1. (a) For each of the following functions, determine whether it is $O(n^2)$, $\Omega(n^2)$, or both.
You need only check the appropriate box or boxes; you do not need to provide proof or justification. Note that we are *not* asking you to provide the "best" O-notation/Ω-notation; merely to describe it with one or both of the choices provided.

✓

$a(n) = n(\log n)^{10}$	■ $O(n^2)$	☐ $\Omega(n^2)$
$b(n) = n^2 + n\sqrt{n}$	■ $O(n^2)$	■ $\Omega(n^2)$
$c(n) = n^{2.5}$	☐ $O(n^2)$	■ $\Omega(n^2)$

(b) Give a closed form for each of the following recurrences:

i. $T(n) = 4T(n/2) + 3$ $\quad a=4 \quad b=2 \quad f(n)=3 \quad \log_b a = \log_2 4 = 2$

$\boxed{T(n) \text{ is } \Theta(n^2)}$

✓

ii. $T(n) = 2T(n/2) + n$ $\quad a=2 \quad b=2 \quad f(n)=n \quad \log_b a = \log_2 2 = 1$

$\boxed{T(n) \text{ is } \Theta(n \log n)}$

iii. $T(n) = 3T(n/3) + n^2$ $\quad a=3 \quad b=3 \quad f(n)=n^2 \quad \log_b a = \log_3 3 = 1$

$\boxed{T(n) \text{ is } \Theta(n^2)}$

(c) The KNAPSACK problem has input of n items, each of which has a positive integer weight w_i and a real number value v_i. The goal is to select a subset of items whose total weight is at most some integer W (also part of the input) and whose total value is as large as possible. Which of the following would be a polynomial runtime for an algorithm that solves KNAPSACK?

-3

i. $O(n \log W)$
ii. $O(W \log n)$
iii. $O(nW)$
iv. $O(n \max_i w_i)$

(d) Recall that a heap is commonly represented as an array, and that a string is an array of characters. Which of the following strings' array representation is a max-heap, using alphabetical order ($z > y > x$ etc) for comparisons?

✓

i. grading
ii. pinhead
iii. oranges
iv. michael

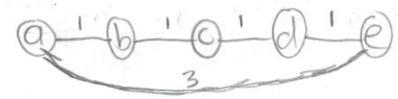
c d e z g i

2. For each of the following questions, decide whether it is true or false by checking the appropriate box. If it is true, give a short explanation. If it is false, give a short counter-example.
Suppose we have a connected undirected graph $G = (V, E)$ with positive costs c_e on the edges, and a distinguished vertex s. The costs may or may not be distinct. We also have T, a minimum spanning tree of G, and P, a tree of the shortest paths in G from s to all other vertices.
Now suppose we create G', an exact copy of G's vertices and edges, but we replace each cost c_e with c_e^2.

(a) P is still the shortest path tree of G' from s: True ☐ False ■.

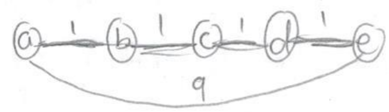
Counter example:

G:



shortest path from a to e goes directly to e, with cost = 3

G':



shortest path from a to e goes through b, c, and d with cost = 4

these are different shortest path trees!

(b) T is still a minimum spanning tree of G': True ■ False ☐.

When finding the minimum spanning tree, we only look at the relative costs of each edge individually. Squaring each edge will still keep the edges in the same relative ordering. Although the total cost may change, the same edges will still be used in G and G'.

3. Suppose you have an array A of n integers. You wish to produce a two-dimensional array B that is $n \times n$, where $B[i][j]$ (for $i < j$) holds the sum of elements $A[i..j]$ (inclusive). $B[i][j]$, for $i \geq j$, is undefined - you may use those spots in B however you want (or leave garbage/defaults there). Give an efficient algorithm that will produce array B; for full credit, your algorithm should run in time $\Theta(n^2)$.

```
Sum = 0;
for j from ∅...n
{
    Sum += A[j]
    if j > ∅
    {
        B[∅][j] = sum
    }
    Count = 0
    for i from 1 to i < j
    {
        Count += A[i-1]
        B[i][j] = (Sum - count)
    }
}
```

Code unclear

A= 1 2 3 4

	0	1	2	3
0	X	1	3	
1				
2				
3				

Sum=3

j=2

i=1