Zachary Shoults
HW 4 Algorithms
27 Nov 2016

1. Check number is prime in polynomial time?
   Yes. **Agrawal–Kayal–Saxena primality test**
   "… 2005, Carl Pomerance and H. W. Lenstra, Jr. demonstrated a variant of AKS that runs in **$O(\log(n)^6)$** operations, where *n* is the number to be tested "
   Source: **Primality testing with Gaussian periods** by H.W. Lenstra Jr and Carl Pomerance
   https://math.dartmouth.edu/~carlp/aks041411.pdf

2. The time it takes to complete all tasks is constant, the sum of t1, t2, t3 … tn. The only way to minimize time is to find the ordering of jobs such that the time taken to reconfigure the server is minimal (sum of all Tij 's). In this way it's the same as a Travelling Salesman problem. One can use branch and bound to determine a sequencing of tasks such that the edges Tij's are minimized. An adjacency matrix of the server tasks and configuration times can be made and a solution can be found in polynomial time, just as one would with TSP.

3. The way I solved it was to make a 2d matrix, size n x n, and initialize all values to be "blank". Choose any position on the grid to be the starting location. Change the value of that location to the "good" value. Add that position to the stack of moves made thus far. Then make a legal Knight move to any "blank" space on the board. Mark that position as "bad". Add that move to the stack. Then make another legal Knight move to a "blank" location. Every move alternates "good" and "bad" until you reach a position that has no moves to "blank" locations. At that point, start popping off moves from the stack looking for one that has legal moves you can take from that location and continue from there. When the stack is empty, the algorithm is done. The algorithm looks for a valid move from every position on the board twice. The runtime is $O(n^2)$ because there are n^2 locations on the board. This algorithm is very similar to Depth First Search. (*Non-optimized code included*)

4. The way I solved this problem is to approach it like a Bin Packing problem where Bins are like the number of days we have to finish all jobs and the capacity per bin is the total number of employees hired * 8hrs.

   There is no desire to minimize the number of hours a person works per day, just the number of people hired. So it is best to get a full 8 hours of work from every person hired to maximize the amount of each job accomplished per day. This will effectively minimize the number of people hired by maximizing the number of hours spent working on each job.

   The ideal minimum number of people to do the jobs would be those necessary to complete the jobs in exactly 10 days. Therefore, the best-case bin size would be the sum of all job times divided by 10. But this number is likely not exactly divisible by 8 hours, so our initial bin size will be the first number greater than the ideal that is divisible by 8. (if the jobs will take a total of 277 hours to complete, the ideal would be 27.7 hours/day over 10 days, but we will need 4 employees to cover 27.7 hours of work in a day and we will have them work at least 8 hours to maximize efficiency). Now, using the first-fit, best-fit technique, try to fit all jobs into the first bin that can contain them. If the solution is 10 or fewer bins, this is the number of employees we need. If there are jobs that cannot be fit into bins of this size, find the sum unused hours from each bin and see if this covers any un-assigned jobs. If it does, this is the solution. If not, the bin sizes need to be increased by 8 and the algorithm run again. (*Non-optimized code included*)

