Task 1.1

```
def split(L, low, high):
    mid = (low + high) // 2
    pivot = L[mid]
    L[low], L[mid] = L[mid], L[low]
    left = low + 1
    right = high

    while left <= right:
        while left <= right and L[left] <= pivot:
            left += 1

        while L[right] > pivot:
            right -= 1

        if left < right:
            L[left], L[right] = L[right], L[left]

    pos = right
    L[low] = L[right]
    L[pos] = pivot


    return pos


def quicksort(L, low, high):
    if low < high:
        pivot = split(L, low, high)
        quicksort(L, low, pivot - 1)
        quicksort(L, pivot + 1, high)

L = []
with open('INTEGERS.txt', 'r') as f:
    for line in f:
        L.append(int(line.strip()))
quicksort(L, 0, len(L) - 1)

filename = 'SORTED.txt'

with open(filename, 'w') as g:
    for item in L:
```

```
        g.write(str(item) + '\n')
```

Task 1.2

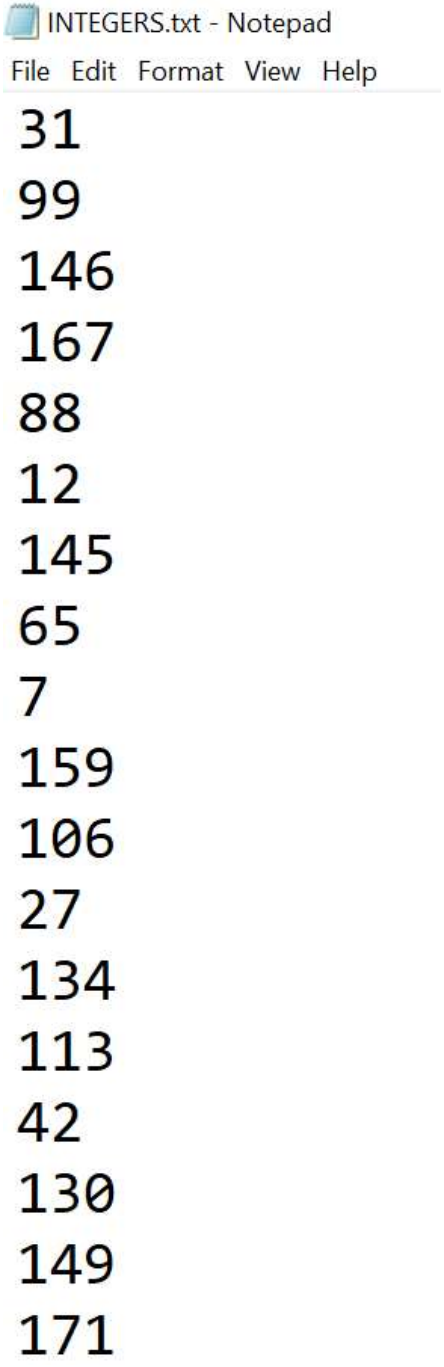
```
def BinarySearch(L, target):

    count = 0
    low = 0
    high = len(L) - 1
    found = False

    while not found and low <= high:
        mid = (low + high) // 2
        count += 1
        if L[mid] == target:
            found = True
        elif L[mid] < target:
            low = mid + 1
        else:
            high = mid - 1


    if found:
        print(target, 'is found')
    else:
        print(target, "is not found!")

    return count
```

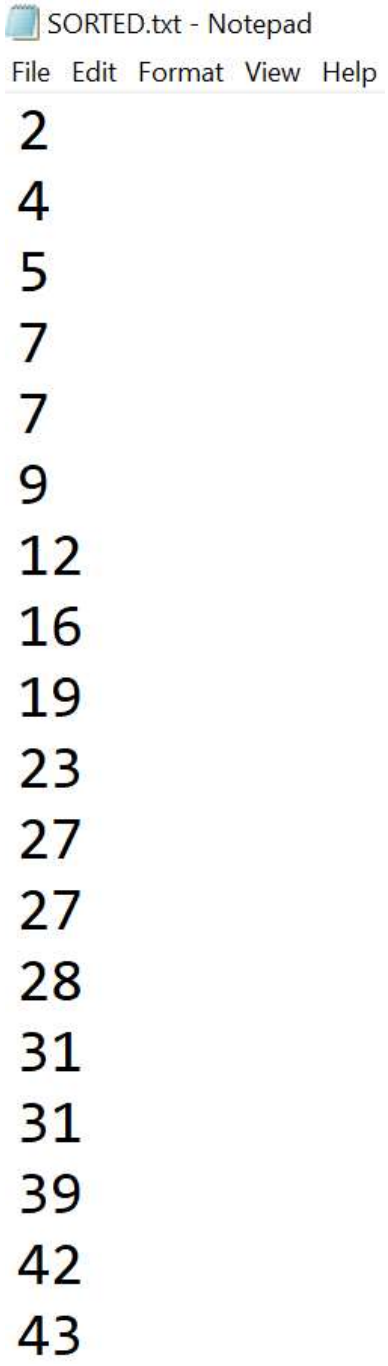
doesn't matter whether it's the low or high center. All values are checked anyways

