# COSC 202, Final Exam Practice

1. What is the runtime of the following algorithm?

```
Alg(n)
$c = 0$
for $i = 1$ to $n$ do
    $j = 1$
    while $j < n$ do
        $c = c + 1$
        $j = j \times 2$
    end while
end for
```

**Solution:**

$$\sum_{i=1}^{n} \log i \approx n \times \log n$$

2. Give a $\theta(n)$-time non-recursive procedure that reverses a singly linked list of $n$ elements. The procedure should use no more than constant storage beyond that needed for the list itself.

**Solution:** Initialize three pointers **prev** as NULL, **current** as head, and **next** as NULL.

```
while current != NULL do
    next = current.next
    current.next = prev
    prev = current
    current = next
end while
```

See link for visualization:
https://www.geeksforgeeks.org/reverse-a-linked-list/

3. You want to sort a singly-linked list of $n$ strings, each of which has $w$ characters in an alphabet of size $a$. (For example, for lowercase English words, $a = 26$.) Below is a variation of Least-Significant-Digit-First radix sort that solves this problem:

```
Create a map where keys are characters and each value is a queue
For each character c in the alphabet:
     map.put(c, new Queue())
For each index i = w-1,...,0:
     Until the input list is empty:
          Remove the first string s
          Examine c = s.charAt(i), the ith character of the string
          Add s to the queue in map.get(c)
     For each character c in the alphabet in sorted order:
          Move all strings from map.get(c) back to the input list
```

Answer the following two questions in terms of parameters $n$, $w$, and $a$. Note. Not all parameters necessarily appear in the answers.

(a) What is the runtime of this algorithm?

> **Solution:** Runtime: $O(w \times (n + a))$.

(b) What is the **auxiliary** space complexity of this algorithm?
*Hint*: Because linked lists dynamically allocate nodes, removing an item from one list and adding it to another has a net effect of zero on the space used.

> **Solution:** Aux. space: $O(a)$.

4. You have to stack $n$ books on the shelves of length $L$. For each book $i$ you are given its height $h[i]$ and its thickness $t[i]$. The books must be placed in the given order $1 \ldots n$. We want to determine which books go on which shelves, given the following constraints:

(a) We want to minimize the number of shelves used.

> **Solution:** Fill as many books as possible in each shelf. When shelf is too full, start another. run-time: $O(n)$. (Greedy algorithm works here)

(b) We are able to adjust the height of the shelves to make a perfect fit. This time, we want to minimize the total height of the shelves used, but assume that all books are of the same height.

> **Solution:** Fill as many books as possible in each shelf. When shelf is too full, start another. run-time: $O(n)$. (Greedy algorithm works here as well)

(c) We want to minimize the total height of the shelves used, but now, books can be of different heights. Can you think of a recurrence relation for this problem?

> **Solution:** For a complete explanation, watch: Vijay Ramachandran's video: *S21 COSC 202 Topic 8.4.mp4* in the course website under **Resources**.

5. In this question you will analyze the time complexity for a divide and conquer problem and design a dynamic programming algorithm.

   You are going on a road trip, and have planned out your route from start to finish on a straight line. It is a multi-day trip and requires making stops for overnight stays. You have set a limit of traveling 300 miles per day. You have mapped out $n-2$ possible locations to stay overnight along your route. i.e., with start and destination, that would be a total of $n$ locations. You are given two arrays $d$ and $c$ such that for each location $i$, you know:

   - Distance of $i$ from the start of the route, $d[i]$
   - The cost to stay overnight at $i$, $c[i]$

   The start point of the trip is marked by $d[0] = 0$ and the ending point of the trip is marked by $d[n-1]$. Since you don't need to stay overnight at the start and end locations, $c[0] = c[n-1] = 0$.

The following recursive algorithm finds the minimum total cost for the trip by optimizing the number of stops. (It assumes that it is possible to complete the trip)

```
# global variable
MaxMilesPerDay = 300

# declaring the recursion
def RecursiveRoadTrip(d,c,startIndex):
    """Returns minimum total cost.
    """
    if startIndex == len(d) - 1:
        return 0 #end of trip
    curmin = math.inf
    j = startIndex + 1
    while j < len(d) and d[j] - d[startIndex] <= MaxMilesPerDay:
        cur = c[j] + RecursiveRoadTrip(d,c,j)
        if  cur < curmin:
            curmin = cur
        j += 1
    return curmin

# calling the recursion
print(RecursiveRoadTrip(d,c,0))
```

Using the above algorithm, you can verify that for the following input, the total cost of the trip is indeed 12. Notice that location $i = 7$, or **len(d) - 1**, is the destination. It is also the the base case of the recursion!

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $d[i]$ | 0 | 100 | 150 | 270 | 350 | 400 | 550 | 600 |
| $c[i]$ | 0 | 10 | 5 | 15 | 7 | 8 | 1 | 0 |

(a) (4 points) If there are $n$ locations, and at any given location $i$, there are on average $k$ possible locations between $i$ and the end that are within 300 miles to $i$, what is the worst-case run-time of the recursive algorithm above in terms of $n$ and $k$?