Zachary Shoults
HW 4 Algorithms
27 Nov 2016

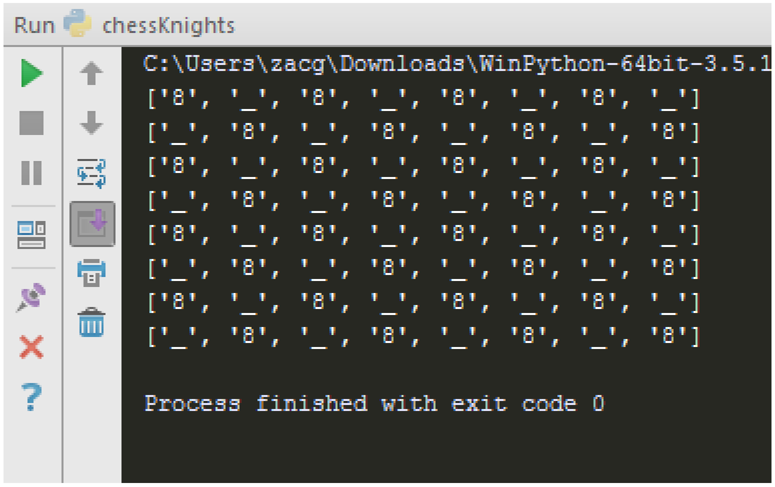
Chessboard code:

```
unvisited = " "      #initialize the chessboard with empty spaces
good = "8"           #put "8" in positions where knights can be
bad = "_"            #put "_" in positions where knights cannot be
board_w = 8          #board width
board_h = 9          #board height

def move_to(tup):    #move to the location given, change the color at that position
    global next_color, unvisited, good, bad
    r = tup[0]
    c = tup[1]
    if chessboard[r][c] == unvisited:
        chessboard[r][c] = next_color
        change_color()

def find_next(tup):
    global unvisited, board_h, board_w
    for x in [-2,-1,1,2]:
        for y in [-2,-1,1,2]:
            r = tup[0]
            c = tup[1]
            if(abs(y)==abs(x)):
                continue
            else:
                r+=x
                c+=y
                if r>board_h-1 or r<0 or c>board_w-1 or c<0:
                    continue
                if chessboard[r][c]==unvisited:
                    return (r,c)
    return False

def change_color():      #change the current color from "good" to "bad" or vice versa
    global next_color
    if next_color == good:
        next_color = bad
    else:
        next_color = good

#Create the chessboard with empty pieces
chessboard = [[unvisited for x in range(board_w)] for y in range(board_h)]
moves = [(r.randrange(0,board_h),r.randrange(0,board_w))]        #initial, random move
next_color = good  # keep 1  remove 8
move_to(moves[0])

while len(moves)>0:
    last_move = moves[len(moves)-1]
    next_move = find_next(last_move)
    if next_move==False: #no moves from current spot
        moves.pop()
        change_color() #chage the color because the previous move was different
        if len(moves)==0: #no moves from any previous square
            break
        continue
    else:
        move_to(next_move)
        moves.append(next_move)

[print(row) for row in chessboard]
```

Zachary Shoults
HW 4 Algorithms
27 Nov 2016

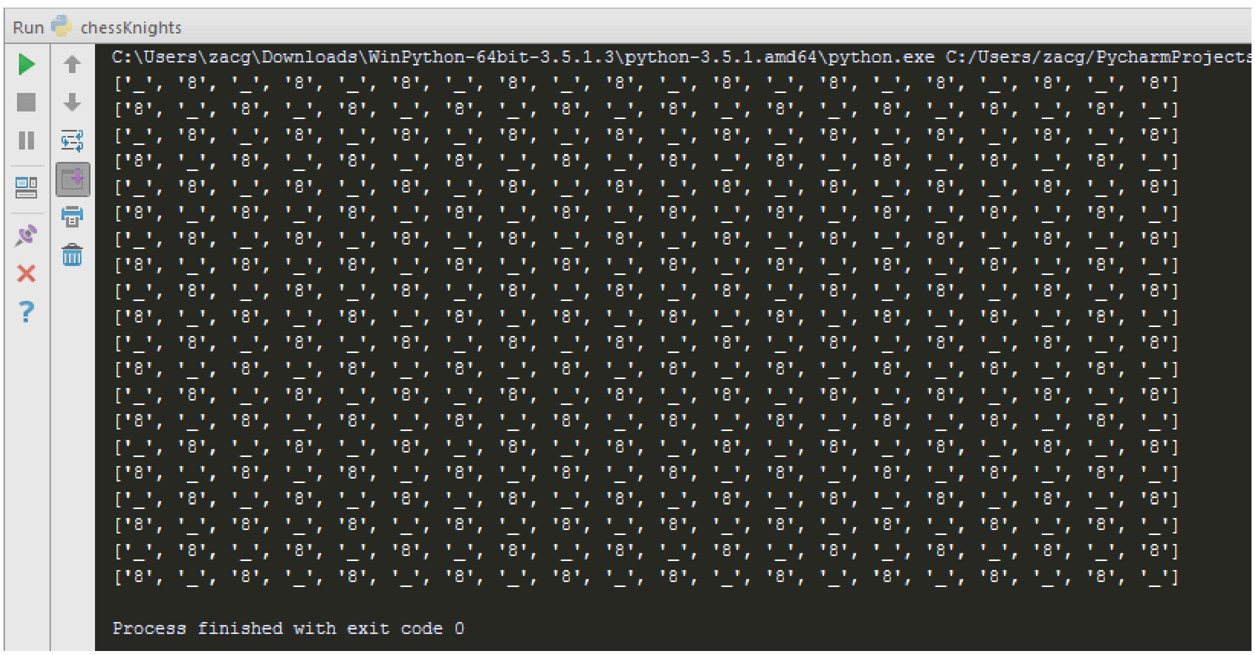
Chessboard code OUTPUT:

