Hash Tables

- *Finding Primes*
  - **Sieve of Eratosthenes**:
    - Efficient algorithm to find all primes up to (n).
    - **Steps**:
      1. Create a boolean array `prime[0:n]` and initialize all entries as true.
      2. Set `prime[0]` and `prime[1]` to false (0 and 1 are not primes).
      3. For $(p = 2)$ to $(\sqrt{n})$:
         - If `prime[p]` is true, then mark all multiples of `p` as false.
      4. Remaining true indices are prime numbers.

Sorting Routine with Quicksort

- Quicksort Algorithm

```
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(right)
```

- Sequential Cutoff:
  - Switch to Insertion Sort for small subarrays to improve performance.
  - Use Case: When the size of the subarray is below a certain threshold (e.g., 10 elements).

Programming Concepts

- ***Debugging Pseudocode***
- *Common Errors*
  - Off-by-one: Loop Boundries
  - incorrect Loop range
  - Infinite Loops: Incorrect loop conditions
  - Missing termination condition
  - Base Case Errors: In recursion, ensuring base cases are correct
  - Incorrect base case in recursive function:
- ***Recursion Vs Iteration***
- *Recursion:*
- Elegant, simpler code
- Higher space complexity (stack Frames)
- *Iteration:*
  - Often more efficeient (less space)
  - Can be harderr to write for complex problems.

```python
infinit = float('inf') #infinity

# function for calculating the node with the shortest distance
def min_distance_node(distance, visited):
    min_dist = float('inf')
    min_dist_node = None
    for node in distance:
        dist = distance[node]
        if dist< min_dist and node not in visited:
            min_dist = dist
            min_dist_node = node
    return min_dist_node

def dijkstra(graph, start, target):
    # create a distance dictionary for all other nodes with distances set to infinity
    # starting distance set to 0
    distance = {node: infinit for node in graph}
    distance[start] = 0
    # create a dictionary to keep track of the parent nodes
    # all nodes set to None
    parents = {node: None for node in graph}
    # list of nodes that we have visited
    visited = []

    # select the current node as the lowest distance node that has not been visited yet
    current = min_distance_node(distance, visited)
    while current: # if all nodes have already been visited, finish the loop
        dist = distance[current] # get the distance to the current node
        neighbors = graph[current] # get the neighbors of the current node
        for n in neighbors.keys(): # Go through the neighbors of the current node
            new_dist = dist + neighbors[n] # calculate the new distance to that node
            if distance[n] > new_dist: # if it is shorter to get to the neighbor
                # by going through this node
                distance[n] = new_dist # update the distance for this node
                parents[n] = current # the node becomes the new parent of the neighbor
            visited.append(current) # mark the node as visited
        current = min_distance_node(distance, visited) # find the next node that we have to process
        # and continue looping through


    ## Recreate the path and the distance to the target node
    path = [] # list to store the nodes that we visited along the path
    if distance[target] is not infinit: # if the target node is reachable from the starting node
        current = target # start from the end
        # backtrack to recreate the path
        while current: # while current is not none
            path.append(current) # add the current node that we are at to the path
            current = parents[current] # go to the current node's parent node
            # then continue looping until there are no more parent nodes
        path.reverse() # reverse the order of the path so that we start at the starting node

    # returns a tuple with the path and the distance of the path
    return path, distance[target]
```