Task 1.3

```
import random
L = []
with open('SORTED.txt', 'r') as f:
    for line in f:
        L.append(int(line.strip()))

avg = 0

for i in range(50):
    target = random.randint(1, 200)
    avg += BinarySearch(L, target)

print("Average comparison:", avg / 50)
```

Partial Screenshot of Text Files





Task2_1.sql [2M]

```
Create database ServiceLog.db

CREATE TABLE "Log" (
      "LogID"      INTEGER,
      "Sender"     TEXT,
      "AccessDate" TEXT,
      "Status"     INTEGER,
      "AppType"    TEXT,
      PRIMARY KEY("LogID" AUTOINCREMENT)
);
```

Task2_2.py [8M]

```
import sqlite3

class ServiceRecord:
    def __init__(self, sender=None, accessdate=None,status=None,
            app=None):
        self._sender = sender
        self._accessdate = accessdate
        self._status = status
        self._apptype = app

    def getSender(self):
        return self._sender

    def getAccessDate(self):
        return self._accessdate

    def getStatus(self):
        return self._status

    def isSuccess(self):
        if self._status==200:
            return True
        else:
            return False

    def getAppType(self):
        return self._apptype
```

Task2_2.py

```
#method to insert data into People table
def insertService(service):
    connection = sqlite3.connect("ServiceLog.db")
    connection.execute("INSERT INTO Log(Sender,AccessDate,Status,
AppType) VALUES (?,?,?, ?)",
      (service.getSender(),   service.getAccessDate(),
      service.getStatus(), service.getAppType()))
    connection.commit()
    connection.close()
```

```
def main():
    with open("LOG.txt","r") as fobj:
        line = fobj.readline()
        while line !='':
            record=line.split(" ")
            #differentiate between IP and phone
            if record[0].find(".")>0:
                accessdate = record[1].strip()
                srv = ServiceRecord(record[0].strip(), accessdate,
                        record[2].strip(), record[3].strip())
            else:
                accessdate = record[1].strip()
                srv = ServiceRecord(record[0].strip(), accessdate,
                        record[2].strip())
            insertService(srv)
            line = fobj.readline()

if __name__ == "__main__":
   main()   # stuff only to run when not called via 'import' here
```

Task2_3.py[4M]

```
from  Task2_2 import ServiceRecord

class  AppServiceRecord(ServiceRecord):
    def __init__(self,sender=None, accessdate=None,status=None,
                 app=None):
        super().__init__(sender, accessdate, status, app)

    def getAppType(self):
        if self._apptype=='WA':
            return 'WHATSAPP'
        elif self._apptype=='FB':
            return 'FACEBOOK MESSENGER'


    def getSuccess(self):
        if self.isSuccess():
            return 'SUCCESS'
        else:
            return 'FAILED'

class  SmsServiceRecord(ServiceRecord):
    def __init__(self,sender=None, accessdate=None,status=None,
                 app=None):
        super().__init__(sender, accessdate, status, app)

    def getAppType(self):
        return 'SHORT MESSAGE SERVICE'

    def getSuccess(self):
        if self.isSuccess():
            return 'SUCCESS'
        else:
            return 'FAILED'
```

Task2_4.py [7M]

```
import flask, os, sqlite3
from flask import render_template, request, redirect, url_for

from  Task2_3 import AppServiceRecord
from  Task2_3 import SmsServiceRecord

app = flask.Flask(__name__,  static_folder='./static',
template_folder='./templates')


@app.route('/', methods=['GET'])
def index():
    db = sqlite3.connect('ServiceLog.db')
    db.row_factory = sqlite3.Row
    cursor = db.execute('SELECT Sender, AccessDate, Status, AppType
                         FROM Log ')
    all_rows = cursor.fetchall()
    cursor.close()
    db.close()
    listx=[]
    for row in all_rows:
        if row[0].find(".")>0:    #differentiate between IP and phone
            srv = AppServiceRecord(row[0], row[1], row[2], row[3])
        else:
            srv = SmsServiceRecord(row[0], row[1], row[2], row[3])
        listx.append(srv)
    return render_template('index.html', results=listx)


if __name__ == '__main__':
    app.run()
```

