Gebze Technical University
Department of Computer Engineering
CSE 321 Introduction to Algorithm Design
Fall 2020
Final Exam (Take-Home)
January 18th 2021-January 22nd 2021

Student ID and Name	Q1 (20)	Q2 (20)	Q3 (20)	Q4 (20)	Q5 (20)	Total
171044065 Berke Belgin						

Read the instructions below carefully

- You need to submit your exam paper to Moodle by January 22nd, 2021 at 23:55 pm as a single PDF file.

- You can submit your paper in any form you like. You may opt to use separate papers FOR your solutions. If this is the case, then you need to merge the exam paper I submitted and your solutions to a single PDF file such that the exam paper I have given appears first. Your Python codes should be in a separate file. Submit everything as a single zip file. Please include your student ID, your name and your last name both in the name of your file and its contents.

Q1. Suppose that you are given an array of letters and you are asked to find a subarray with maximum length having the property that the subarray remains the same when read forward and backward. Design a dynamic programming algorithm FOR this problem. Provide the recursive formula of your algorithm and explain the formula. Provide also the pseudocode of your algorithm together with its explanation. Analyze the computational complexity of your algorithm as well. Implement your algorithm as a Python program. **(20 points)**

Q2. Let $A = (x_1, x_2, \ldots, x_n)$ be a list of n numbers, and let $[a_1, b_1], \ldots, [a_n, b_n]$ be n intervals with $1 \le a_i \le b_i \le n$, FOR all $1 \le i \le n$. Design a divide-and-conquer algorithm such that FOR every interval $[a_i, b_i]$, all values $m_i = \min\{x_j \mid a_i \le j \le b_i\}$ are simultaneously computed with an overall complexity of $O(n \log(n))$. Express your algorithm as pseudocode and explain your pseudocode. Analyze your algorithm, prove its correctness and its computational complexity. Implement your algorithm using Python. **(20 points)**

Q3. Suppose that you are on a road that is on a line and there are certain places where you can put advertisements and earn money. The possible locations FOR the ads are x_1, x_2, …, x_n. The length of the road is M kilometers. The money you earn FOR an ad at location x_i is $r_i > 0$. Your restriction is that you have to place your ads within a distance more than 4 kilometers from each other. Design a dynamic programming algorithm that makes the ad placement such that you maximize your total money earned. Provide the recursive formula of your algorithm and explain the formula. Provide also the pseudocode of your algorithm together with its explanation. Analyze the computational complexity of your algorithm as well. Implement your algorithm as a Python program. **(20 points)**

Q4. A group of people and a group of jobs is given as input. Any person can be assigned any job and a certain cost value is associated with this assignment, FOR instance depending on the duration of time that the pertinent person finishes the pertinent job. This cost hinges upon the person-job assignment. Propose a polynomial-time algorithm that assigns exactly one person to each job such that the maximum cost among the assignments (not the total cost!) is minimized. Describe your algorithm using pseudocode and implement it using Python. Analyze the best case, worst case, and average-case performance of the running time of your algorithm. **(20 points)**

Q5. Unlike our definition of inversion in class, consider the case where an inversion is a pair $i < j$ such that $x_i > 2 x_j$ in a given list of numbers $x_1, \ldots, x_n$. Design a divide and conquer algorithm with complexity O(n log n) and finds the total number of inversions in this case. Express your algorithm as pseudocode and explain your pseudocode. Analyze your algorithm, prove its correctness and its computational complexity. Implement your algorithm using Python. **(20 points)**

Q1.

I used loops instead of recursion instead of wasting computation time for the values already searched. Complexity is **O(n^2).**

```
PROCEDURE subArray(arr)
  FOR i IN RANGE arr_size
    IF i != 0 AND arr[i] == arr[i - 1] THEN
      tmp_arr = arr[i-1:i+1]
    ELSE THEN
      tmp_arr = arr[i]
    ENDIF
    FOR n_r IN RANGE i + 1, arr_size
      n_l = n_r - len(tmp_arr) - 1
      IF n_r < arr_size AND n_l >= 0 and arr[n_r] == arr[n_l] THEN
        tmp_arr = arr[n_l] + tmp_arr + arr[n_r]
      ELSE THEN
        BREAK
      ENDIF

    IF len(tmp_arr) >= len(res_arr) THEN
      res_arr = tmp_arr
    ENDIF
  print('string: ' + res_arr)
  print('size: ' + str(len(res_arr)))
END
```

Q2.

In this part I created a 2d array to look for minimums in given boundaries.

```
PROCEDURE preprocess(arr: list, n: int)
  FOR i in range(n):
    lookup[i][0] = i

  j = 1
  while (2 ** j) <= n:
    i = 0
    while i + (2 ** j) - 1 < n:
      IF (arr[lookup[i][j - 1]] < arr[lookup[i + (2 ** (j - 1))][j - 1]]):
        lookup[i][j] = lookup[i][j - 1]
      else:
        lookup[i][j] = lookup[i + (2 ** (j - 1))][j - 1]
      i += 1
    j += 1
END
PROCEDURE query(arr, L, R)
  j = int(log2(R - L + 1))

  IF (arr[lookup[L][j]] <= arr[lookup[R - (2 ** j) + 1][j]]):
    return arr[lookup[L][j]]
  else:
    return arr[lookup[R - (2 ** j) + 1][j]]
END
PROCEDURE RMQ(arr, n, left, right)
  preprocess(arr, n)
  print("Minimum of [%d, %d] is %d" % (left, right, query(arr, left, right)))
END
```

Q3.

