@author Rufina Talalaeva

Theoretical Part

3.1

Theorem 4.1 (Master theorem)
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n),$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

a. $T(n) = 16 * N\left(\frac{n}{4}\right) + n$

Answer. $T(n) = \Theta(n^2)$. Case 1. Justification: we have $a = 16$, $b = 4$, $f(n) = n$, and thus we have that $n^{\log_b a} = n^{\log_4 16} = \Theta(n^2)$. Since $f(n) = O\left(n^{\log_4 16 - \epsilon}\right)$, where $\epsilon = 1$, we can apply case 1 of the master theorem and conclude that the solution is $T(n) = \Theta(n^2)$.

b. $T(n) = N\left(\frac{n}{2}\right) + 2^n$

Answer. $T(n) = \Theta(2^n)$. Case 3. Justification: we have $a = 1$, $b = 2$, $f(n) = 2^n$, and thus we have that $n^{\log_b a} = n^{\log_2 1} = \Theta(1)$. Since $f(n) = \Omega\left(n^{\log_2 1 + \epsilon}\right)$, where $\epsilon > 0$, we can apply case 3 if we can show that the regularity condition holds for $f(n)$. For sufficiently large n, we have that $a * f\left(\frac{n}{b}\right) = 2^{\frac{n}{2}} \leq \frac{1}{2} * 2^n = c * f(n)\ for\ c = \frac{1}{2}$. Consequently, by case 3 of the master theorem, the solution is $T(n) = \Theta(2^n)$.

c. $T(n) = 2 * T\left(\frac{n}{2}\right) + \frac{n}{\log n}$

Answer. Do not apply – non-polynomial difference between $f(n)$ and $n^{\log_b a}$. Justification: we have $a = 2$, $b = 2$, $f(n) = \frac{n}{\log n}$, and thus we have that $n^{\log_b a} = n^{\log_2 2} = \Theta(n)$. We might mistakenly think that case 1 should apply, since $f(n) = \frac{n}{\log n}$ is asymptotically smaller than $n^{\log_b a} = n$. The problem is that it is not polynomially smaller. The ratio $\frac{n^{\log_b a}}{f(n)} = \frac{n}{\frac{n}{\log n}} = \log n$ is asymptotically less than n^{ϵ} for any positive constant ϵ. Consequently, the recurrence falls into the gap between case 1 and case 2.

d. $T(n) = 4 * T\left(\frac{n}{2}\right) + n^2$

Answer. $T(n) = \Theta(n^2 * \lg n)$. Case 2. Justification: we have $a = 4$, $b = 2$, $f(n) = n^2$, and thus we have that $n^{\log_b a} = n^{\log_2 4} = \Theta(n^2)$. Since $f(n) = \Theta\left(n^{\log_2 4}\right)$, we can apply case 2 of the master theorem and conclude that the solution is $T(n) = \Theta(n^2 * \lg n)$.

3.2

Step1. Define your sub-problem.

Let $m(i)$ be the cost for the best solution to travel from station i to station n($1 \leq i \leq n$). Then the cost we are looking for is $m(1)$.

Step2. Present your recurrence.

Function $m(i)$ chooses the minimal price to get from i-th city to the n-th city by checking all such distances as $f_{i,j}$+ $m(j)$, where j is [i+1 .. n], and memorizing the best price for i in the array of prices.

$$m(i) = \begin{cases} 0, \quad if\ i = n, \\ \min(f[i][n], f[i][n-1] + m(n-1), \dots, f[i][i+1] + m(i+1)), \quad \text{if } i\ != n \end{cases}$$

Step3. Prove that your recurrence is correct.

We are given the table of $f_{i,j}$ - distances from i-th city to the j-th city($1 \leq i \leq j \leq n$).

1. Base of induction: m(n) = 0
2. Let's suppose that for m(i) = min(f[i][n],
   f[i][n-1] + m(n-1),
   …,
   f[i][i+1] + m(i+1)).
3. Let prove our assumption for m(i-1). Until this moment we have calculated all optimal prices for trip from i..n city to n-th city. Now we should calculate the optimal price for trip from (i-1)-th city to the n-th city. We can do it only by taking minimum out of all prices like f[i-1][j] + m(j), (i < j)where m(j) has been already calculated as optimal price from j-th city to n-th and f[i-1][j] is known price for the ticket from (i-1)-th town to j-th town. So, m(i-1) = min(f[i-1][n],
   f[i-1][n-1] + m(n-1),
   …,
   f[i-1][i] + m(i)). ■

Step4. State your base cases.

<u>Base case</u>: m(n) = 0.

Step5. Present the algorithm.

```
1  // price[i] means the minimum price to go from station i to station n.
2  // initially set all prices to INFINITY, where INFINITY is really large number
3  for i := 1 to n
4      price[i] := INFINITY
5
6  int m(i):
7      // distance from n to itself is 0 according to the task
8      if i = n:
9          return 0
10     // if the minimum price from i to n has been already
11     // calculated before, just return it
12     if price[i] != INFINITY:
13         return price[i]
14     // if it is not calculated
15     else:
16         // initial price is set to INFINITY
17         res = INFINITY
18         // for each station j in [i+1 .. n]
19         // calculate the minimal price out of all possible roads
20         for j := i+1 to n
21             dist := f[i][j] + m(j)
22             if res > dist:
23                 res = dist
24         // memorizing the minimal price to get from i-th city to the n-th city
25         price[i] = res
26         return res
```

Step6. Running time.

$T(i) = (number\ of\ calls\ of\ function\ m) * (\text{time it takes to execute each of these calls})$

Two ways of calling function m:
1.when we call this function for the first time we call function m is when we call m(1)(To get from 1st city to the n-th)
2.call this function from the 21st line of pseudo-code

Notice that block of code(15-26) will be executed at most n times, and it's true, because there are n possible arguments to this function(1..n). And each time this function is called with each of those arguments, the return value will be stored in price at index i(price[i]). And now let's look at 21st line of pseudo-code, function m is called (n-1) times(when i=1).

O(n)

$T(1) = T(2) + T(3) + \dots + T(n)$ = in This step all values of m(i..n) will be calculated

$T(2) = T(3) + T(4) + \dots + T(n)$

. . .

$T(n-1) = T(n)$ = $O(1)$ in this step T(n-1) will use value from m(n)

$T(n) = O(1)$

Thus, $T(i) = \big(1 + n(n-1)\big) * \mathrm{O}(1) = \mathrm{O}(n^2 - \mathrm{n} + 1) = \mathrm{O}(n^2)$.

Source that I have read before I complete this task: https://www.geeksforgeeks.org/find-the-minimum-cost-to-reach-a-destination-where-every-station-is-connected-in-one-direction/