Template [2M]

```
<!DOCTYPE html>
<html>
<head><title>Service Log</title>
<link rel="stylesheet" type="text/css"
href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
<p>Service Log</p>
<table>
<tr><th>Sender</th><th>Access Date</th><th>APP
Type</th><th>Status</th></tr>
{% if results|length > 0 %}
      {% for item in results  %}
    <tr>
        <td>{{ item.getSender() }}</td>
            <td>{{ item.getAccessDate() }}</td>
            <td>{{ item.getAppType() }}</td>
            <td>{{ item.getSuccess() }}</td>

    </tr>
      {% endfor %}
```

```
{%else%}
    <tr>
        <td colspan="2">No logs</td>
    </tr>
{%endif%}
</table>
</body>
</html>
```

Generated Html [1M]

Service Log

Sender	Access Date	APP Type	Status
54.36.149.41	22/Jan/2021	WHATSAPP	SUCCESS
188.226.164.216	22/Jan/2021	FACEBOOK MESSENGER	FAILED
92783423	22/Jan/2021	SHORT MESSAGE SERVICE	SUCCESS
188.226.164.216	23/Jan/2021	FACEBOOK MESSENGER	FAILED
88188293	23/Jan/2021	SHORT MESSAGE SERVICE	FAILED

```
#Task 3.1
```

```
class Node:
    def __init__(self, next = 0):
        self.Data = ''
        self.Priority = ''
        self.Next = next

    def setData(self, data):
        self.Data = data

    def setPriority(self, priority):
        self.Priority = priority

    def setNext(self, next):
        self.Next =  next

    def getData(self):
        return self.Data

    def getPriority(self):
        return self.Priority

    def getNext(self):
        return self.Next

    def display(self):
        print (f'{self.Data:<10}{self.Priority:<10}{self.Next}')


class PQueue:
    SIZE = 10

    def Initialise(self):
        self.ThisPQueue = [None] * (self.SIZE+1)
        for i in range(1, self.SIZE):
            self.ThisPQueue[i] = Node(i+1)
        self.ThisPQueue[self.SIZE] = Node(0)

        self.Front = 0
        self.Rear = 0
        self.NextFree = 1

    def IsEmpty(self):
        return (self.Front == 0)
```

```
def IsFull(self):
    return (self.NextFree == 0)
def PQInsert(self, NewItem, Priority):
    if self.IsFull():
        print ('Queue is full!')
    else:
        idx = self.NextFree
        self.NextFree = self.ThisPQueue[idx].getNext()

        self.ThisPQueue[idx].setData(NewItem)
        self.ThisPQueue[idx].setPriority(Priority)
        self.ThisPQueue[idx].setNext(0)

        if self.IsEmpty():
            self.Front = idx
        else:
            self.ThisPQueue[self.Rear].setNext(idx)

        self.Rear = idx


def PQDelete(self):
    if self.IsEmpty():
        print ('Queue is empty!')
        return ''
    else:
        # find position of the highest priority node
        pos = self.HPriorityPos()

        # remove the highest priority node at pos
        delNode = self.Delete(pos)
        return delNode.getData()
```

```
def HPriorityPos(self):
# find position of the node with highest priority
    pos = 1
    count = 1he
    hPriority = self.ThisPQueue[self.Front].getPriority()

    idx = self.ThisPQueue[self.Front].getNext()
    while idx != 0:
        count += 1
        priority = self.ThisPQueue[idx].getPriority()
        if priority < hPriority:
            pos = count
            hPriority = priority
        idx = self.ThisPQueue[idx].getNext()

    return pos

def Delete(self, pos):
# remove the highest priority node at pos
    prev = 0
    curr = self.Front

    for i in range(pos-1):
        prev = curr
        curr = self.ThisPQueue[curr].getNext()

    delNode = self.ThisPQueue[curr]

    if curr == self.Front:
        self.Front = delNode.getNext()
        if self.Front == 0:
            self.Rear = 0
    elif curr == self.Rear:
        self.ThisPQueue[prev].setNext(0)
        self.Rear = prev
    else:
        self.ThisPQueue[prev].setNext(delNode.getNext())

    self.ThisPQueue[curr] = Node(self.NextFree)
    self.NextFree = curr

    return delNode
```

```
    def DisplayPQueue(self):
        print (f'Front: {self.Front}')
        print (f'Rear: {self.Rear}')
        print (f'Next Free: {self.NextFree}')
        print()
        print (f"{'Index':<8}{'Data':<10}{'Priority':<10}{'Next'}")
        for i in range(1, len(self.ThisPQueue)):
            node = self.ThisPQueue[i]
            print (f"{i:<8}", end = '')
            node.display()
        print()


def main():
    # Task 3.2
    print ('Task 3.2')

    q = PQueue()
    q.Initialise()

    with open("PATIENTS.txt", 'r') as f:
        patients = f.readlines()

    for line in patients:
        name, priority = line.rstrip().split(",")
        priority = int(priority)
        q.PQInsert(name, priority)

    q.DisplayPQueue()

    # Task 3.3
    print ('Task 3.3')

    for i in range(2):
        data = q.PQDelete()

    q.PQInsert("Carol", 4)

    for i in range(2):
        data = q.PQDelete()

    q.DisplayPQueue()

main()
```

