Exam - June 2020

Algorithms and Data Structures (DAT2, SW2, DV2)

Instructions. This exam consists of **five questions** and you have **four hours** to solve them. You can answer the questions directly on this paper, or use additional sheets of paper which have to be hand-in as a **single pdf file**.

- Before starting solving the questions, read carefully the exam guidelines at `https://www.moodle.aau.dk/mod/page/view.php?id=1060646`.
- Read carefully the text of each exercise before solving it! Pay particular attentions to the terms in bold.
- **CLRS** refers to the textbook T.H. Cormen, Ch. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms* (3rd edition).
- Make an effort to use a readable handwriting and to present your solutions neatly.

Question 1. 20 Pts

Identifying asymptotic notation. (Note: lg means logarithm in base 2)

(1.1) [5 Pts] Mark **ALL** the correct answers. $\lg n^2 + \lg 8^n + \sqrt{n^2}$ is

☐ **a)** $\Theta(\lg n)$ ☐ **b)** $\Theta(n)$ ☐ **c)** $\Theta(\sqrt{n})$ ☐ **d)** $\Theta(n^2)$ ☐ **e)** $\Theta(2^n)$

(1.2) [5 Pts] Mark **ALL** the correct answers. $n \lg n^2 + \lg 32^n + 100n$ is

☐ **a)** $O(\lg n)$ ☐ **b)** $O(n)$ ☐ **c)** $O(n \lg n)$ ☐ **d)** $O(n^2)$ ☐ **e)** $O(2^n)$

(1.3) [5 Pts] Mark **ALL** the correct answers. $n \lg n^2 + \lg 32^n + 100n$ is

☐ **a)** $\Omega(\lg n)$ ☐ **b)** $\Omega(n)$ ☐ **c)** $\Omega(n \lg n)$ ☐ **d)** $\Omega(n^2)$ ☐ **e)** $\Omega(2^n)$

(1.4) [5 Pts] Mark **ALL** the correct answers. Consider the following recurrence

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2T(n-1) + 1 & \text{if } n > 0 \end{cases}$$

☐ **a)** Using the master method one can prove $T(n) = \Theta(2^n)$

☐ **b)** The master method cannot be used to solve the above recurrence

☐ **c)** Using the substitution method one can prove $T(n) = O(n^2)$

☐ **d)** Using the substitution method one can prove $T(n) = \Theta(2^n)$

☐ **e)** The substitution method cannot be used to solve the above recurrence

Solution 1.

(1.1) $\lg n^2 + \lg 8^n + \sqrt{n^2} = 2\lg n + n\lg 8 + n = \Theta(n)$ therefore the only correct answer is (b).

(1.2) $n\lg n^2 + \lg 32^n + 100n = 2n\lg n + n\lg 32 + 100n = \Theta(n\lg n)$. Therefore the correct answers are (c), (d), and (e).

(1.3) $n\lg n^2 + \lg 32^n + 100n = 2n\lg n + n\lg 32 + 100n = \Theta(n\lg n)$. Therefore the correct answers are (a), (b), and (c).

(1.4) Note that we gave seen this recurrence in Exercise Session 05 where we mentioned that one can prove using the substitution method that $T(n) = 2^{n+1} - 1 = \Theta(2^n)$. Clearly, the recursion is not in the format $T(n) = aT(n/b) + f(n)$ as the master method requires. Therefore the correct answers are (b) and (d).

Question 2. 20 Pts

Recognising and use properties of known algorithms

(2.1) [5 Pts] Mark **ALL** the correct statements. Consider the array $A = [9, 5, 8, 3, 4, 6, 7, 2, 1, 10]$ interpreted as a binary tree

- ☐ **a)** The height of A is 3
- ☐ **b)** If $A.\textit{heap-size} = A.\textit{length}$ then A satisfies the max-heap property
- ☐ **c)** A satisfies the binary-search tree property
- ☐ **d)** A satisfies the red-black tree property
- ☐ **e)** If $A.\textit{heap-size} = A.\textit{length} - 1$ then A satisfies the max-heap property

(2.2) [5 Pts] Mark **ALL** the correct statements. Consider a modification to QUICKSORT, called MAXQUICKSORT, such that each time PARTITION is called, the maximum element of the sub-array to partition is found and used as a pivot.

