Due 23:59 Oct 9 (Sunday). There are 100 points in this assignment.
Submit your answers (**must be typed**) in pdf file to CourSys
`https://coursys.sfu.ca/2022fa-cmpt-705-x1/`. Submissions received after 23:59 of Oct 9 will get penalty of reducing points: 20 and 50 points deductions for submissions received at $[00:00, 00:10]$ and $(00:10, 00:30]$ of Oct 10, respectively; no points will be given to submissions after 00 : 30 of Oct 10.

1. (Chapter 4 Problem 3 of text book) 20 points

   There are n boxes arriving at a shipping company in the order $b_1, .., b_n$, each box b_i has a positive weight w_i. The company uses trucks to deliver the boxes to a destination, each truck has a weight capacity W, satisfying:

   - the total weight of all boxes in each truck is at most W; and
   - the order of arrivals is preserved: for any pair of boxes b_i and b_j with $i < j$, if b_i and b_j are packed in different trucks, say b_i to x and b_j to y, then x departs earlier than y.

   The company uses the following greedy algorithm to solve the problem: Boxes are packed into a truck in the order they arrive until if the next box is packed, the total weight of the boxes is greater than W; then the truck is sent on its way; and the boxes are packed into the next truck as above. Prove that the greedy algorithm uses the minimum number of trucks to deliver the boxes as required.

Answer If the weights of the packages are represented by w with subscripts representing the indices and W being the total weight of each of the trucks, let there be two trucks for this example x and y. The first given is that the trucks must leave the facility in an order, i.e., a truck cannot leave before the prior truck has left the facility. This means that the list of trucks is sorted in a non-decreasing order. One assumption is that no package has a weight greater than W, i.e., $w_i \not> W$. This can be proved by contradiction, say that there is another approach which provides the optimal solution. The case that needs to be disproved is that the optimal solution does not change to the next box/package before the greedy approach. As the opposite of it would never arise, i.e., if the greedy approach were to switch to the next box before the optimal solution it would mean that the optimal solution will switch to the next box at the same time (as it is the optimal solution).

Hence the other case is if the optimal solution packages the next box before the greedy approach. Since the trucks cannot be overweight, this means that if the truck x is packaged and is about to add another package b_k, but the workers figure out that the package would make the truck overweight they give the signal to the truck x to go and place the package b_k in the next truck y. This is what the greedy approach does. So, this means that the optimal solution and the greedy approach are one and the same.

---

2. (Chapter 4 Problem 8 of the text book) 20 points

   Prove that a connected graph G with edge costs that are all distinct has a unique minimum spanning tree.

Answer

This statement can be proved by contradiction. Let's assume that there are two minimum spanning trees to a graph with unique edges. Having unique edges means that the tree has one and only one edge with least weight, say e_1. According to the initial assumption there are two minimum spanning trees to this connected graph G, say T_1 and T_2. The edge with the least weight belongs to T_1, and not T_2. There is another edge e_2 in the second tree T_2, which for sure has more weight than the edge e_1. If we were to have the edge e_1 included in the tree T_1, it would form a cycle. Since the tree T_1 is the minimum spanning tree different from T_2, therefore tree T_1 does not have the edge e_2. Now as the edge e_1 has the least weight, if we were to remove e_2 from the second tree T_2 we will get the same tree as T_1. Which means that the now tree T_2 is also a Minimum Spanning Tree.

---

3. (Chapter 4 Problem 13 of the text book) 20 points

   Consider the following variation on the Interval Scheduling Problem. You have a processor that can operate 24 hours a day, every day. People submit requests to run daily jobs on the processor. Each such job comes with a start time and an end time; if the job is accepted to run on the processor, it must run continuously, every day, for the period between its start and end times (certain jobs can begin before midnight and end after midnight; this makes for a type of situation different from the Interval Scheduling Problem). Given a list of n such jobs, your goal is to accept as many jobs as possible (regardless of their length), subject to the constraint that the processor can run at most one job at any given point in time. Provide an algorithm to do this with a running time that is polynomial in n. You may assume for simplicity that no two jobs have the same start or end times.