**Solution:** $O(k^n)$

(b) (8 points) With the help of the above recursive solution, design a dynamic programming algorithm that solves this problem faster. Please call it min-cost-trip(d,c).

**Solution:**

```
min-cost-trip(d,c)
Let Cost[0...n+1] be a new array
```

```
Cost[n+1]=0
for $k$ ←n to 0 do
    Cost[k] = $\infty$
    j=k+1
    while d[j]-d[k] $\leq$ 500 and j $\leq$ n+1 do
        current = Cost[j] + c[j]
        if Cost[k] = $\infty$ or Cost[k] > current then Cost[k] = current
        end if
        j = j + 1
    end while
end for
return Cost[0]
```

(c) (4 points) Under the same assumptions of part(a), what would be the run-time of min-cost-trip(d,c) in terms of $n$ and $k$?

**Solution:** $O(n \times k)$

6. You want to traverse a given graph, $G$, from a vertex $A$, using both Breadth-First-Search (BFS) and Depth-First-Search (DFS) traversals. Not known to you at the time of running the traversals, $G$ turns out to be a complete binary tree with $A$ at its root. In retrospect, and in terms of time and/or space complexities, which of the two traversals was more efficient?

**Solution: Answer:** DFS is more efficient, since its auxiliary memory (stack) will be $O(\log(n))$ whereas that of BFS (queue) will be $O(n)$.

7. Dijkstra's algorithm determines the shortest distance between a source node and any other node in a graph by continuously calculating the shortest distance beginning from a starting point, and to excluding longer distances when making an update:

```
Dijkstra(Graph, source)
for $v \in G$ do                                  ▷ initialization
    dist[$v$] = $\infty$                          ▷ initial distance from source to $v$
    previous[$v$] = NULL                          ▷ parent node of $v$ in optimal path
end for
dist[source] = 0
Q = priority_queue()                              ▷ Q is a min heap
for $v \in G$ do                                  ▷ push all nodes to Q
    Q.push($v$)
end for
while Q.size()$\neq$ 0 do                         ▷ main loop
    print(Q)                                      ▷ for debugging
    $u$ = Q.pop()                                 ▷ $u$ node in Q with smallest dist[ ]
    for neighbor $v$ of $u$ do                    ▷ where $v$ still in Q
        alt = dist[$u$] + dist_between($u$, $v$)
        if alt < dist[$v$] then
            dist[$v$] = alt                       ▷ update distance from source to $v$
            previous[$v$] = $u$
        end if
    end for
end while
return previous[ ]                                ▷ return all parent nodes (paths)
```

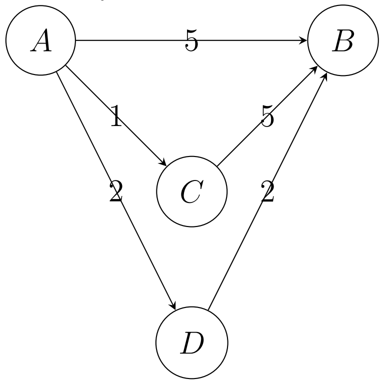
Suppose in the following graph, we would like to find the shortest paths from node $A$ to other nodes, i.e., parents = Dijkstra(Graph, $A$).

(a) What will be the content of parents?

**Solution:**

| $v$ | A | B | C | D |
|---|---|---|---|---|
| $parents[v]$ | NULL | D | A | A |

(b) Write down the content of the min heap, Q, as it changes throughout the algorithm at every iteration of the while loop; i.e., Execute print(Q)



*Hint*: For content of Q, you can write your values in pairs (node, distance). For instance, before the first iteration of the while loop, all nodes and associated distances are in Q; i.e., Q = [($A$, 0),($B$, $\infty$),($C$, $\infty$),($D$, $\infty$)].

**Solution:** 1st print(Q), Q = [($A$, 0),($B$, $\infty$),($C$, $\infty$),($D$, $\infty$)]
2nd print(Q), Q = [($B$, 5),($C$, 1),($D$, 2)]
3rd print(Q), Q = [($B$, 5),($D$, 2)]
4th print(Q), Q = [($B$, 4)]
Note: The above only shows the content of the min heap at each moment. Please ignore the order of the items in the above. Item with minimum distance is always removed when Q.pop() is called.

8. Kruskal's algorithm finds a Minimum Spanning Tree (MST) of an undirected connected weighted graph. After sorting edges from smallest to largest edge-weights, it then loops through the edges and adds them to the tree. You run this algorithm on graph $G$ and obtain an MST. Answer the following:

(a) True or False: Suppose we subtract a constant positive value $X$ from all edge weights in $G$, such that the weight $w$ of any edge $e$ is now $w - X$. The original MST will still be an MST but with a different value.

**Solution: True**

(b) True or False: Suppose we increase the weight of the edge having already highest value. The original MST will still be an MST on the new graph.

**Solution: True**