- ☐ **a)** MAXQUICKSORT worst-case running time is $\Theta(n^2)$
- ☐ **b)** MAXQUICKSORT best-case running time is $\Theta(n \lg n)$
- ☐ **c)** MAXQUICKSORT best-case running time is $\Theta(n^2)$
- ☐ **d)** MAXQUICKSORT worst-case running time is $\Omega(n \lg n)$
- ☐ **e)** Like MERGESORT, MAXQUICKSORT works in-place

(2.3) [10 Pts] The median of a sequence of numbers $a_1, \ldots, a_n$ is defined as the $\lfloor (n+1)/2 \rfloor$-th smallest element in the sequence. Consider the following algorithm to find the median of an unsorted array $A[1\,..\,n]$

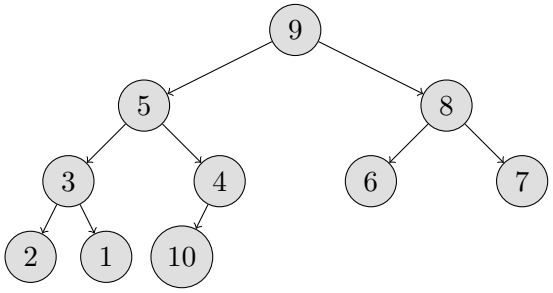
```
MEDIAN(A)
1  BUILD-MAX-HEAP(A)
2  for i = A.length downto ⌊n/2⌋
3       exchange A[1] with A[i]
4       A.heap-size = A.heap-size − 1
5       MAX-HEAPIFY(A, 1)
6  return A[1]
```

(a) Is the above algorithm correct? Argument your answer.

(b) What is the asymptotic worst-case running time of MEDIAN(A)?

Solution 2.

(2.1) The array $A = [9, 5, 8, 3, 4, 6, 7, 2, 1, 10]$ interpreted as a binary tree looks like this



Recall that the height of a tree is defined as the maximal number of edges from the root to some leaf node. Therefore the height of the above tree is 3.

When, $A.\textit{heap-size} = A.\textit{length}$ the heap A contains all the nodes in the tree. In contrast, when $A.\textit{heap-size} = A.\textit{length} - 1$ the heap A looks like the above tree without the last node (i.e., the one with key 10).

Recall that the binary search property requires that for all nodes, the left child is smaller or equal than its parent and the right child is greater than its parent. Clearly, the above tree does not satisfy the BST property, see e.g., node 4 with left child 10. To satisfy the red-black-tree property, it is necessary to first satisfy the BST property. Therefore the tree above does not satisfy the red-black tree property.

Therefore the correct answers are (a) and (e).

(2.2) MaxQuicksort select as a pivot element the max element in the subarray. Since finding the max element in an unsorted list of elements can be done in linear time, we have that both the best-case and worst-case running time of MaxQuicksort are the solution of the following recurrence.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T(n-1) + \Theta(n) & \text{if } n \geq 2 \end{cases}$$

As we have seen in the exercise sessions, one can show using the substitution method that $T(n) = \Theta(n^2)$. Finding the max element can be easily implemented *in-place* therefore MaxQuicksort, like Quicksort, works in-place. In contrast, MergeSort does not work in-place. Therefore the correct answers are (a) and (c) and (d).

(2.3) (a) The pseudocode for Median works similarly to Heapsort, but iterates the for-loop only for $i = A.\textit{length}$ down to $\lfloor n/2 \rfloor$. As for the convention used in CLRS the last iteration is performed with $i = \lfloor n/2 \rfloor$.

If we run the algorithm on the instance $A = [1, 2, 3]$ the algorithm iterates 3 times the for-loop (because $\lfloor 3/2 \rfloor = 1$) and returns 1 but the $\lfloor (n+1)/2 \rfloor$-th smallest element of A is 2. Therefore the algorithm is **not** correct.

It is important to remark that to prove correctness of an algorithm it is not enough to show that it works correctly for a small subset of its input instances, but one is required to provide a formal proof of correctness (e.g., using loop invariants or by induction).

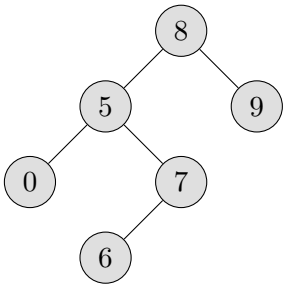
(b) The running-time of Median is $\Theta(n \lg n)$ because the procedure Build-Max-Heap takes $\Theta(n)$ and Max-Heapify takes $\Theta(\lg n)$.