Non-recursive solution for given question. Complexity is **O(n).**

```
PROCEDURE maxRevenue(min_distance, total_distance, distances, revenue, size)
   maxRev = [0] * (total_distance + 1)
   next_bb = 0
   FOR i in range(1, total_distance + 1):
      IF (next_bb < size) THEN
         IF (distances[next_bb] != i) THEN
            maxRev[i] = maxRev[i - 1]
         ELSE THEN
            IF (i <= min_distance) THEN
               maxRev[i] = max(maxRev[i - 1], revenue[next_bb])
            ELSE THEN
               maxRev[i] = max(maxRev[i - min_distance - 1] + revenue[next_bb], maxRev[i - 1])
            next_bb += 1
             END IF
         END IF
      ELSE THEN
         maxRev[i] = maxRev[i - 1]
      END IF
   return maxRev[total_distance]
END
```

Q4.

Searches the table for its max value and deletes it until there is one value in a column or row left, then it inserts that value into result array

```
PROCEDURE  clearColumn(x, assignments, sizeY, columnSizes, resultAssignment):
   foundY = 0
   columnSizes[x] = 0
   FOR i in range(sizeY):
      IF assignments[i][x] != -1:
         print(assignments[i][x])
         foundY = i
         resultAssignment[x] = assignments[i][x]
         assignments[i][x] = -1
   return foundY

PROCEDURE  clearRow(y, assignments, sizeX, rowSizes, resultAssignment):
   foundX = 0
   rowSizes[y] = 0
   FOR i in range(sizeX):
      IF assignments[y][i] != -1:
         print(assignments[y][i])
         foundX = i
         resultAssignment[i] = assignments[y][i]
         assignments[y][i] = -1
   return foundX

PROCEDURE  removeElement(y, x, assignments, sizeY, sizeX, rowSizes, columnSizes, resultAssignment):
   assignments[y][x] = -1
   rowSizes[y] -= 1
   columnSizes[x] -= 1

   IF columnSizes[x] == 1:
      foundY = clearColumn(x, assignments, sizeY, columnSizes, resultAssignment)

   IF rowSizes[y] == 1:
      foundX = clearRow(y, assignments, sizeX, rowSizes, resultAssignment)

PROCEDURE  getResult(sizeY, sizeX, assignments):
   resultLocalMaxAssignment = float('inf')

   resultAssignment = []
   rowSizes = []
   columnSizes = []

   FOR i in range(sizeX):
      resultAssignment.append(-1)
   FOR i in range(sizeY):
      rowSizes.append(sizeX)
   FOR i in range(sizeX):
      columnSizes.append(sizeY)

   maxValue = -1
   while maxValue != 0:
      FOR i in range(sizeY):
         print(assignments[i])
      print('')


      maxValue = 0
      FOR y in range(sizeY):
         FOR x in range(sizeX):
            IF assignments[y][x] > maxValue:
               maxValue = assignments[y][x]
      FOR y in range(sizeY):
         FOR x in range(sizeX):
            IF assignments[y][x] == maxValue:
               removeElement(y, x, assignments,sizeY, sizeX, rowSizes, columnSizes, resultAssignment)
   return resultAssignment
```

Q5.

I used merge sort to implement this question.

```
PROCEDURE merge(arr, temp_arr, left, mid, right)
   inv_count = 0
   i = left
   j = mid
   k = left

   WHILE i <= mid - 1 AND j <= right
      IF arr[i] > 2 * arr[j]:
         inv_count += (mid - i)
         j += 1
      else:
         i += 1

   i = left
   j = mid
   k = left

   WHILE i <= mid - 1 AND j <= right
      IF arr[i] <= arr[j] THEN
         temp_arr[k] = arr[i]
         i = i + 1
         k = k + 1
      ELSE
         temp_arr[k] = arr[j]
         j = j + 1
         k = k + 1

   WHILE i <= mid - 1
      temp_arr[k] = arr[i]
      i = i + 1
      k = k + 1

   WHILE j <= right
      temp_arr[k] = arr[j]
      j = j + 1
      k = k + 1

   FOR I IN range(left, right + 1)
      arr[i] = temp_arr[i]

   RETURN inv_count
END

PROCEDURE mergeSort_h(arr, temp_arr, left, right)
   inv_count = 0
   IF right > left:
      left_l = left
      left_r = (right + left) // 2
      right_l = (right + left) // 2 + 1
      right_r = right

      inv_count = mergeSort_h(arr, temp_arr, left_l, left_r)
      inv_count += mergeSort_h(arr, temp_arr, right_l, right_r)
      inv_count += merge(arr, temp_arr, left, right_l, right_r)
   RETURN inv_count
END

PROCEDURE mergeSort(arr)
   n = len(arr)
   temp_arr = [0 FOR i in range(n)]
   RETURN mergeSort_h(arr, temp_arr, 0, n - 1)
END
```