g(n)})$

so $f(n) \leq c \cdot g(n) \xrightarrow{\text{implies}} 2^{f(n)} \leq c \cdot 2^{g(n)}$

$2^{f(n)} \leq 2^{c \cdot g(n)}$ not same ↗

Counter example:

$f(n) = 4n$
$g(n) = n$

$f(n) \leq 4 \cdot g(n)$ $c = 4$

$2^{4n} \leq c \cdot 2^n$

$2^{3n} \leq c$ $\lim_{n \to \infty} 2^{3n} = \infty$

so c doesn't exist

$f(n) \in O(g(n))$
does not imply
$2^{f(n)} \in O(2^{g(n)})$

2.) a.)

for $i = 1 \ldots = n-1$ $= n-1$

for $j = i \ldots = n$ $- n-i+1$

for $k = 1 \ldots = j$ $- j$ ops

exe $- C$

$$C \cdot \sum_{i=1}^{n-1} \sum_{j=i}^{n} \sum_{k=1}^{j} 1 = C \cdot \sum_{i=1}^{n-1} \sum_{j=1}^{n} j \qquad 1+n+3+4 \cdots +n$$

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

$$\in O(n^3)$$

2.)b.) i=0 j=n-1) 2 ops

while (i<j)) depends on array layout

while (a[i] < k) i++

while (a[j] > k) j--

if (i<j) swap i,j] 4 ops

] always preform n ops in total

a_max > k > a_min
else crashes

i 10 | 11 k=9 2 12 3 j

Worst case:
Swap increment swap in swap inc

$n + \frac{n}{2} \cdot 4 + 2 = 3n + 2$

$n + 4c + 2$

1
#of i<j occurs based on order of array

$\in O(n)$

3.) lastest_zero(A) → i: (assuming A is at least length 1)

```
if A[0] == 1:
    return -1

if A[length] == 0
    return length

if A[length/2] == 0:
    return length/2
        + latest_zero(A.sub(length/2 ... length)   not inclusive

else
    return latest_zero(A.sub(0 ... length/2)
```

length	# of ops
1	c
2	c
3	c
4	c
4	c
⋮	⋮
k	c

$len = n/2^{k-1}$ $\quad \frac{n}{2^{k-1}} = 1 \quad n = 2^{k-1}$

$\log n = k-1 \quad k = \log_2 n + 1$

$\in O(\log n)$

4.) a.) delete(i):

replace A[i] with A[n-1]
empty flag A[n-1]

if i not 0 and A[i] < A[$\frac{i-1}{2}$]
heapify-up()
else
heapify-down()

$\in O(\log n)$

b.) delete(v):

for i in 0...n:
if A[i] == v;

delete(i)
return

$\in O(n \log n)$

5a) almost-sorted(A, k) → S:

```
PriorityQueue pq  # MinHeap based
push first k+1 elements of A to pq

for i in k+1..n:
    append pq.extract to S
    push ith element in A to pq

for remaining elements in pq:
    append pq.extract to S
```

Guarantees first to be correct

Extracts all but k+1 elements guaranteed in order

Extracts remaining k+1 elements

Initial push
$\log(1) + \log(2) + \log(3) \ldots + \log(k+1) = \log((k+1)!)$

Extract & push
$n-k-1$ times
$2 \cdot \log(k+1)$ $\leq 2n\log(k+1) - 2k\log(k+1) + 2\log(k+1)$ (largest growth (so bounded by))

Extract
$\log(k+1) + \log(k) \ldots + \log(2) + \log(1) \leq \log((k+1)!)$

$\in O(n \log k)$