Some of you noticed a similarity with the pseudocode of the Build-Max-Heap procedure because the loop iterates half the length of the array and concluded that the runtime analysis of Median could be improved to $O(n)$. This is wrong, because the analysis of Build-Max-Heap relies on the fact that the call to Max-Heapify is made on i. This is not the case for Median, where Max-Heapify is always called on the same index i.e., 1.

Question 3. 20 Pts

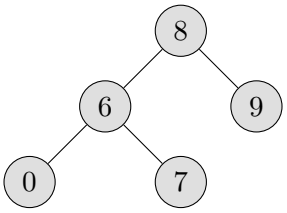
Understanding of known algorithms

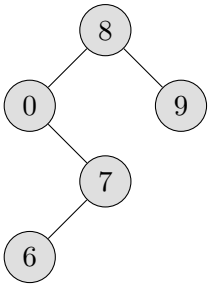
(3.1) [5 Pts] Mark **ALL** the correct statements. Consider the binary tree T depicted here below.

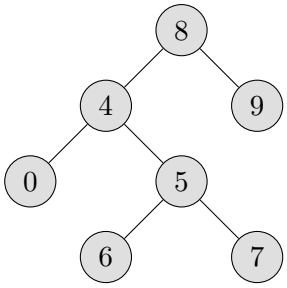


Here we refer to the implementation of Tree-Insert and Tree-Delete as described in CLRS Chapter 12.

- ☐ **a)** The tree below to the right shows the result of Tree-Insert$(T, 4)$
- ☐ **b)** The tree below in the centre shows the result of Tree-Delete$(T, 5)$
- ☐ **c)** All the trees below satisfy the binary-search tree property
- ☐ **d)** The tree below to the left shows the result of Tree-Delete$(T, 5)$
- ☐ **e)** None of the trees below represents the result of Tree-Delete$(T, 5)$







(3.2) [5 Pts] Insert the keys $10, 22, 31, 9, 15, 2, 17, 100, 59$ into a hash table (initially empty) of length $m = 11$ using *open addressing* with the auxiliary function $h'(k) = k$.

Mark the hash table resulting by the insertion of these keys using linear probing.

- ☐ **a)** $22, 9, 2, 100, 15, 59, 17,$ NIL, NIL, $31, 10$
- ☐ **b)** $110,$ NIL, NIL, $22, 31, 9, 15, 2, 17, 100, 59$
- ☐ **c)** $22, 100, 2, 59, 15,$ NIL, $17,$ NIL, $9, 31, 10$
- ☐ **d)** $22, 100, 9, 17, 15,$ NIL, $2,$ NIL, $59, 31, 10$

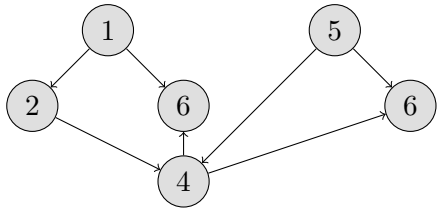
Mark the hash table resulting by the insertion of these keys using quadratic probing with $c_1 = 1$ and $c_2 = 3$.

- ☐ **a)** $22, 9, 2, 100, 15, 59, 17,$ NIL, NIL, $31, 10$
- ☐ **b)** $110,$ NIL, NIL, $22, 31, 9, 15, 2, 17, 100, 59$
- ☐ **c)** $22, 100, 2, 59, 15,$ NIL, $17,$ NIL, $9, 31, 10$
- ☐ **d)** $22, 100, 9, 17, 15,$ NIL, $2,$ NIL, $59, 31, 10$

Mark the hash table resulting by the insertion of these keys using double hashing with $h_1(k) = k$ and $h_2(k) = 1 + (k \mod (m-1))$.

- ☐ **a)** $22, 9, 2, 100, 15, 59, 17,$ NIL, NIL, $31, 10$
- ☐ **b)** $110,$ NIL, NIL, $22, 31, 9, 15, 2, 17, 100, 59$
- ☐ **c)** $22, 100, 2, 59, 15,$ NIL, $17,$ NIL, $9, 31, 10$
- ☐ **d)** $22, 100, 9, 17, 15,$ NIL, $2,$ NIL, $59, 31, 10$

(3.3) [10 Pts] Consider the directed graph G depicted below.



(a) Write the intervals for the discovery time and finishing time of each vertex in the graph obtained by performing a depth-first search visit of G. *Remark:* If more than one vertex can be chosen, choose the one with smallest label.

(b) Write the corresponding "parenthesization" of the vertices in the sense of Theorem 22.7 in CLRS

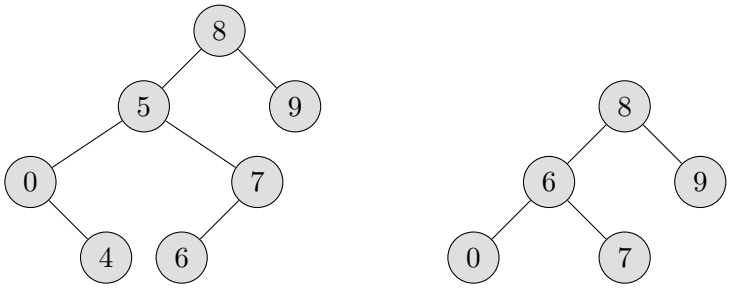
(c) Assign with each edge a label T (tree edge), B (back edge), F (forward edge), C (cross edge) corresponding to the classification of edges induced by the DFS visit performed before.

(d) If G admits a topological sorting, then show the result of TOPOLOGICAL-SORT(G).

Solution 3.

(3.1) Note that the first two trees satisfy the BST property, but the third one doe not because 6 is the left child of 5.

The results of TREE-INSERT(T,4) and TREE-DELETE($T, 5$) are respectively



Therefore the only correct answer is (d).

(3.2) The resulting hash tables are respectively:

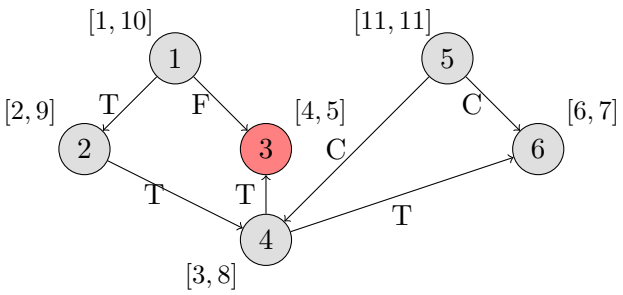
linear probing: $22, 9, 2, 100, 15, 59, 17,$ NIL, NIL, $31, 10$

quadratic probing: $22, 100, 9, 17, 15,$ NIL, $2,$ NIL, $59, 31, 10$

double hashing: $22, 100, 2, 59, 15,$ NIL, $17,$ NIL, $9, 31, 10$

(3.3) Also in this exercise there was a typo: the leftmost 6 should have been a 3 to avoid ambiguity in which order adjacent vertices should be explored. For clarity, here I solve the question both for the case where the vertex label has been replaced (case 1), and the case where the graph has been left as originally presented in the exam sheet (case 2).

(Case 1) If you have replaced the label of the vertex as suggested during the exam (the vertex is highlighted in red). (a and c) The correct answer for (a) and (c) are depicted in the graph below. There each vertex $v \in V$ is associated with the interval $[v.d, v.f]$ as computed by DFS, and each edge is labelled according to the corresponding classification.



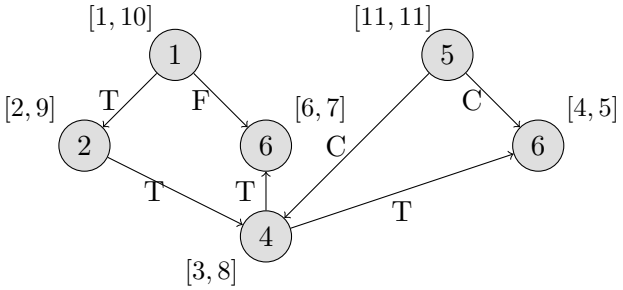
(b) the corresponding "parenthesization" of the vertices is

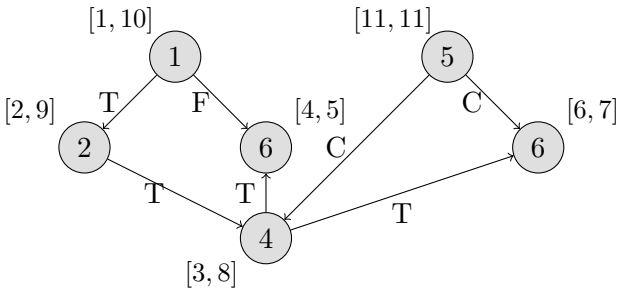
$$(1\ (2\ (4\ (3\ 3)\ (6\ 6)\ 4)\ 2)\ 1)\ (5\ 5).$$

(d) The graph admits a topological sorting because it is acyclic (this is also seen by the fact that there is no back edge in the edge classification shown before). Then, the result of Topological-Sort(G) is

$$5;\ 1;\ 2;\ 4;\ 6;\ 3\ .$$

(Case 2) If you have considered the graph G as given in the exam sheet. For the subquestion (a) there are 2 possible correct answers according to which adjacent vertex of 4 one decides to explore first. However, the two possible choices result in the same edge classification. Below are depicted the correct answers for (a) and (c).





(b) the corresponding "parenthesization" of the vertices is

$$(1\ (2\ (4\ (6\ 6)\ (6\ 6)\ 4)\ 2)\ 1)\ (5\ 5).$$

(d) The graph admits a topological sorting because it is acyclic (this is also seen by the fact that there is no back edge in the edge classification shown before). Then, the result of Topological-Sort(G) is

$$5;\ 1;\ 2;\ 4;\ 6;\ 6\ .$$

Note that the implementation of Topological-Sort(G) as presented in CLRS requires one to simply write the nodes in decreasing order of finishing time. Many have used alternative algorithms which where still correct but were giving different topological sortings.

Question 4. 30 Pts

There are $n \geq 2$ students $s_1, \ldots, s_n$ in the 2nd Semester (for simplicity, assume this list is already ordered alphabetically by name). In an effort to optimise group formation, Kurt, the semester coordinator, has asked the students to provide a list of pairs of friends, and has received m pairs of indices $(a_1, b_1), \ldots, (a_m, b_m)$, such that for $1 \leq i \leq m$, student s_{a_i} is a friend of student s_{b_i}. A pair of friends in this list is said to be *respected* if both friends end up in the same study group. Kurt wants to form exactly k study groups each containing at least 2 students, where $1 \leq k \leq n/2$. To simplify the group formation process, Kurt lines up the students in a list in alphabetical order by name, and cuts the list in $k-1$ places to form the k groups. Help him out by giving an algorithm which determines the $k-1$ locations where the list of students should be cut, such that the number of respected friendships is maximised.

Let $M(i, j)$ denote the maximum number of respected friendships if one considers only students $s_1, \ldots, s_i$ and attempts to partition just these students into exactly j groups. We can express $M(i, j)$ recursively in terms of optimal solutions to smaller problems by noting that in order to optimally partition students $s_1, \ldots, s_i$ into j groups, we must first optimally partition some prefix $s_1, \ldots, s_l$ of the first i students into $j-1$ groups, and then add a single group consisting of students $s_{l+1}, \ldots, s_i$. We therefore have

$$M(i, j) = \max_{2(j-1) \leq l \leq i-2} \{M(l, j-1) + T(l+1, i)\},$$

where $T(i, j)$ denotes the total number of friendship pairs within the set of students $s_i, \ldots, s_j$.

(4.1) [10 Pts] Describe an algorithm to compute all $T(i, j)$ for $1 \leq i, j \leq n$ and analyse the running time of your algorithm.

(4.2) [10 Pts] Assume you can compute all $T(i, j)$ for $1 \leq i, j \leq n$ in $O(n^2)$ time. Describe a dynamic programming algorithm to compute $M(i, j)$ for all $1 \leq i \leq n$ and $1 \leq j \leq k$, and analyse the running time of your algorithm.

(4.3) [10 Pts] Describe how to reconstruct the actual k groups within a solution which achieve the optimal value $M(n, k)$.

Solution 4.

(4.1) An easy way to compute $T(i, j)$ for all $1 \leq i \leq j \leq n$ works as follows. First scan over all pairs of friendships and with them construct an $n \times n$ matrix F, such that $F[\min(i, j), \max(i, j)] = 1$ if s_i and s_j are friends (i.e., if the pair (i, j) belongs to the list of m pairs given to Kurt), and 0 otherwise. Note that in this way we account for the fact that the pairs shall not be considered as ordered (because we can fairly assume that the friendship relation is symmetric). This requires $O(n^2)$ time since $m \leq n^2$. Once we have populated the matrix F, we can for each $i \leq j$, count the number of friendships in the list $s_i, \ldots, s_j$. This can be done in $O(n^4)$ time as described by the following pseudocode.

SlowComputeT(P)

```
 1  // Construct F
 2  let F be an n × n matrix initialised with 0s.
 3  for each (i, j) ∈ P
 4      F[min(i, j), max(i, j)] = 1
 5  // Compute T
 6  let T be an n × n matrix initialised with 0s
 7  for i = 1 to n
 8      for j = i to n
 9          for k = i to j
10              for h = k to j
11                  T[i, j] = T[i, j] + F[k, h]
12  return T
```

The above solution is sufficient to get full marks. Now we show that we can do better.

A faster way to compute $T(i, j)$ for all $1 \leq i \leq j \leq n$ in $O(n^2)$ time works as follows. Each $T(i, j)$ can be computed using the following recursive formula

$$T(i,j) = \begin{cases} F(i,j) + T(i+1,j) + T(i,j-1) - T(i+1,j-1) & \text{if } i \leq j, \\ 0 & \text{if } i > j. \end{cases} \tag{1}$$

The formula works in this way. The number of friendship pairs within $s_i, \ldots, s_j$ can be retrieved by summing up the number of friendship pairs in the sub-lists $s_{i+1}, \ldots, s_j$ and $s_i, \ldots, s_{j-1}$, from which we subtract the number of friendship pairs in the overlapping sub-list $s_{i+1}, \ldots, s_{j-1}$, which otherwise would be counted twice. At this point the only pair that we are missing to count is the pair (i, j). When $i > j$ the list $s_i, \ldots, s_j$ is empty, thus $T(i, j) = 0$.

We can then compute $T(i, i)$ for all i, then $T(i, i + 1)$ for all i, then $T(i, i + 2)$ for all i, and so on. By observing the dependency graph induced by the formula (1), it will only take $O(1)$ time to evaluate the formula when computing $T(i, j)$, as we will already have computed $T(i + 1, j)$, $T(i, j - 1)$, and $T(i + 1, j - 1)$ before.

Let $P[1\,..\,m] = [(a_1, b_1), \ldots, (a_m, b_m)]$ be the array of friendship pairs, the pseudocode below returns an $n \times n$ array T

FastComputeT(P)

```
 1  // Construct F
 2  let F be an n × n matrix initialised with 0s.
 3  for each (i, j) ∈ P
 4      F[min(i, j), max(i, j)] = 1
 5  // Compute T
 6  let T be an n × n matrix initialised with 0s
 7  for k = 0 to n − 1
 8      for i = 1 to n − k
 9          j = i + k
10          T[i, j] = F[i, j] + T[i + 1, j] + T[i, j − 1] − T[i + 1, j − 1]
11  return T
```

As said before, constructing F (lines 1–4) takes $O(n^2)$ time. Executing line 6 takes $O(n^2)$ time, while executing the for-loop (lines 7–10) takes $O(n^2)$.

(4.2) After first computing the all $T(i,j)$ values, we can compute $M(i,j)$ in sequence as follows: first we compute $M(i,1)$ for all is, then $M(i,2)$ for all is, and so on. The following procedure takes an array of pairs P, the number of students $n \geq 2$, the number of groups $k \leq n/2$, and computes $M(i,j)$ for $i \in \{2,..,n\}$ and $j \in \{1,\ldots,k\}$. It returns the matrix representing M and another matrix S encoding the optimal cuts performed to divide the students in groups.

```
BOTTOM-UP-GROUPFORMATION(P, n, k)
 1  T = FASTCOMPUTET(P)
 2  let M[0..n, 0..k] be an (n+1) × (k+1) matrix initialised with 0s
 3  let S[1..n, 1..k] be an n × k matrix
 4  for j = 1 to k
 5      for i = 2 to n
 6          // Compute M(i,j) = max_{2(j-1)≤l≤i-2}{M(l, j-1) + T(l+1, i)}
 7          q = −∞
 8          for l = 2(j − 1) to i − 2
 9              if q < M[l, j − 1] + T[l + 1, i]
10                  q = M[l, j − 1] + T[l + 1, i]
11                  S[i, j] = l
12          M[i, j] = q
13  return M and S
```

The overall running time of the above procedure is $O(kn^2)$, since computing T takes $O(n^2)$, initialising M and S respectively take $O(n^2)$ and $O(kn)$, and the three nested for-loops overall take $O(kn^2)$.

One can alternatively implement a top-down solution of the group formation problem implementing memoization.

(4.3) The following procedure takes an array of pairs P, the number of students $n \geq 2$, the number of groups $k \leq n/2$, and prints a list of index intervals each representing a group.

```
PRINTGROUPS(P, n, k)
1  (M, S) = BOTTOM-UP-GROUPFORMATION(P, n, k)
2  l = n
3  for j = k downto 1
4      print (S[l, j] + 1, l)
5      l = S[l, j]
```

An alternative solution may use a recursive procedure, as described below.

```
PRINTGROUPS(P, n, k)
1  (M, S) = BOTTOM-UP-GROUPFORMATION(P, n, k)
2  PRINTGROUPSAUX(S, n, k)

PRINTGROUPSAUX(S, n, k)
1  if k ≥ 1
2      PRINTGROUPSAUX(S, S[n, k], k − 1)
3      print (S[n, k] + 1, n)
```

Question 5. 20 Pts

Six degrees of separation is the idea that all people are six, or fewer, social connections away from each other. As a result, a chain of "a friend of a friend" statements can be made to connect any two people in a maximum of six steps. This concept has been ironically re-proposed as a game called "Six degrees of Kevin Bacon" which challenges people to link any arbitrary actor to Kevin Bacon linking the two through their film roles within six steps. The smallest number of steps from an actor a to Kevin Bacon is known as *the Bacon number for* a.

(5.1) [10 Pts] Assume you are given a graph $G = (V, E)$ representing the Hollywood film industry network of collaborations, where $V = \{v_1, \ldots, v_n\}$ represents all Hollywood actors, and two actors $v_i, v_j \in V$ are connected by an edge $\{v_i, v_j\} \in E$ if and only if v_i, v_j have played in the same movie. Describe an $O(|V| + |E|)$ algorithm that determines whether all actors in V are at most six degrees of separation from Kevin Bacon.

(5.2) [10 Pts] Consider now a weighted variant of $G = (V, E)$ which takes into consideration the unpopularity of the movies used to link Bacon to other actors. The weight $w(i, j)$ is defined as the lowest score given by Rotten Tomatoes (scores range from 0 to 100) for a movie where both actors v_i and v_j played a role ($w(v_i, v_j) = \infty$ if the two actors never played a role in the same movie). A *worst collaboration path* between two actors, is a path of minimal weight in G. Describe an algorithm which finds an actor with highest valued worst collaboration path among those actors who have finite Bacon number (i.e., actors who have Bacon number $n < \infty$). Analyse the running time of your algorithm.

Solution 5.

(5.1) Let $s \in V$ be the vertex representing Kevin Bacon. The algorithm first applied BFS with source vertex s, then checks if all vertices v have attribute $v.d \leq 6$. The call to BFS takes $O(|V| + |E|)$ time, and checking the distance attributes takes $O(|V|)$. Overall the algorithm takes $O(|V| + |E|)$ time in the worst-case. The pseudocode corresponding to the procedure described above is

```
SIX-DEGREES(G, s)
1  BFS(G, s)
2  for each v ∈ G.V
3       if v.d > 6
4            return FALSE
5  return TRUE
```

(5.2) We can find the actor with highest valued worst collaboration path as follows.

(a) first we solve a sigle-pair shortest-paths problem of the weighed graph G with source vertex $s \in V$ representing the actor Kevin Bacon. Since the graph has nonnegative weights we can use Dijkstra's algorithm.

(b) At this point for each vertex $v \in V$ the attribute $v.d$ represent the value of a worst collaboration path from Kevin Bacon. Thus we can select, among the vertices who have $v.d < \infty$, one which has maximal $v.d$.

This is done by the following pseudocode

```
HighestWorstCollaborator(G, w, s)
1  Dijkstra(G, w, s)
2  h = 0 // Stores the highest value
3  c = s // Stores the corresponding actor
4  for each v ∈ G.V
5       if v.d < ∞ and v.d > h
6            h = v.d
7            c = v
8  return c
```

The running time of HighestWorstCollaborator(G, w, s) depends on the way one implements the min-priority queue used by Dijkstra's algorithm. In class we have only considered an implementation of the min-priority queue by using a simple array. Thus the call Dijkstra(G, w, s) takes $O(V^2)$. The initialisation of the for-loop takes constant time, while the for loop takes time $\Theta(V)$. Therefore, the running-time of HighestWorstCollaborator is $O(V^2)$.