Task 3.2

Front: 1
Rear: 6
Next Free: 7

Index	Data	Priority	Next
1	George	2	2
2	Jane	1	3
3	Sandra	4	4
4	Bill	3	5
5	Dave	5	6
6	Bob	1	0
7			8
8			9
9			10
10			0

Task 3.3

Front: 3
Rear: 6
Next Free: 4

Index	Data	Priority	Next
1			2
2			7
3	Sandra	4	5
4			1
5	Dave	5	6
6	Carol	4	0
7			8
8			9
9			10
10			0

```python
from random import randint
# Task 4.1  [3]
board = []
for i in range(4):
    board.append (['.']* 4)

row = randint(0, 4-1)
col = randint(0, 4-1)
board[row][col] = 'T'



# Task 4.2  [2]
def displayBoard(board):
    for row in range(4):
        for col in range(4):
            print (board[row][col], end=' ')
        print()



# Task 4.3  [4]
def getPlayerMove(board, player):

    valid = False
    while not valid:

        row = int(input('Enter row (1-4): '))
        while not (1 <= row <= 4):
            row = int(input('Error! Enter row (1-4): '))

        col = int(input('Enter col (1-4): '))
        while not (1 <= col <= 4):
            col = int(input('Error! Enter col (1-4): '))



         row -= 1 # Zero-index
        col -= 1

        if board[row][col] == '.':
            valid = True
        else:
            print("Error! Cell is already taken.")

    board[row][col] = player

    return (row, col)
```

```python
# Task 4.4  [5]
def checkWin(board, move, player):
    row, col = move

    # check row
    win = True
    for c in range(4):
        if board[row][c] not in (player, 'T'):
            win = False
    if win: return True

    # check col
    win = True
    for r in range(4):
        if board[r][col] not in (player, 'T'):
            win = False
    if win: return True

    # check diagonal from left top to right bottom
    if row == col:
        win = True
        for i in range(4):
            if board[i][i] not in (player, 'T'):
                win = False
        if win: return True


    # check diagonal from left bottom to right top
    if row + col == 3:
        win = True
        for c in range(4):
            if board[3-c][c] not in (player, 'T'):
                win = False
        if win: return True


    return False



def getJokerT(board):
    row = randint(0, 4-1)
    col = randint(0, 4-1)
    board[row][col] = 'T'


def hasEmptySquare(board):
    for row in range(4):
        for col in range(4):
            if board[row][col] == '.':
                return True
    return False
```

```
# Task 4.5  [7]
def main():

    board = [['.']* 4 for i in range(4)]
    getJokerT(board)
    displayBoard(board)

    pX = 'X'
    pO = 'O'
    player = pX

    win = False
    while (not win) and (hasEmptySquare(board)):
        print (f"\nPlayer {player}")

        move = getPlayerMove(board, player)
        displayBoard(board)

        win = checkWin(board, move, player)
        if not win:
            player = pX if player == pO else pO

    if win:
        print (f"Player {player} has won the game!")
    else:
        print ('The game ended in a draw')


main()
```

Outputs [3] -- Player X wins, Player O wins, and a draw game