Name: _____

Directions: **Work only on this sheet** (on both sides, if needed). DO NOT turn in any supplementary sheets of paper. There is actually plenty of room for your answers, as long as you organize yourself BEFORE starting writing. When appropriate, SHOW YOUR WORK.

1. (20) Fill in the blanks below. We have a graph with adjacency matrix **adj**, so that **adj[i][j]** is 1 or 0, indicating the presence or absence of an edge from i to j. The function returns a list of vertices that are immediate neighbors of i.

```
def find1(adj,i):  # find 1-hop neighbors of i, excluding i
   # get number of vertices in graph
   nverts = ____________________________
   nghbrs = []
   for j in range(nverts):
      if ________________________________________:
         ________________________________________
   return nghbrs
```

2. (20) The following program is run with the command line

```
python SplitIn2.py f n f1 f2
```

It splits the file **f** into files **f1** and **f2**, where **f1** consists of the first **n** bytes of **f** and **f2** is the remainder. Fill the blanks. Make sure it is platform-independent.

```
import sys, os

def main():
   f = open(_________________________________)
   n = int(sys.argv[2])
   f1 = open(_________________________________)
   f2 = open(_________________________________)
   firstn = f.read(_____________________________________)
   _________________________________
   f1.close()
   nbytesremaining = _____________________________ - n
   therest = ____________________________
   f2.write(therest)
   f2.close()

if __name__ == '__main__':
   main()
```

3. The following code was meant to produce an **n**x**n** matrix of 1s:

```
all1s = n*[1]  # a row of 1s
x = n*all1s  # n rows of 1s
```

(a) (10) The above code fails to meet its goal. Show how to fix this by modifying the first statement.

(b) (10) Explain why is the above code, even after fixing, probably is not a good idea.

4. (20) The function **dsort()** below will "sort" a dictionary **d**, returning a list of two-element lists. Each of the latter is key-value pair from **d**. Sorting is according to keys. For example:

```
>>> dsort({'abc':12,'sailing':'away','aa':15})
[['aa', 15], ['abc', 12], ['sailing', 'away']]
```

Fill in the blanks:

```
def dsort(d):
   dk = ___________________________________
   dk.sort()
   return map(___________________________________)
```

5. (20) Fill in the blanks in **rgmx()**, a function that returns both the maximum value in a list and the index for that value (or *some* index, in the case of ties). For example, **rgmx([5,12,13,8])** will return [13,2].

```
def rgmx(x):
   xandi = ___________(x,____________________)
   return max(xandi)
```

Solutions:

1.

```
len(adj)
adk[i][j] == 1 and i != j
nghbrs.append(j)
```

2.

```
sys.argv[1],'rb'
sys.argv[3],'w'
sys.argv[4],'w'
n
f1.write(firstn)
os.path.getsize(f.name)
f.read(nbytesremaining)
```

A better way would be to bypass computing **nbytesremaining** and then simply call **f.read()**.

3.a

```
add1s = [n*[1]]
```

3.b The problem is that you would get **n** copies of the object pointed to by the reference **all1s**. If you then changed one element of the matrix, the whole column would change.

4.

```
d.keys()
lambda z: [z,d[z]],dk
```

5.

```
zip
range(len(x))
```