Homework 3

3.3 2. Yes, you can design a more efficient algorithm than the one based on the brute-force strategy.

One way to do it is to first sort the numbers in ascending order which will take $O(n \log n)$ time. Next compute the differences between adjacent numbers in sorted list, $O(n)$. Finally find the smallest difference, $O(n)$. Thus run time equals $O(n \log n) + O(n) + O(n)$ which becomes $O(n \log n)$

4a. $d_m(p_1, p_2) \geq 0$ i) $d_m(p_1, p_2) = 0$ if and only if $p_1 = p_2$

$0 = |x_1 - x_2| + |y_1 - y_2|$

Thus, $x_1 = x_2$ and $y_1 = y_2$ $\therefore$ P_1 and P_2 are same,

ii) $d_m(P_1, P_2) = |x_1 - x_2| + |y_1 - y_2|$

$= |x_2 - x_1| + |y_2 - y_1|$

$= d_m(P_2, P_1)$

$\therefore d_m(p_1, p_2) = d_m(p_2, p_1)$

iii) $|x_1 - x_2| + |y_1 - y_2| \leq |(x_1 - x_3) + (x_3 - x_2)| + |(y_1 - y_3) + (y_3 - y_2)|$

$\leq |x_1 - x_3| + |x_3 - x_2| + |y_1 - y_3| + |y_3 - y_2|$

$\leq |x_1 - x_3| + |y_1 - y_2| + |x_3 - x_2| + |y_3 - y_2|$

$\leq d_m(p_1, p_2) + d_m(p_3, p_2)$

b. $|(x-0)| + |(y-0)| = 1$

$|x| + |y| = 1$

$\pm x \pm y = 1$

boundary of square $(1,0)$, $(0,1)$

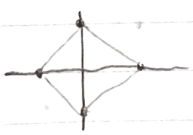
$(-1,0)$, $(0,-1)$

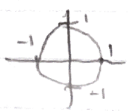
$\sqrt{(x-0)^2 + (y-0)^2}$

$x^2 + y^2 = 1$

circle around $(0,0)$

w/ radius 1





c False, the two metrics will give different pair of points.

3.4 1a total number of tours $\frac{1}{2}(n-1)!$

$A(n) = n \times$ total number of tours — since n cities, n additions for every tour

$= n \times \frac{1}{2}(n-1)!$

$= \frac{1}{2}n!$

∴ efficiency class of given exhaustive search is $\Theta(n!)$

bi. total number of addition $\frac{1}{2}n!$ n cities

$10,000,000,000$ additions = 1 sec

1 addition = $\frac{1}{10^9}$ sec

$\frac{1}{2}n! \, 10^{-9} \leq t$

$n! \leq 2(10^9)t$

1 hour to sec : 60 minute × 60 secs

$= 3600$

$= 3.6 \times 10^3$ sec

$n! \leq 2 \times 10^9 \times 3.6 \times 10^3$ s

$\leq 7.2 \times 10^{12}$

$\approx 15!$

The maximum number of cities the problem can solve in 1 hour is **15**

bii 24 hours = 24 × 60 minutes × 60 sec

$= 86400$

$= 8.64 \times 10^4$ sec

$n! \leq 2 \times 10^9 \times 8.64 \times 10^4$ sec

$\leq 17.28 \times 10^{13}$

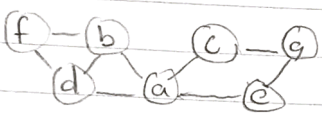
$\approx 16!$

The maximum number of cities the problem can solve in 24 hours is **16**.

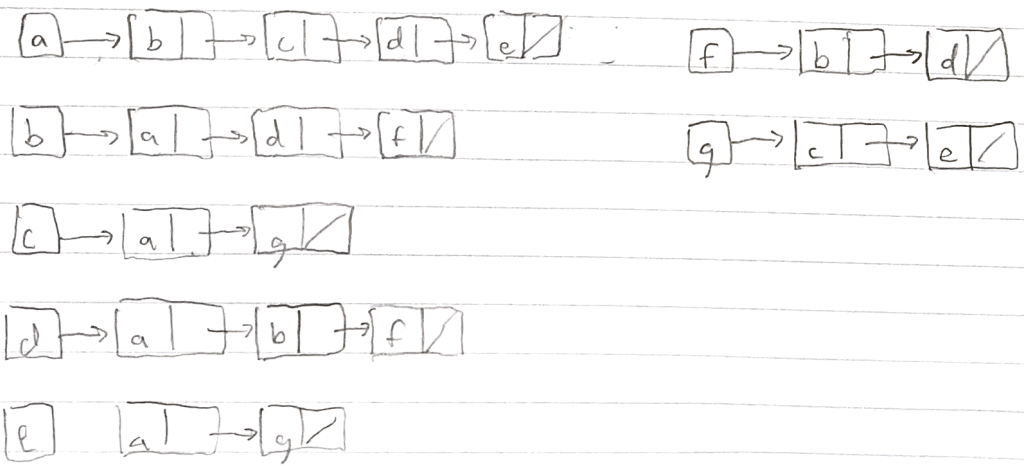
6. Algorithm Partition(n)

```
Sum ← 0
for i ← 1 to n do
    Sum ← Sum + a[i]      // a is an array that contains n positive integers
if Sum mod 2 = 1
    exit
else
    generate subsets until either subset whose elements' sum
    is Sum/2 or no more elements left to subset.
```

3.5 1a.



	a	b	c	d	e	f	g
a	0	1	1	1	1	0	0
b	1	0	0	1	0	1	0
c	1	0	0	0	0	0	1
d	1	1	0	0	0	1	0
e	1	0	0	0	0	0	1
f	0	1	0	1	0	0	0
g	0	0	1	0	1	0	0

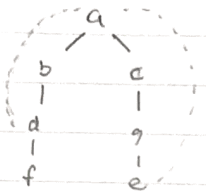


b.

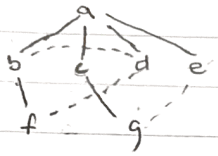
$a \to b \to d \to f \to c \to g \to e$

number on left is push order

number on right is pop order



4.



order $a \to b \to c \to d \to e \to f \to g$

front

g f e d c b a

8a. Algorithm DFSbipartite(G)

// Input: graph $G = \{V, E\}$

// Output: returns a bool, true if graph is bipartite, false if not

checked[0 to n-1] ← 0

color[0 to n-1] ← 0

flag ← 0

For every vertex v in V do

if checked[v] = 0

dfs(v)

dfs(v)

checked[v] ← 1

if color[v] = 0

color[v] ← 1

for each vertex w in V adjacent to v do

if checked[w] = 0

if color[v] = 1

color[w] = 2

else

color[v] = 1

dfs(w)

if checked[w] = 1 and color[w] = color[v]

flag = 1

Algorithm BFS Bipartite (G)
// Input: a graph G = {V, E}
// Output: if flag = -1 not bipartite else it is bipartite
flag ← 0
checked [0 to n-1] ← 0
color [0 to n-1] ← 0
for each vertex v in V do
    if checked[v] = 0
        bfs(v)
bfs(v)
checked[v] ← 1
color[v] ← 1
enqueue v
while queue is not empty do
    for each vertex w in V adjacent to front vertex v do
        if checked[w] = 0
            checked[w] ← 1
            color[w] ← 2
            enqueue w
        if checked[w] = 1 and color[v] = color[w]
            flag = -1
    dequeue w

4.1 4. Algorithm Powerset ({a1, .., an}, n)
// Input A set of n elements and size of set
// Output: Power set of input set

if n = 0
    return {}
T(n-1) = Powerset ({a1, .. an-1}, n-1}
T(n) = {an} ∪ {an T(n-1)}
return T(n)

6. Algorithm Team Ordering (n)
if n = 1
- then puzzle is done being solved

if n > 1
- solve recursively for (n-1) teams
- scan the array of (n-1) teams
- insert team which isn't added in group right before the first team on the array which lost to it in the contest
- if such team is not found (not yet on the array, they lost to every team), add that team at the end (last) of the array

7. E | X A M P L E
E X | A M P L E
A E X | M P L E
A E M X | P L E
A E M P X | L E
A E L M P X | E
A E E L M P X

8a. Sentinel: Any value less than or equal to every element in the array

b. Yes the sentinel version is in the same efficiency class as the original version

$$C_{worst}(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i-1-0+1 = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \in \Theta(n^2)$$

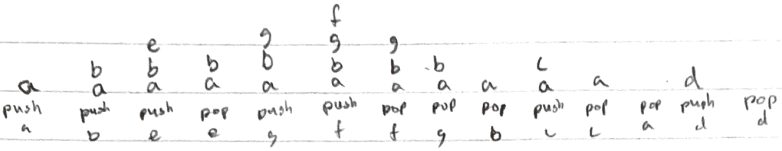
12a. S H E L L S O R T I S U S E F U L
L H E L L E F R S I S U S S O U T
L E E L L H F R S I O U S S S U T
E E F H L I L L O R S S S U S U T
E E F H I L L L O R S S S S T U U

b. Yes, shellsort is a stable sorting algorithm since the order of same elements remains when sorted due to swapping only if one is less than or greater than, never equal to. Also shellsort is an extension of insertion sort which is also stable, thus shellsort has to be stable

c.

```
import java.util.Arrays;

public class Shellsort {
    char temp; int j;
    void sort(char arr[]) {
        int size = arr.length;
        for(int k = size/2 ; k > 0; k/=2) {
            for(int i = k; i < size; i+=1) {
                temp = arr[i];
                for(j = i; j >= k && arr[j-k] > temp; j-=k) {
                    arr[j] = arr[j-k];
                    arr[j] = temp;
                }
            }
        }
    }

    public static void main(String args[]) {

        char arr[] = {'S','H','E','L','L','S','O','R','T','I','S','U','S','E','F','U','L'};

        Shellsort x = new Shellsort();
        x.sort(arr);
        System.out.println(Arrays.toString(arr));

    }
```

4.2 1a.

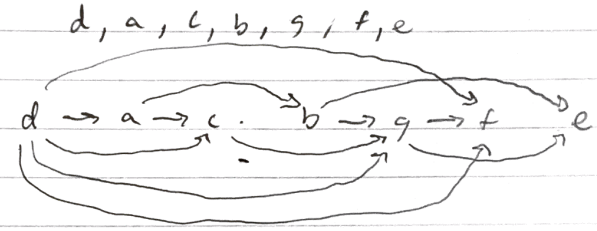


e_1 f_2 g_3 b_4 a_6 c_5 d_7 number is popping order

pop order: e, f, g, b, c, a, d

topologically sorted list is reversed stack contents thus

d, a, c, b, g, f, e



3a The time efficiency of the DFS-based algorithm for topological sorting is $\Theta(|V|^2)$ for the adjacency matrix and $O(|V|+|E|)$ for adjacency linked lists representation

b One can modify the DFS-based algorithm to avoid reversing the vertex ordering generated by DFS by filling the array of length $|V|$ with vertices being popped off the DFS traversal's stack right to left.