Answer

For this algorithm we can just run a loop over the sorted version of the list (sorted based on the finish times) and just run the assigned tasks, and after it is finished we move on to the other task. This takes O(n) running time to run the tasks.

```
def schedule(A):
        A = sort_finish_times(A)
        run A[0]

        for i in range(1, len(A)):
                if A[i].start >= A[i − 1].finish:
                        run A[i] // Add the activity to the schedule
```

---

4. (Chapter 5 Problem 1 of the text book) 20 points

   Assume that there are two separate databases, each database contains n numerical values and no two of the $2n$ values are the same. You would like to determine the median (the n-th smallest value) of these $2n$ values. The only way you can access these values is through queries to the databases. In a single query, you can specify a value k to one of the two databases, and the chosen database will return the k-th smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible. Give an algorithm that finds the median value using at most $O(\log n)$ queries.

Answer Let us say the expensive queries of the database query take 1 time step, we can start from the middle of both the databases and query the database for the median. Then we move from there given the value that comes from the query to the database, if the median of the database 1 is greater than that of database 2, we move the pointer of the first database by a factor of $\frac{n}{2^i}$ to the left, and for the database 2 we move it to the right by the same factor. We move reverse of what is described previously if the median of database 2 is greater than that of database 1. This algorithm assumes that both the databases are in increasing sorted order and that no element is repeated in the algorithm. This would ensure that there are no two values same, which would mean that we are moving the wrong direction when comparing the two median values. In the end we just compare the two median values, and whichever is smaller is the required value. This algorithm takes O(log n) time, given that the two databases are sorted, as each of the queries take 1 time step, and we are essentially dividing the task and solving them alone. The algorithm is given below:

```
def databasesMinMedian(db1, db2):
        n = len(db1)
        i = n//2
        j = n//2

        while loop:
                median1 = medianQuery(db1, i)
                median2 = medianQuery(db2, j)
                if median1 > median2:
                        move the pointer i to left by n/2 factor of the looping variable
                        move the pointer j to right by n/2 factor of the looping variable
                else:
                        move the pointer i to right by n/2 factor of the looping variable
                        move the pointer j to left by n/2 factor of the looping variable
        return min(median1, median2)
```

---

5. (Chapter 5 Problem 2 of the text book) 20 points

   Given a sequence of n distinct numbers $a_1, .., a_n$, a pair (a_i, a_j) is called a significant inversion if $i < j$ and $a_i > 2a_j$. Give an $O(n \log n)$ time algorithm to count the number of significant inversions for a given sequence of n distinct numbers.

Answer Given a sequence of distinct numbers, we divide the list into two parts, and find their individual inversions and then merge the two parts with their counts. The splitting of the list and quering is done recursively and then the merging is done based on the given conditional statement $a_i > 2a_j$. The algorithm for getting the significant inversions is as follows:

A = [a_1, a_2, ..., a_n]

```
def findInversions(A):
        count = 0
        // Divide the Array into two halves
        A1 = [a_1, ..., a_{n/2}]
        A2 = [a_{n/2}, ..., a_n]

        countA1 = findInversions(A1)
        countA2 = findInversions(A2)
        countA = mergeCount(A1, A2)

        return count, mergeA

def mergeCount(A1, A2):
        // Loop over each of the arrays and maintain their pointers
        i, j, k = 0, 0, 0
        count = 0

        while i < len(A1) and j < len(A2):
                if a_i > 2a_j
                        mergedA[k] = A1[i]
                        i += 1
                else:
                        mergedA[k] = A2[j]
                        j += 1
                        count += len(A1) − i
                k += 1

        return count, mergedA
```

Given that the inversions take a time of the factor $O(logn)$ based on the conditional $a_i > 2a_j$ because we are dividing the problem into parts recursively and solving the problem when we have sort of a tree and then we make our way up (merging). If given the worst-case scenario for merging, the algorithm will take $O(n)$, as it might have to walk the whole of the original data list. So, the overall running complexity is $O(nlogn)$.