8x8:



```
Run chessKnights
C:\Users\zacg\Downloads\WinPython-64bit-3.5.1
['8', '_', '8', '_', '8', '_', '8', '_']
['_', '8', '_', '8', '_', '8', '_', '8']
['8', '_', '8', '_', '8', '_', '8', '_']
['_', '8', '_', '8', '_', '8', '_', '8']
['8', '_', '8', '_', '8', '_', '8', '_']
['_', '8', '_', '8', '_', '8', '_', '8']
['8', '_', '8', '_', '8', '_', '8', '_']
['_', '8', '_', '8', '_', '8', '_', '8']

Process finished with exit code 0
```

20x20:



```
Run chessKnights
C:\Users\zacg\Downloads\WinPython-64bit-3.5.1.3\python-3.5.1.amd64\python.exe C:/Users/zacg/PycharmProjects
['_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8']
['8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_']
['_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8']
['8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_']
['_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8']
['8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_']
['_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8']
['8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_']
['_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8']
['8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_']
['_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8']
['8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_']
['_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8']
['8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_']
['_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8']
['8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_']
['_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8']
['8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_']
['_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8']
['8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_', '8', '_']

Process finished with exit code 0
```

Zachary Shoults
HW 4 Algorithms
27 Nov 2016

Jobs and People Code:

```
num_jobs = r.randrange(30,200)                          ##Randomly generate job count
jobs = [r.randrange(3,24) for x in range(num_jobs)]   ##Randomly generate job durations
employee_hrs = 8                                          #Start with 1 person
bin_stamp = [0,employee_hrs,[]]    #Bins contain [current use, max, and list of jobs]
bins = [deepcopy(bin_stamp)]          #Start with 1 bin
large_jobs = []                       #Jobs that are too big for any bin
print("Jobs: {}".format(len(jobs)))

while True:
    job_num = 0                          #Give the jobs a number to track completeness
    for job in jobs:
        job_num+=1
        if job>employee_hrs:          #if job too big for any bin, track it separately
            large_jobs.append(job)
            continue
        found_bin = False
        for bin in bins:
            if bin[0]+job <= bin[1]:
                bin[0]+=job
                bin[2].append(job_num)
                found_bin = True
                break
        if not found_bin:          #create a new bin for this job if existing ones full
            n_bin = deepcopy(bin_stamp)
            n_bin[0]=job
            n_bin[2].append(job_num)
            bins.append(n_bin)

    remaining_hours = 0                 #calculate unused hours from each day
    for bin in bins:
        remaining_hours += bin[1]-bin[0]
    if sum(large_jobs)<=remaining_hours and len(bins)<=10:
         #if we can satisfy requirements, end the while-loop
        print("PASS")
        print("Solved with {} bins".format(len(bins)))
        for i,bin in enumerate(bins):
            print(bin)
        print("Additional jobs need {} hrs".format(sum(large_jobs)))
        print("Unused time from bins {} hrs".format(remaining_hours))
        print("Employees needed: {}".format(employee_hrs//8))
        break
    else:
        employee_hrs += 8                    #increase bin size before next loop
        bin_stamp = [0, employee_hrs, []]
        bins = [deepcopy(bin_stamp)]
        large_jobs = []
```

Zachary Shoults
HW 4 Algorithms
27 Nov 2016

Jobs and People OUTPUT: